Reasoning about the Properties of an Enterprise Information System

John A. van der Poll[1] and Paula Kotzé[2]

School of Computing
University of South Africa
[1]vdpolja@unisa.ac.za [2]kotzep@unisa.ac.za

Abstract. A condensed specification of a multi-level marketing enterprise in the Z specification language is presented and a number of proof obligations that result from operations on the state is stated. The feasibility of using certain reasoning heuristics for discharging proof obligations emerging from the specification is investigated and we show how two important proof obligations arising from the specification of a real-life enterprise may successfully be discharged using a suite of well-chosen heuristics.

1 Introduction

Among the benefits to be gained by using a formal specification language like Z [1] is that the specifier can prove things about the specification. The process of constructing proofs can aid in the understanding of the system and may reveal hidden assumptions [2].

The huge cost and inconvenience of detecting and correcting errors only after the system has been released [3], justifies the effort to identify and correct errors at an early stage (e.g. specification phase). However, the readiness with which the information technology industry would accept such a methodology is likely to depend on the availability of environments that ease the burden on the specifier by automating much of the specifier's tasks. The more sophisticated the environment (and thus the greater its contribution to the partnership), the more natural becomes the inclusion of a reasoning algorithm as one component.

Reasoning about the properties of an enterprise information system at the specification level may, however, be a non-trivial task owing to the size of the system or the complexity of the structures that make up such a system. Accounts of costly, yet failed proof attempts exist. Mokkedem et al. [4] report that an attempt to generate a proof monolithically in one step from a stated property to a protocol was prohibitively difficult. The one step proof was abandoned, unfinished, after 18 months of effort which led to the specifiers eventually adopting an incremental proof strategy.

Since Z is based on first-order logic and a strongly typed fragment of Zermelo-Fraenkel (ZF) set theory, it makes sense to investigate to what extent a set of heuristics [5] for proving theorems in set theory may be used to reason about the Z specification of a multi-level marketing enterprise.

1.1 Why Reasoning Heuristics?

Traditionally set-theoretic proofs pose demanding challenges to automated reasoning programs [6, 7], since unlike number theory or group theory or applications to real systems such as power stations, the denotations of terms in the context of set theory are strongly hierarchical: one object (perhaps at a very fine level of granularity) is a member of another (coarser) object, which in turn may be a member of a higher-level (even coarser) object, and so on. The possibility of moving between levels is a provocation to much irrelevant activity; intelligence would be realised by heuristics that limit the movement up or down to productive changes of granularity [8].

It is furthermore an open problem as to which inference rule would build set theory into a theorem prover the same way as paramodulation builds in equality-oriented reasoning [9]. Paramodulation is a rule applied to a pair of clauses and requiring that at least one of the two contains a positive *equality* literal, and yielding a clause in which an equality substitution corresponding to the equality literal has occurred. The object of an application of paramodulation is, therefore, to cause an equality substitution to take place from one clause into another.

Devising a set of heuristics appears to be the best strategy for reasoning about set-theoretic constructs [8]. Such a set of heuristics was developed by one of the authors [5] and in this paper we investigate to what extent these heuristics are useful for reasoning about the properties of a franchise or a multi-level marketing enterprise [10].

1.2 Structure of this Paper

Section 2 presents a brief overview of OTTER [11], the automated reasoner used in this work. A number of heuristics for reasoning about set-theoretic structures is presented in Sect. 3. A brief Z specification of a generic multi-level marketing enterprise [10] is given in Sect. 4. Some applications of the said heuristics are illustrated in Sect. 5 where two proof obligations (POs) are stated and discharged using an automated reasoner. A summary and some ideas about future work conclude this paper.

2 The OTTER Theorem Prover

OTTER (Organized Techniques for Theorem Proving and Effective Research) [11] is a resolution-based theorem-proving program for first-order logic with equality and includes the inference rules binary resolution, hyperresolution (both positive and negative versions), UR-resolution and binary paramodulation. OTTER was written and is distributed by William McCune at the Argonne National Laboratory in Illinois.[1]

[1] At the time of writing the latest version of OTTER is available at: http://www-unix.mcs.anl.gov/AR/otter.

OTTER can convert first-order formulae into sets of clauses, which constitute the input to the resolution algorithm. Of course, OTTER cannot accept formulae in the highly evolved notation of set theory so the user has to rewrite set-theoretic formulae in terms of a weaker first-order language having the relevant relations and functions as predicate symbols and function symbols in its alphabet. Some other capabilities of OTTER are factoring and weighting. The purpose of a weight clause is give a weight to variables or terms and if such weight is chosen sufficiently high then the generation of too many irrelevant paramodulants is effectively blocked. Note, however, that the use of a weight leads to an incomplete search strategy.

An OTTER program is divided into several sections, each such section made up of first-order formulae or clauses (an exception is the section containing the optional demodulators which must be in clausal form already). The most important sections are the *usable list* and the *set-of-support* (sos) list. It is customary to place the negation of the theorem to be proven in the sos and the rest of the information in the usable list.

Next we introduce a number of heuristics for reasoning about set theory. These heuristics were developed to address the problems discussed in Sect. 1.1.

3 Set-Theoretic Reasoning Heuristics

The heuristics presented in this section are detailed in [5, 12] and have been developed empirically through observing the behaviour of, as well as studying the format of the clauses generated by the reasoner during a proof attempt. In total 14 heuristics were developed and we briefly discuss some prominent ones below:

1. *Weight strategy*: Use the setting `weight(x,n)`, for `n` $\in \{3,4,5\}$, whenever the sos consists of the negation of an equality literal. Equality reasoning with paramodulation generally results in the generation of many irrelevant clauses. Assigning a weight of `n` to all variables avoids the generation of too many irrelevant paramodulants. Empirically we found a weight of 3, 4 or 5 to be sufficient.
2. *Extensionality*: Use the principle of extensionality to replace an equality in the sos with the condition under which two sets are equal, i.e. whenever their elements are the same.
3. *Nested functors*: Avoid, if possible, the use of *nested functor* symbols in definitions. Terms built up with the aid of function symbols (called functors) are more complex, potentially leading to difficulties with unification of terms, especially when these functors are nested inside other structures.
4. *Divide and Conquer*: Perform two separate subset proofs whenever the problem at hand requires one to prove the equality of two sets. An equality in the sos implies (via Extensionality) an ‘if and only if’. Hence a specifier may opt for two proofs, one for the only-if part and another for the if part.
5. *Multivariate functors*: Make terms in sets as simple as possible — either not involving functors at all, or else involving functors with the minimum number

of argument positions taken up by variables. The more variables occur as arguments to a functor, the greater the likelihood of thrashing caused by the unification of these variables with other terms.

6. *Intermediate structures*: Avoid complex functor expressions by using an indirect definition for an internal structure whenever this appears less likely to produce complex functor expressions than the direct definition. In practice we simply give a name to a complex structure that is nested inside another structure and then define the inner structure externally on its own, instead of unfolding its definition directly inside the enclosing structure.
7. *Element structure*: Define the elements of relations and functions directly in terms of ordered pairs or ordered n-tuples whenever the tuples need to be opened to find a proof. An ordered n-tuple is an example of a functor and projecting out the coordinates of the tuple often avoids the various functor problems listed above.
8. *Search-guiding*: Generate and use half definitions, via the technique of resolution by inspection, for biconditional formulae in the usable list whenever the sos consists of a conditional formula or a single literal. A half definition is an implication (e.g. only-if) as opposed to an if and only if definition. Through inspection it is often possible to trace the initial steps a reasoner would perform starting with the conditional formula in the sos. Hence it is possible to predict which half of some definitions in the usable list would probably be needed and which 'other halves' are redundant.
9. *Inference rule selection*: Use `set(neg_hyper_res)` in the place of positive hyperresolution whenever the combined use of `set(hyper_res)` and `set(ur_res)` rapidly makes the sos empty. If no rapid proof results, try binary resolution. Both forms of hyperresolution are capable of generating homogeneous clauses only (i.e. just positive or just negative but not mixed). Although many researches warn against the use of binary resolution [7] we found such rule to be occasionally useful (see Sect. 5 below).
10. *Resonance*: Attempt to give corresponding terms in formulae a syntactically similar structure to aid the resolution process [13]. Not only does this apply to terms just in the usable list, but also to a term in the sos and a corresponding term in the usable list.

4 A Multi-level Marketing Enterprise

A multi-level marketing (MLM) enterprise [10] markets consumable products through people as follows: A new *distributor* registers with the enterprise either as a direct associate of the company, or under an existing distributor called an *upline*. Both the upline (also called the sponsor) and the new distributor (now called a downline) then go on to each sponsor more new distributors, and so on. In this way a network of distributors of the products of the company is built. Hence, a MLM structure can be modelled by forests and trees [14].

Distributors buy products from the company and every product carries a *point value* (pv) as well as a *business value* (bv). The business value is directly

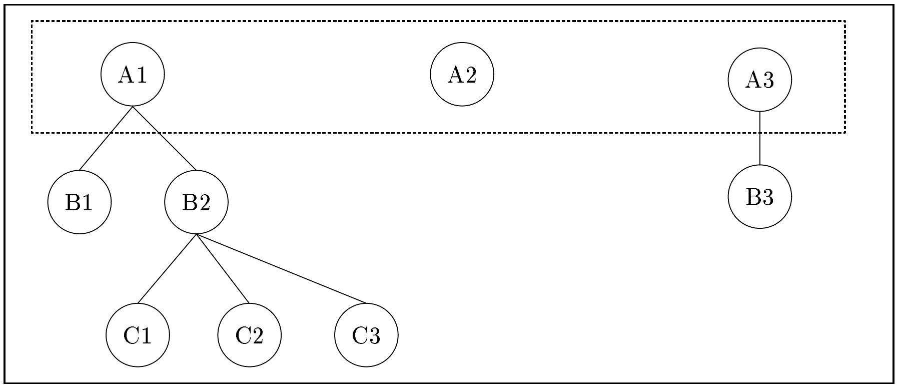

Fig. 1. An example network

related to the price of the product. Both the points and the business values are accumulated per distributor throughout a calendar month. At the end of the month the total business value in the network for each distributor is calculated, and the distributor is paid (in the appropriate currency) a certain percentage (determined by the pv) of the total business value for his or her group. This is called a *bonus*.

A small MLM network is shown in Fig. 1. Distributors $A1$, $A2$ and $A3$ associated with the company directly are called the roots of the forest (or network in MLM terms).

The state of our MLM enterprise is (\ represents set-theoretic difference):

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{MLM}} \\
\hline
\mathit{known} : \mathbb{P}\,\mathit{ID} \\
\mathit{NRoots} : \mathbb{P}\,\mathit{ID} \\
\mathit{NUplines} : \mathit{ID} \leftrightarrow \mathit{ID} \\
\mathit{NDist} : \mathit{ID} \mathrel{\mkern8mu\mapstochar\mkern-8mu\rightarrow} \mathit{Name} \times \mathit{Address} \times \mathit{PV} \times \mathit{BV} \times \mathit{Bonus} \\
\hline
\mathit{known} = \mathrm{dom}\,\mathit{NDist} \\
\mathrm{dom}\,\mathit{NUplines} \cup \mathrm{ran}\,\mathit{NUplines} \subseteq \mathit{known} \\
\mathit{NRoots} = \mathit{known} \setminus \mathrm{ran}\,\mathit{NUplines} \\
\mathit{Inj}(\mathit{NUplines}) \\
\hline
\end{array}
$$

The set *known* contains the identity codes of all distributors in the system. *NRoots* represents all root distributors. The relation *NUplines* represents the network of distributors while the function *NDist* represents a mapping from a unique identity code to the particulars for that distributor. *NDist* is not necessarily injective since two (or more) distributors may have the same particulars (i.e. name, address, etc.). Every distributor has at most one upline, captured by the following general definition of injectivity:

$$(\forall\, R)(\mathit{Inj}(R) \leftrightarrow (\forall\, i)(\forall\, j)(\forall\, k)(\,((i,k) \in R \wedge (j,k) \in R) \longrightarrow (i = j))) \qquad (1)$$

The following operation registers a new distributor $p!$ below an existing one, $q?$:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\, \mathit{Register\_with\_upline} \,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta \mathit{MLM} \\
p!, q? : \mathit{ID} \\
\mathit{name}? : \mathit{Name};\ \mathit{addr}? : \mathit{Address} \\
\hline
p! \notin \mathit{known} \land q? \in \mathit{known} \\
\mathit{known}' = \mathit{known} \cup \{p!\} \\
\mathit{NUplines}' = \mathit{NUplines} \cup \{q? \mapsto p!\} \\
\mathit{NDist}' = \mathit{NDist} \cup \{p! \mapsto (\mathit{name}?, \mathit{addr}?, 0, 0.0, 0.0)\} \\
\hline
\end{array}
$$

A new identity code $p!$ is generated by the system and the new distributor is linked to $q?$ in $\mathit{NUplines}'$. Initial product information pertaining to $p!$ is reflected in the personal pv being 0 and both the business value and potential bonus equal to the real value 0.0.

An order placed by a distributor is given by:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\, \mathit{Order} \,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta \mathit{MLM} \\
\mathit{id}? : \mathit{ID};\ \mathit{pv}? : \mathit{PV};\ \mathit{bv}? : \mathit{BV} \\
\hline
\mathit{id}? \in \mathit{known} \\
(\exists\, \mathit{pv} : \mathit{PV};\ \mathit{bv} : \mathit{BV} \bullet \\
\quad \mathit{pv} = \mathit{third}(\mathit{NDist}(\mathit{id}?)) + \mathit{pv}? \land \\
\quad \mathit{bv} = \mathit{fourth}(\mathit{NDist}(\mathit{id}?)) + \mathit{bv}? \land \\
\quad \mathit{NDist}' = \mathit{NDist} \oplus \{\mathit{id}? \mapsto \\
\qquad (\mathit{first}(\mathit{NDist}(\mathit{id}?)), \mathit{second}(\mathit{NDist}(\mathit{id}?)), \\
\qquad\ \mathit{pv}, \mathit{bv}, \mathit{fifth}(\mathit{NDist}(\mathit{id}?)))\}) \\
\hline
\end{array}
$$

The functions *first*, *second*, etc. project out an element at the appropriate position in the tuple. NDist' is obtained from *NDist* by replacing the tuple with first coordinate $\mathit{id}?$ as specified above. Many additional operations may be defined on the state but are beyond the scope of this paper. The interested reader is referred to [15].

Next we show how some of the heuristics introduced in Sect. 3 may be used to successfully discharge two proof obligations that arise from the *MLM* specification where otherwise proofs are not easily arrived at.

5 Reasoning about the Specification

Showing $\mathit{NRoots} = \mathit{known}' \setminus \mathbf{ran}\, \mathit{NUplines}'$. Normally in Z a correct operation is assumed to preserve the invariant. Nevertheless, a specifier may want to verify the following as a postcondition of schema *Register_with_upline* (Note that $\mathit{NRoots}' = \mathit{NRoots}$):

$$\mathit{NRoots} = \mathit{known}' \setminus \operatorname{ran} \mathit{NUplines}' \tag{2}$$

In effect the above predicate claims that the set of root elements is still equal to the new set of all distributors ($known'$) minus the new set of all downline distributors ($\operatorname{ran} NUplines'$). If we define $NewRoots = known' \setminus \operatorname{ran} NUplines'$ and pose the negation of the following equality in the sos

$$NRoots = NewRoots \tag{3}$$

then OTTER finds no proof in 20 minutes using a weight of 3, 4 or 5 and either positive or negative hyperresolution. Since neither form of hyperresolution is able to find a proof, we apply our *inference rule selection* heuristic and resort to binary resolution but still using our weight template. Now the reasoner finds a proof after just *0.66 seconds*.

Why does the reasoner fail to find a proof for (3) using hyperresolution? The sos format (3) requires the axiom of Extensionality [16]

$$(\forall A)(\forall B)[(\forall x)(x \in A \leftrightarrow x \in B) \rightarrow (A = B)] \tag{4}$$

to 'open' the equality in terms of elementhood to (loosely speaking) arrive at the following form of (3):

$$(\forall x)(x \in NRoots \leftrightarrow x \in NewRoots) \tag{5}$$

The negation of (5) clausifies into:

$$\$c1 \in NRoots \vee \$c1 \in NewRoots \tag{6}$$

$$\$c1 \notin NRoots \vee \$c1 \notin NewRoots \tag{7}$$

Formula (2) is unfolded in first-order notation as

$$(\forall x)(x \in NRoots \leftrightarrow x \in known' \wedge x \notin \operatorname{ran}(NUplines'))$$

and it clausifies into

$$x \notin NRoots \vee x \in known' \tag{8}$$

$$x \notin NRoots \vee x \notin \operatorname{ran}(NUplines') \tag{9}$$

$$x \in NRoots \vee x \notin known' \vee x \in \operatorname{ran}(NUplines') \tag{10}$$

Note that positive hyperresolvents can be generated by resolving the sos clause (6) with (8), but the sos clause (7) is not capable of generating a positive hyperresolvent with any of the clauses (8) - (10). The result is that a proof attempt using positive hyperresolution cannot start off correctly. A similar problem occurs with negative hyperresolution. Binary resolution creates no such problem, since binary resolvents may be mixed.

Still with this proof attempt, suppose a specifier is initially, due to the weight clause, concerned about an incomplete search for a proof. If we omit the weight template in the above binary resolution proof then the reasoner again finds no proof in 20 minutes (as opposed to a proof in 0.66 seconds). This forms the basis for a further heuristic that may be applied to our last failed proof attempt.

In the proof of (3) we unfolded the predicate $NUplines' = NUplines \cup \{q? \mapsto p!\}$ in schema *Register_with_upline* into an 'OTTER-like' notation as

$$(\text{all } x)(El(x, NUplines') \leftrightarrow El(x, NUplines) \mid El(x, Sin(ORD(q?, p!)))) \quad (11)$$

using the following first-order definition for a singleton:

$$(\forall\, x)(\forall\, y)(x \in Sin(y) \leftrightarrow x = y) \quad (12)$$

Together with definition (11), we also needed the following fact about ordered pairs from [16]:

$$(\forall\, u)(\forall\, v)(\forall\, w)(\forall\, x)(ORD(u, v) = ORD(w, x) \leftrightarrow ((u = w) \wedge (v = x))) \quad (13)$$

Upon studying the clauses generated by the search for a proof, we note that (12) and (13) interact to generate literals of the form $El(ORD(x, y), Sin(ORD(u, v)))$ where x, y, u and v are variables. This literal contains *nested functors*, a practice discouraged by our heuristic #3, since it, in the absence of a weight template, leads to a large number of unnecessary unifications.

If we, therefore, rewrite (11) as

$$(\text{all } x)(El(x, NUplines') \leftrightarrow El(x, NUplines) \mid (x = ORD(q?, p!))) \quad (14)$$

and still omit the weight template, then OTTER again finds a proof for (3), but in *3.70 seconds*. According to our *element structure* heuristic #7 we can further rewrite (14) as

$$(\forall\, y)(\forall\, z)$$
$$(ORD(y, z) \in NUplines' \leftrightarrow (ORD(y, z) \in NUplines \vee (y = q? \wedge z = p!))) \quad (15)$$

which cuts the execution time of 3.70 seconds down to just *0.06 seconds*.

Cardinality proof. After the execution of operation *Register_with_upline* we expect the following to hold regarding the cardinality of the set $known' = known \cup \{p!\}$:

$$\#known' = \#known + 1 \quad (16)$$

We use the following two definitions of cardinality ($Card(A, n)$ denotes $\#A = n$)

$$(\forall\, A)(Card(A, 0) \leftrightarrow A = \varnothing) \quad (17)$$
$$(\forall\, A)(\forall\, n)(Card(A, n + 1) \leftrightarrow (\exists\, x)(x \in A \wedge Card(A - \{x\}, n))) \quad (18)$$

Suppose we start with the precondition $Card(known, n)$ and pose the following question in the sos:

$$-Card(known', n + 1) \quad (19)$$

OTTER finds no proof for (19) in 30 minutes and closer investigation reveals that the term $Card(A - \{x\}, n)$ above contains nested functors (i.e. a singleton definition inside a set difference inside the functor *Card*), a practice discouraged by our *nested functor* heuristic. As a first step we unfold definition (18) as:

$$\begin{aligned}&(\forall A)(\forall n)(Card(A, n+1) \leftrightarrow \\ &\quad (\exists B)(\exists x)(x \in A \wedge Card(B, n) \wedge (\forall y)(y \in B \leftrightarrow y \in A \wedge y \notin \{x\}))) \qquad (20)\end{aligned}$$

With this unfolding OTTER still finds no proof, but since such unfolding is in turn against the recommendation put forward by the *intermediate structure* heuristic we replace the definition of set B in (20) with

$$(\forall y)(y \in B \leftrightarrow y \in DIFF(A, \{x\})) \qquad (21)$$

where x is still existentially quantified as in (20) and *DIFF* is defined by:

$$(\forall x)(x \in DIFF(known', \{p\}) \leftrightarrow x \in known' \wedge x \notin \{p\}) \qquad (22)$$

With these definitions OTTER finds a short proof for (19) in just *0.21 seconds*. Definition (22) is in line with our *multivariate functor* heuristic which advocates cutting down on the number of variables as arguments of functors. This is mainly the reason why the nested functor in definition (21) turns out to be harmless. For example, if we rewrite (22) above as

$$(\forall A)(\forall p)(\forall x)(x \in DIFF(A, \{p\}) \leftrightarrow x \in A \wedge x \notin \{p\}) \qquad (23)$$

then OTTER again finds no proof in 20 minutes. Replacing one of the variables (say A) in (23) above with a constant again helps OTTER to find a proof in *8.59 seconds*.

We may also fit our *search-guiding* heuristic onto the last definition of *Card* above. The technique of resolution by inspection reveals that the sos question (19) needs just the 'if-direction' of (20). If we make such adjustments we can even find a proof using (23), but in *1 minute 27 seconds*.

6 Summary and Future Work

This paper illustrated how some set-theoretic reasoning heuristics previously developed may be used to discharge two proof obligations that arise from the specification of a multi-level marketing enterprise. We showed that the same PO may be discharged in more than one way. This is significant, since if a particular heuristic fails to deliver then another one may be applied instead. The full suite of heuristics defined in Sect. 3 have been shown to be useful in reasoning about the properties of an extended version of the Information Enterprise described in Sect. 4 of this paper. Details appear in [12].

A number of problem areas, however, remain: Our enterprise model is inherently recursive, resulting in the reasoner experiencing difficulty when reasoning about recursive structures. For example, using our traditional definitions of cardinality (17) and (18) allows the reasoner to easily prove that the cardinality of the empty set is 0, or the cardinality of a singleton equals 1. If we, however, pick a set with two elements, say $X = \{2, 3\}$, then OTTER fails to find a proof of the property $\#X = 2$ in 20 minutes using any of our heuristics listed above. More work will have to be undertaken to successfully guide the reasoner through the minefield of recursion.

Further empirical work is also called for to scale up the proofs reported on in this paper to industrial sized proof attempts and we anticipate that additional heuristics would have to be developed to address the challenges that may unfold from such experiments.

References

[1] Bowen, J.P.: Z: A formal specification notation. In Frappier, M., Habrias, H., eds.: Software Specification Methods: An Overview Using a Case Study. FACIT. Springer-Verlag (2001) 3 – 19

[2] Woodcock, J., Davies, J.: Using **Z**: Specification, Refinement, and Proof. Prentice-Hall, London (1996)

[3] Fagan, M.: Design and code inspections to reduce errors in program development. IBM Systems Journal **15** (1976) 182 – 211

[4] Mokkedem, A., Hosabettu, R., Jones, M.M., Gopalakrishnan, G.: Formalization and proof of a solution to the PCI 2.1 bus transaction ordering problem. Formal Methods in Systems Design **16** (2000) 93 – 119

[5] van der Poll, J.A., Labuschagne, W.A.: Heuristics for Resolution-Based Set-Theoretic Proofs. South African Computer Journal **Issue 23** (1999) 3 – 17

[6] Boyer, R., Lusk, E., McCune, W., Overbeek, R., Stickel, M., Wos, L.: Set Theory in First-Order Logic: Clauses for Gödel's Axioms. Journal of Automated Reasoning **2** (1986) 287 – 327

[7] Quaife, A.: Automated Development of Fundamental Mathematical Theories. Automated Reasoning Series. Kluwer Academic Publishers (1992)

[8] Bundy, A.: A Survey of Automated Deduction. Technical Report EDI-INF-RR-0001, Division of Informatics, University of Edinburgh (1999)

[9] Wos, L., Pieper, G.W.: A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning. World Scientific Publishing Company (1999)

[10] Golden Neo-Life Diamite International: Distributor Business Guide: The Business Plan. (1997)

[11] McCune, W.W.: OTTER 3.3 Reference Manual. Argonne National Laboratory, Argonne, Illinois. (2003) ANL/MCS-TM-263.

[12] van der Poll, J.A.: Automated Support for Set-Theoretic Specifications. PhD thesis, University of South Africa (2000)

[13] Wos, L.: The Resonance Strategy. Computers and Mathematics with Applications **29** (1995) 133 – 178 (Special issue on Automated Reasoning).

[14] Scheurer, T.: Foundations of Computing : System Development with Set Theory and Logic. International Computer Science Series. Addison-Wesley (1994)

[15] van der Poll, J.A., Kotzé, P.: A Multi-level Marketing Case Study: Specifying Forests and Trees in Z. South African Computer Journal **Issue 30** (2003) 17 – 28

[16] Enderton, H.: Elements of Set Theory. Academic Press, Inc. (1977)