A Multi-level Marketing Case Study: Specifying Forests and Trees in Z

John A. van der Poll[a] Paula Kotzé[b]

School of Computing, University of South Africa
vdpolja@unisa.ac.za[a] kotzep@unisa.ac.za[b]

Abstract

*A formal specification of a multi-level marketing (MLM) business is presented. Specifying a MLM business boils down to specifying properties of and operations on mathematical forests and trees. The usefulness of the model-based specification language, **Z**, is investigated as a vehicle for a formal specification of these recursive structures. The specification is presented following a prescribed format, namely the Established Strategy for constructing a **Z** specification. The Established Strategy is augmented with the notion of proof aimed at corroborating the correctness of critical parts of a specification. We show how attempts at discharging two different proof obligations using the resolution-based, first-order theorem prover OTTER calls for the use of two automated reasoning strategies, namely avoiding equality and using resonance.*

Keywords: *automated reasoning, Established Strategy, formal specification, multi-level marketing, OTTER, precondition calculation, resolution, schema,* Z
Computing Review Categories: *D.2.4, F.3.1, F.4.1, I.2.3*

1 Introduction

A case study of a multi-level marketing (MLM) business is conducted using **Z** [13]. A MLM business may be modelled by a mathematical *forest* made up of various *trees*. The study of forests and trees [2] is well established in Computer Science and these structures have been used to describe various entities, for example, the specification of a text editor [12] in ordinary Zermelo-Fraenkel set theory [6]. However, as far as we are aware, such structures have never been used to specify a MLM business (modelled by forests and trees) and its various intrinsic operations in **Z**. Owing to their recursive nature, forests and trees possess rather unique properties and present a challenging opportunity for specifying these in Z.

The case study is cast in the format of the Established Strategy [3], distilled from work by John Wordsworth [20, 21] as well as Jim Woodcock and Jim Davies [19]. The format of this strategy is presented below.

A lucrative feature of a formal specification is that the specifier can reason about the properties of the specification. One can show that the specification has certain desirable features, or that certain undesirable consequences are absent from the specification. Apart from stating a proof obligation that an initial state exists, the standard Established Strategy for presenting a **Z** specification makes no mention of the idea of discharging proof obligations that may arise from additional system operations defined on the state. Hence we augment the strategy in a small way by insisting that critical proof obligations that arise from the specification, be discharged. Various reasoning mechanisms are available to a specifier and in Section 7 we state two proof obligations that arise from the definitions of two different operations and show how these proof obligations can be discharged using the OTTER theorem prover [8].

Our MLM specification presented in this paper gives rise to a rather large state space which in turn poses demanding challenges to a resolution-based automated reasoning assistant when reasoning about the properties of state components. We show that proof attempts of two rather simple properties call for the application of two important reasoning strategies, namely avoiding equality and using resonance. These strategies are only two of many. Other strategies for aiding the operation of a resolution-based reasoning assistant have been developed and presented in [17].

The layout of the paper is as follows: In Section 2 we present the generic format for a **Z** specification as proposed by the Established Strategy, followed by a background discussion of a typical MLM business based on forests and trees in Section 3. A natural language requirements definition of the specification, followed by the basic types and abstract state space of the system appear in Section 4. Partial definitions of four system operations are presented in Section 5 and the specification is strengthened in Section 6 according to the proposed strategy. In Section 7 two correctness proofs, one arising from a precondition calculation and the other related to the recursive nature of the specification, are discharged. Enhancements to the specification are presented in Section 9 while pointers for future work are established in Section 10.

2 The Established Strategy

The Established Strategy for presenting a specification document in **Z** is presented in [3] and stems from work done by Wordsworth [20, 21] as well as Woodcock et al. [19]. It proposes that a **Z** specification be presented as follows:

- Define all global constants and basic types, and give a natural language description of these.
- Present the abstract state space, using the constants and basic types above.
- Give an initial state of the system and prove that such a state exists.
- Introduce partial definitions of each of the system operations, together with a short informal description of each.
- Calculate the preconditions of the abstract operations on the state. Check the description of each abstract operation to ensure that its precondition is explicit in the operation's predicate; if not the operation is modified accordingly. For the sake of continuity of the specification in this paper we determine the precondition of each operation at the same point at which such operation is first defined.
- Draw up a table showing all the partial operations together with their inputs, outputs and preconditions for correct operation.
- Define all schemas that present error conditions.
- Use the **Z** schema calculus to make all the partial operations total.
- Provide any additional information to assist the reader of the specification, e.g. give a summary of all the robust operations at the end.

3 Background to MLM systems

Traditional multi-level marketing (MLM) businesses GNLD [7] have been around for a number of years and many conventional businesses are now starting to add some form of MLM to their existing operation. A multi-level marketing business normally markets consumable products[1] and operates as follows:

A new *distributor* (also called a member) pays a registration fee and joins the business either as a direct associate of the company, or under an existing distributor called a *sponsor.* The sponsor does not sponsor the new distributor with money, but rather with knowledge and advice about the business. Both the sponsor and the new distributor 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, the 'work' in a MLM business involves the following:

- **Step 1:** Become a distributor of the consumable products available from the company.
- **Step 2:** Sponsor others to become product distributors as well.

The sponsor is also called the *upline* of the new distributor, while the new distributor is generally known as the *downline* of the sponsor. All distributors have to renew their registration annually to remain in the business. An example of a MLM network is shown in Figure 1. The distributors $A1$, $A2$, and $A3$ in Figure 1 associated with the company directly are called the roots of the forest (or network in MLM terms).

Each product has a *point value* (pv for short) as well as a *business value*[2] (bv) associated with it. 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 (called a *turnover* in MLM terms) in the network for each distributor is calculated, and the distributor is paid (in the appropriate currency) a certain percentage of the total business value for his or her group. This is called a *bonus.*

The point value determines, on a sliding scale, the percentage to be used in the calculation of a bonus — a lower percentage for a lower turnover and a higher percentage for a higher turnover. Bonuses are the main source of income for distributors in such a network.[3] A distributor qualifies for a bonus at the end of a month *only* after having accumulated a certain number of points through personal product consumption and a (possibly different) number of points generated by downline distributors.

Distributors may also buy products from the company and sell these to customers at a profit. A customer is somebody who uses the products, but did not join the MLM business to become a distributor. A distributor may furthermore sell her or his business (or part of it) to an existing distributor, or somebody outside the MLM business who then automatically becomes a distributor. Multi-level marketing companies normally have fixed guidelines regarding such a

[1]One of the most common consumable products in such a business is soap, since it normally has a high turnover. Alternatively, one can trade with luxury items like motor cars or houses, resulting in lower turnover and often lower profits.

[2]The business value is an amount which is some indication of the price of the product.

[3]These multi-level marketing businesses are sometimes called pyramids. In a typical pyramid, the sponsor always earns more than any of his or her downline. While this might be true for some MLM businesses, it is not the case in general. Often the amount allocated to a downline is subtracted from the gross income of the sponsor (i.e. the upline). In this way it is quite possible for a downline to earn more per month than the upline. The use of a sliding scale furthermore ensures that the amount subtracted from the gross bonus of an upline is not more than the gross bonus itself.

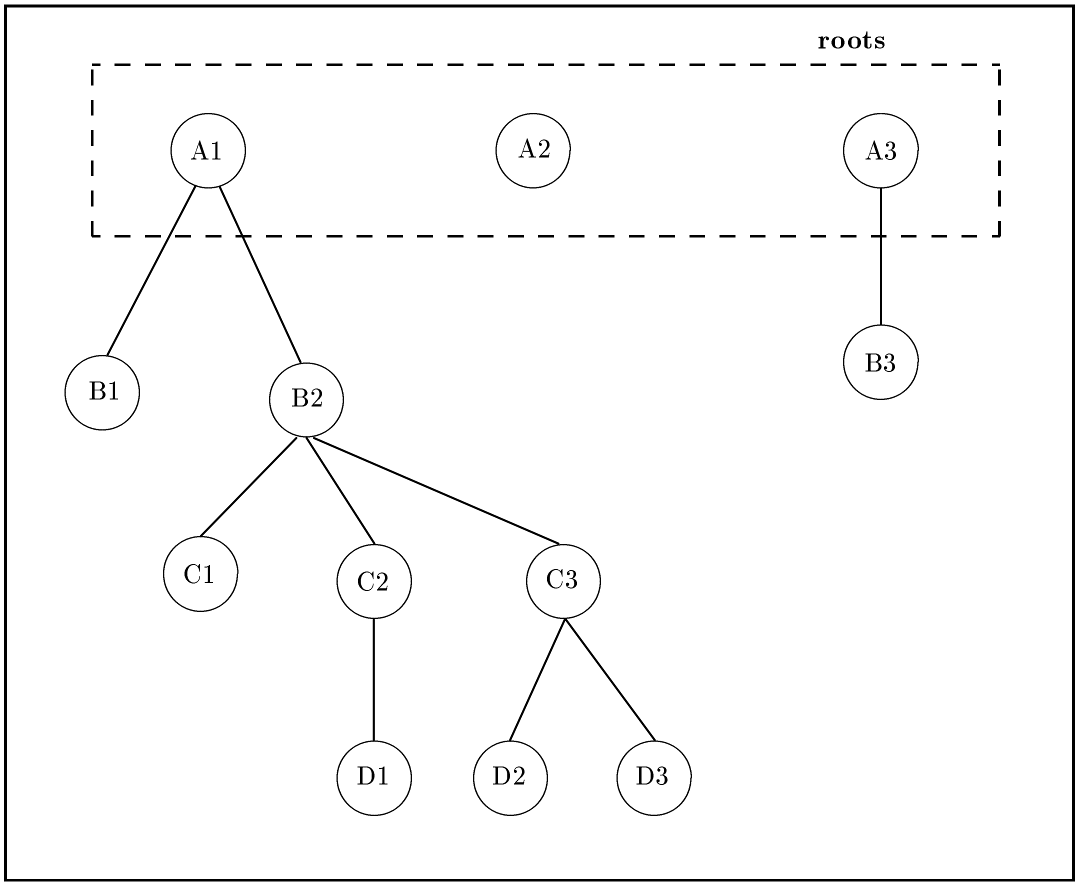

Figure 1: An example network

transaction. (See e.g. the section *The marketing plan*, pages 20 — 23 in [7].)

We are now ready to start the specification of a typical MLM business.

4 Basic definitions and state space

Following the first step of the Established Strategy we define all global constants and basic types, and explain these. A useful starting point is to draw up a natural language requirements definition of the system:

> Specify a system where new distributors may join a MLM business and become consumers of products available from the company. A distributor is allocated a unique identity code upon enrollment. Other information to be maintained for a distributor include the name, address, personal pv, and personal bv. Every product carries a point value as well as a business value. The system must allow for operations like enrolling a new distributor, ordering of products, calculation of bonuses at the end of a month and removing a distributor from the network of distributors.

From the requirements definition we identify the following basic types:

$$[\mathit{ID}, \mathit{Name}, \mathit{Address}, \mathit{PV}, \mathit{BV}, \mathit{Bonus}, \mathit{Message}]$$

ID represents the set of all possible identity codes, *Name* and *Address* the sets of all possible distributor names and addresses respectively, *PV* and *BV* the sets of point values and business values respectively, *Bonus* all bonuses to be paid out to distributors and *Message* the set of all possible messages generated by the system to provide feedback to users.

$$\begin{aligned}&\mathit{Message} ::= \\ &\quad \mathit{New\_distributor\_added} \mid \mathit{Order\_captured} \\ &\quad \mid \mathit{Bonus\_calculated} \mid \mathit{Distributor\_deleted} \\ &\quad \mid \mathit{No\_more\_identity\_codes} \mid \mathit{Unknown\_distributor}\end{aligned}$$

The specification below is developed from the viewpoint of the company which creates the environment within which distributors can build their businesses. One possibility is that the company views the businesses as a group of top-level upline distributors, each with their own network of downline distributors. This viewpoint corresponds to the idea of a *forest*, where the *roots* of the trees in the forest represent the top-level upline distributors, and the network of downline distributors for each top-level upline is represented by a corresponding *tree*. This view fits the generic model for forests and trees put forward by Scheurer [12].

The next step is to define the abstract state:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{MLM}} \\
\hline
\textit{known} : \mathbb{P}\,\textit{ID} \\
\textit{NRoots} : \mathbb{P}\,\textit{ID} \\
\textit{NUplines} : \textit{ID} \leftrightarrow \textit{ID} \\
\textit{NDist} : \textit{ID} \nrightarrow \\
\quad \textit{Name} \times \textit{Address} \times \textit{PV} \times \textit{BV} \times \textit{Bonus} \\
\hline
\textit{known} = \operatorname{dom} \textit{NDist} \\
\operatorname{dom} \textit{NUplines} \cup \operatorname{ran} \textit{NUplines} \subseteq \textit{known} \\
\textit{NRoots} = \textit{known} \setminus \operatorname{ran} \textit{NUplines} \\
\textit{Inj}(\textit{NUplines}) \\
(\forall\, \textit{Netw}) \\
\quad (\textit{Netw} \subseteq \textit{known} \wedge \textit{NRoots} \subseteq \textit{Netw} \wedge \\
\quad\quad \textit{ClosedBy}(\textit{Netw}, \textit{NUplines}) \\
\quad\quad\quad \longrightarrow \textit{Netw} = \textit{known}) \\
\hline
\end{array}
$$

Schema *MLM* introduces four components and two auxiliary predicates:

- The set *known* contains the identity codes of all distributors known to the system.
- *NRoots* represents the distributors who do not have an upline in the business but joined the company directly.
- The network of distributors is represented by the relation *NUplines*.
- The function *NDist* represents a mapping from a unique identity code to the particulars for that distributor. Note that *NDist* is partial, since for a given value of *NDist* it need not be the case that $\operatorname{dom} \textit{NDist} = \textit{ID}$; neither is it injective since two (or more) distributors may have the same particulars (i.e. name, address, etc.).
- Relation *NUplines* is injective since every distributor in the business has at most one upline. This property is captured by the predicate *Inj*(*NUplines*). *Inj* is defined by:

  $$
  \begin{array}{l}
  (\forall\, R)(\textit{Inj}(R) \leftrightarrow \\
  \quad (\forall\, i)(\forall\, j)(\forall\, k)(\,((i,k) \in R \wedge (j,k) \in R) \\
  \quad\quad \longrightarrow (i = j)))
  \end{array} \tag{1}
  $$

  Note that the first coordinate of a pair in *NUplines* represents the *upline* of a *downline* which is the second coordinate of the pair.
- The purpose of the last predicate in *MLM* is to ensure that *known*, which represents the set of all known identity codes, is the smallest inductive set of identity codes generated by relation *NUplines*, starting from the root nodes, *NRoots*. Predicate *ClosedBy* is given by:

  $$
  \begin{array}{l}
  (\forall\, \textit{Netw})(\forall\, R) \\
  \quad (\textit{ClosedBy}(\textit{Netw}, R) \leftrightarrow \\
  \quad\quad (\forall\, i)(\forall\, j)(i \in \textit{Netw} \wedge (i,j) \in R \\
  \quad\quad\quad \longrightarrow j \in \textit{Netw}))
  \end{array} \tag{2}
  $$

5 System operations

The following step is to specify the system operations and the first operation is to define an initial state. Initialisation is not available to an ordinary user of the system, nevertheless it is viewed as an operation in **Z** since it delivers an after state, $\textit{MLM}'$.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InitMLM}} \\
\hline
\textit{MLM}' \\
\hline
\textit{known}' = \varnothing \wedge \textit{NRoots}' = \varnothing \wedge \\
\textit{NUplines}' = \varnothing \wedge \textit{NDist}' = \varnothing \\
\hline
\end{array}
$$

A proof obligation is to show that $\textit{MLM}'$ above can be realised, i.e. the invariant is preserved with all state components equated to the empty set. This is called the Initialisation Theorem [9] and it takes the form:

$$\vdash \exists\, \textit{MLM}' \bullet \textit{InitMLM} \tag{3}$$

Theorem (3) indeed holds, since each of the five parts of the invariant in *MLM* above is true, given the initial values of the state components in *InitMLM*.

Our first user operation defined on the state is to register a new distributor. A new distributor (say p) may register either directly with the company (i.e. becomes a root node) or under an existing distributor (say q) in the network as follows:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Register\_with\_upline}} \\
\hline
\Delta \textit{MLM} \\
p!, q? : \textit{ID} \\
\textit{name}? : \textit{Name};\ \textit{addr}? : \textit{Address} \\
\textit{mes}! : \textit{Message} \\
\hline
p! \notin \textit{known} \wedge q? \in \textit{known} \\
\textit{known}' = \textit{known} \cup \{p!\} \\
\textit{NUplines}' = \textit{NUplines} \cup \{q? \mapsto p!\} \\
\textit{NDist}' = \textit{NDist} \cup \\
\quad \{p! \mapsto (\textit{name}?, \textit{addr}?, 0, 0.0, 0.0)\} \\
\textit{mes}! = \textit{New\_distributor\_added} \\
\hline
\end{array}
$$

As is customary in **Z**, a new identity code $p!$ is generated by the system and the new distributor is linked to an existing one, $q?$. The network, *NUplines*, is updated accordingly. Initial product information pertaining to the new distributor is reflected in specifying the personal point value to be 0 and both the business value and potential bonus equal to the real value 0.0.

The precondition of *Register_with_Upline* is given by $p! \notin known \wedge q? \in known$ and we have to check whether this precondition is sufficient for the correct operation of *Register_with_Upline*. Calculation of the precondition reveals that the part $p! \notin known$ is not sufficient since the system generates $p!$ from the set *ID* and it's possible that there are no unused identity codes left, i.e. a new $p!$ cannot be generated. It turns out that we need to change the precondition to $known \neq ID \wedge q? \in known$. The complete calculation is presented in Appendix A.

Note that the relationship between the before and after state values of the state component *NRoots* is unspecified. The reason is that *NRoots* is unaffected by the above operation and the calculation in Appendix A identifies this as a critical proof obligation. In Section 7 we discharge this proof obligation.

A new distributor may also join the company directly in which case the distributor becomes a root element and a dummy, unknown identity code is supplied for the upline:[4]

```
Register_no_upline
  $\Delta MLM$
  $p!, q? : ID$
  $name? : Name;\ addr? : Address$
  $mes! : Message$
  ----------
  $p! \notin known \wedge q? \notin known$
  $known' = known \cup \{p!\}$
  $NRoots' = NRoots \cup \{p!\}$
  $NUplines' = NUplines$
  $NDist' = NDist \cup$
    $\{p! \mapsto (name?, addr?, 0, 0.0, 0.0)\}$
  $mes! = New\_distributor\_added$
```

Calculation of the precondition reveals similarly the precondition for the correct operation of *Register_no_upline* to be $known \neq ID \wedge q? \notin known$.

The next operation captures an order.

```
Order
  $\Delta MLM$
  $id? : ID;\ pv? : PV;\ bv? : BV$
  $mes! : message$
  ----------
  $id? \in known$
  $(\exists\, pv : PV;\ bv : BV \bullet$
    $pv = third(NDist(id?)) + pv? \wedge$
    $bv = fourth(NDist(id?)) + bv? \wedge$
    $NDist' = NDist \oplus \{id? \mapsto$
        $(first(NDist(id?)), second(NDist(id?)),$
         $pv, bv, fifth(NDist(id?)))\})$
  $mes! = Order\_captured$
```

Schema *Order* introduces some functions:

- The functions *first*, *second*, and so forth project out an element at the appropriate position in the tuple.
- The symbol $\oplus$ is the relational overriding operator and its effect is to replace a tuple in a relation. Therefore, $NDist'$ is obtained from *NDist* by replacing the tuple with first coordinate $id?$ as specified above.

Calculation of the precondition reveals no additional information. The condition $id? \in known$ is sufficient for the correct operation of *Order*.

An important operation is to calculate the nett bonus for a distributor.[5] The first step is to specify an operation to calculate the gross bonus to be paid to a distributor. For any distributor, say $id?$, the gross bonus is kept as the fifth coordinate of $NDist(id?)$.

```
Gross_bonus
  $\Delta MLM$
  $id? : ID$
  ----------
  $id? \in known \wedge$
  $(\exists\, distpv, totpv : PV;\ distbv, totbv : BV \bullet$
    $distpv = third(NDist(id?)) \wedge$
    $distbv = fourth(NDist(id?)) \wedge$
    $totpv = \sum_{i \in NDescs(id?)} third(NDist(i)) \wedge$
    $totbv = \sum_{i \in NDescs(id?)} fourth(NDist(i)) \wedge$
    $fifth(NDist'(id?)) =$
      **if** $distpv \geq 100$ **then**
        **if** $(totpv < 500)$ **then** $0.0$
        **else if** $(500 \leq totpv < 1000)$ **then**
          $0.05 * totbv$
        **else if** $(1000 \leq totpv < 2000)$ **then**
          $0.1 * totbv$
        **else if** $(2000 \leq totpv < 3000)$ **then**
          $0.15 * totbv$
        **else if** $(3000 \leq totpv < 4000)$ **then**
          $0.2 * totbv$
        **else** $0.25 * totbv$
      **else** $0.0$ )
```

The function *NDescs* recursively calculates all descendents of a distributor (say p), including p:

$$NDescs(p) = \{p\} \cup \bigcup \{NDescs(j) \cup \{j\} \mid (p, j) \in NUplines\}$$

for *NUplines* a component of the *MLM* state space.

Schema *Gross_bonus* specifies a gross bonus for a distributor $id?$ along the following lines: If the personal

[4]The use of a dummy identity code is simply to make the structure of the input syntactically similar to that of operation *Register_with_upline*, a decision that facilitates the process of combining of the 2 operations into a single operation, *Register* below, using schema disjunction.

[5]The values of the bonus structures are just for illustrative purposes and would vary according to the MLM company. Normally, there would be different bonus structures for point values 'much larger' than the 4000 mark, but consideration of these is beyond the scope of this introductory specification.

pv of $id?$ is less than 100 or $id?$'s group pv is less than 500 then the bonus is 0.0. If the personal pv is greater than 99 and the group pv is 500 or more then the group pv determines the percentage to be used in the calculation of the gross bonus for $id?$.

Next we specify a nett bonus calculation:

$$\begin{array}{|l}
\textit{Nett\_bonus} \\
\hline
\Xi \mathit{MLM} \\
id? : \mathit{ID} \\
bonus! : \mathit{Bonus} \\
mes! : \mathit{Message} \\
\hline
id? \in \mathit{known} \\
bonus! = \\
\quad \mathit{fifth}(\mathit{NDist}(id?)) - \\
\qquad \sum_{(id?,i) \in \mathit{NUplines}} \mathit{fifth}(\mathit{NDist}(i)) \\
mes! = \mathit{Bonus\_calculated} \\
\hline
\end{array}$$

A nett bonus for distributor $id?$ is specified as the difference between the gross bonus for $id?$ and the sum of the gross bonuses of all the *immediate* downlines of $id?$. Precondition calculation reveals the condition $id? \in \mathit{known}$ to be sufficient for both $\mathit{Gross\_bonus}$ and $\mathit{Nett\_bonus}$.

Our last user operation deals with the scenario where a distributor decides to quit the business. Suppose a *root* distributor, say $A1$ in Figure 1, decides to drop out of the network. This has the following effect:

(1) Remove $A1$ as a root element, i.e.

$$\mathit{NRoots}' = \mathit{NRoots} \setminus \{A1\} \qquad (4)$$

(2) Remove the pairs $A1 \mapsto B1$ and $A1 \mapsto B2$ from the network, i.e.

$$\mathit{NUplines}' = \{A1\} \lhd \mathit{NUplines} \qquad (5)$$

The symbol $\lhd$ is known as the domain anti-restriction operator. Its effect is to remove all tuples with the first coordinate being an element of the set on the left of the operator from a relation.

(3) Make $B1$ and $B2$ root elements, i.e.

$$\mathit{NRoots}'' = \mathit{NRoots}' \cup \{B1, B2\} \qquad (6)$$

As a schema operation we have:

$$\begin{array}{|l}
\textit{DelRoot} \\
\hline
\Delta \mathit{MLM} \\
id? : \mathit{ID} \\
mes! : \mathit{Message} \\
\hline
id? \in \mathit{NRoots} \land \\
\mathit{known}' = \mathit{known} \setminus \{id?\} \land \\
\mathit{NDist}' = \{id?\} \lhd \mathit{NDist} \land \\
\mathit{NUplines}' = \{id?\} \lhd \mathit{NUplines} \land \\
(\exists\, \mathit{FRoots} : \mathbb{P}\, \mathit{ID} \bullet \\
\quad \mathit{FRoots} = \{r \mid (id?, r) \in \mathit{NUplines}\} \land \\
\quad \mathit{NRoots}' = (\mathit{NRoots} \setminus \{id?\}) \cup \mathit{FRoots}) \land \\
mes! = \mathit{Distributor\_deleted} \\
\hline
\end{array}$$

Note that appropriate after state values are also specified for known and NDist.

Alternatively, if a *non-root* distributor, say $B2$, decides to quit the business then the following actions are applicable:

(1) Node $B2$ is deleted from the forest. Deleting $B2$ from the forest amounts to removing the pairs $B2 \mapsto C1$, $B2 \mapsto C2$ and $B2 \mapsto C3$ from the network, i.e.

$$\mathit{NUplines}' = \mathit{NUplines} \rhd \{C1, C2, C3\} \qquad (7)$$

as well as removing the pair $A1 \mapsto B2$ from the network, i.e.

$$\mathit{NUplines}'' = \mathit{NUplines}' \rhd \{B2\} \qquad (8)$$

The symbol $\rhd$ is known as the range anti-restriction operator. Its effect is to remove all tuples with the second coordinate being an element of the set on the right of the operator from a relation. Note that equation (7) can also be written as $\{B2\} \lhd \mathit{NUplines}$. The reason for using the format in (7) is to simplify the definition of the after state value of $\mathit{NUplines}$ in schema $\mathit{Del\_Not\_Root}$ below.

(2) Nodes $C1$, $C2$ and $C3$ become immediate downlines of $A1$:

$$\begin{array}{l}
\mathit{NUplines}''' = \\
\quad \mathit{NUplines}'' \cup \{A1 \mapsto C1, A1 \mapsto C2, A1 \mapsto C3\}
\end{array}$$

As a schema we have ($^\sim$ denotes an inverse):

$$\begin{array}{|l}
\textit{Del\_Not\_Root} \\
\hline
\Delta \mathit{MLM} \\
id? : \mathit{ID} \\
mes! : \mathit{Message} \\
\hline
id? \in \operatorname{ran} \mathit{NUplines} \land \\
\mathit{known}' = \mathit{known} \setminus \{id?\} \land \\
\mathit{NDist}' = \{id?\} \lhd \mathit{NDist} \land \\
(\exists\, q : \mathit{ID};\ \mathit{FRoots} : \mathbb{P}\, \mathit{ID} \bullet \\
\quad q = \mathit{NUplines}^\sim(id?) \land \\
\quad \mathit{FRoots} = \{r \mid (id?, r) \in \mathit{NUplines}\} \land \\
\quad \mathit{NUplines}' = \\
\qquad (\mathit{NUplines} \rhd (\mathit{FRoots} \cup \{id?\})) \cup \\
\qquad\quad \{(q, r) \mid r \in \mathit{FRoots}\}) \land \\
mes! = \mathit{Distributor\_deleted} \\
\hline
\end{array}$$

The precondition for a correct execution of $\mathit{DelRoot}$ is $id? \in \mathit{NRoots}$, while the correct precondition for $\mathit{Del\_Not\_Root}$ is given by $id? \in \operatorname{ran} \mathit{NUplines}$.

The next step prescribed by the Established Strategy is to summarise the partial operations with their respective inputs, outputs and preconditions. This summary appears in Table 1.

Operation	Inputs/Outputs	Precondition
InitMLM	—	—
Register_with_upline	$q? : ID;\ name? : Name;\ addr? : Address$ $p! : ID;\ mes! : Message$	$known \neq ID \land q? \in known$
Register_no_upline	$q? : ID;\ name? : Name;\ addr? : Address$ $p! : ID;\ mes! : Message$	$known \neq ID \land q? \notin known$
Order	$id? : ID;\ pv? : PV;\ bv? : BV$ $mes! : Message$	$id? \in known$
Gross_bonus	$id? : ID$	$id? \in known$
Nett_bonus	$id? : ID$ $bonus! : Bonus;\ mes! : Message$	$id? \in known$
DelRoot	$id? : ID$ $mes! : Message$	$id? \in NRoots$
Del_Not_Root	$id? : ID$ $mes! : Message$	$id? \in \operatorname{ran} NUplines$

Table 1: Summary of partial operations

6 Strengthening the specification

Following the Established Strategy we now provide robust operations for our four system operations above. A correct implementation of our specification will faithfully perform the operations defined above, so long as there are no mistakes in the input. To cater for exceptions we define a number of error schemas and then combine these with the partial operations above to produce robust operations.

An error condition is specified by *No_more_codes* if there are no free identity codes when attempting to add a new distributor either as a root element or below an existing distributor.

No_more_codes

ΞMLM

$mes! : Message$

$known = ID$

$mes! = No\_more\_identity\_codes$

Hence, a robust operation to register a distributor is given by:

$$Register \mathrel{\widehat=} Register\_with\_upline \lor Register\_no\_upline \lor No\_more\_codes$$

Supplying an unknown identity code results in:

Unknown_distributor

ΞMLM

$id? : ID$

$mes! : Message$

$id? \notin known$

$mes! = Unknown\_distributor$

A robust operation for placing an order is:

$$Order\_product \mathrel{\widehat=} Order \lor Unknown\_distributor$$

A robust bonus calculation is specified as either a gross bonus calculation followed by the calculation of a nett bonus, or an appropriate error condition:

$$Bonus \mathrel{\widehat=} (Gross\_Bonus \fatsemi Nett\_Bonus) \lor Unknown\_distributor$$

Finally, a robust operation for deleting a distributor is given by:

$$DelDist \mathrel{\widehat=} Del\_Root \lor Del\_Not\_Root \lor Unknown\_distributor$$

7 Reasoning about the specification

In line with our proposal that critical proof obligations arising from a specification be stated and discharged, we now prove two properties of our specification. Normally a specifier has a wide choice in which reasoning mechanism to use for proving properties of a specification. Well-known systems are the *interactive*, term-rewriting theorem provers like CaDiZ [15] and Z-Eves [11]. Alternatively, one can use a fully *automatic* reasoner like OTTER [8] or Gandalf [14]. In this paper we use OTTER as our reasoning assistant.

OTTER is a first-order, resolution-based theorem prover developed by William C. McCune at the Argonne National Laboratory in Illinois, USA. Two important sections often present in the input to OTTER are the *usable list* and the *set-of-support (sos) list.* One way in which to use OTTER is to place the relevant facts known to be true in the proof attempt in the usable list and the negation of what we want to prove in the sos.

The precondition calculation in Appendix A for operation *Register_with_upline* reveals a critical proof obligation:

$$NRoots' = known' \setminus \operatorname{ran} NUplines' \qquad (9)$$

Note that the above predicate is part of the invariant defined on the *MLM* state and in **Z** system operations are implicitly assumed to preserve the state invariant [5], hence proving that the state invariant is preserved by an operation is not considered a necessary proof obligation in **Z**. Nevertheless, it is a good idea to check that an operation does not incur unpredictable results as far as the invariant is concerned. In fact, in systems like the **B** method [1, 18] ensuring that an operation preserves the invariant is considered a critical proof obligation.

Suppose a new distributor p is registered below an existing distributor, q. Consider two restrictions specified in *Register_with_upline*:

$$known' = known \cup \{p\} \qquad (10)$$

$$NUplines' = NUplines \cup \{q \mapsto p\} \qquad (11)$$

Suppose we define:

$$NewRoots = known' \setminus \operatorname{ran} NUplines' \qquad (12)$$

and pose the negation of the following equality in the OTTER set-of-support list

$$(NRoots = NewRoots) \qquad (13)$$

then the theorem prover fails to find any proof for (13) in 20 minutes. It is well known that equality reasoning poses demanding challenges to automated reasoning assistants [4, 10]. Therefore, if we apply the first axiom of Zermelo-Fraenkel set theory [6], namely Extensionality, and simply replace (13) with (14) below

$$(\forall\, x)(x \in NRoots \leftrightarrow x \in NewRoots) \qquad (14)$$

then OTTER finds a short proof in *0.49 seconds.*[6] Note that this result also implies that the set of root nodes (i.e. *NRoots*) is not affected by operation *Register_with_upline*.

Another example of a proof is to show that the cardinality of a state component is preserved, e.g.

$$\#NDist' = \#NDist \qquad (15)$$

after operation *Order_product*. Suppose the precondition $id? \in known$ is satisfied and the after state value of *NDist* is as reflected in operation *Order*:

$$\begin{aligned}
&NDist' = \\
&\quad NDist \oplus \\
&\qquad \{id? \mapsto \\
&\qquad\quad (first(NDist(id?)), \\
&\qquad\qquad second(NDist(id?)), pv, bv, bonus)\} \qquad (16)
\end{aligned}$$

[6]Proof performed on Linux Red Hat 6.2 on a Pentium III running at 600 MHz.

Proof obligation (15) has a recursive nature as is evident from the following definitions of cardinality [16]:

$$(\forall\, A)(\#A = 0 \leftrightarrow A = \varnothing) \qquad (17)$$

$$\begin{aligned}
&(\forall\, A)(\forall\, n)(\#A = n + 1 \leftrightarrow \\
&\quad (\exists\, x)(x \in A \land \#(A \setminus \{x\}) = n)) \qquad (18)
\end{aligned}$$

In a more procedural fashion relational overriding, $\oplus$, for any two relations R and S is defined as:

$$R \oplus S = (\operatorname{dom} S \lessdot R) \cup S \qquad (19)$$

Hence, definition (16) is unfolded into the two formulae (20) and (21):

$$\begin{aligned}
&(\forall\, u)(\forall\, v)(\forall\, w)(\forall\, x)(\forall\, y)(\forall\, z) \\
&\quad (6\,TUP(u, v, w, x, y, z) \in NDist'' \leftrightarrow \\
&\qquad 6\,TUP(u, v, w, x, y, z) \in NDist \land u \neq id?) \qquad (20)
\end{aligned}$$

The set $NDist''$ represents the entity $(\operatorname{dom} S \lessdot R)$ in definition (19) above.

$$\begin{aligned}
&(\forall\, u)(\forall\, v)(\forall\, w)(\forall\, x)(\forall\, y)(\forall\, z) \\
&\quad (6\,TUP(u, v, w, x, y, z) \in NDist' \leftrightarrow \\
&\qquad (6\,TUP(u, v, w, x, y, z) \in NDist'' \lor \\
&\qquad\quad 6\,TUP(u, v, w, x, y, z) = \\
&\qquad\qquad 6\,TUP(id?, name, addr, pv, bv, bonus))) \qquad (21)
\end{aligned}$$

where *name*, *addr*, *pv*, *bv* and *bonus* represent the last 5 coordinates of a tuple from *NDist*.

Proof obligation (15) above boils down to showing that if we start by claiming that $\#NDist = k + 1$ (say), then $\#NDist' = k + 1$. OTTER fails to find any proof of this property using the standard definitions (17) and (18). One of the reasons for failure is that these definitions are not suitable for a proof of (15), especially not when we are dealing with sets of tuples instead of simple sets.

We start our attack on the problem by employing a reasoning strategy called *resonance* described in [22]. The aim of the resonance strategy is to give preference (for directing a program's reasoning) to formulae that have the same syntactic shape (ignoring variables) as one or more of the patterns supplied by the specifier. Such formulae are called *resonators*.

We, therefore, rewrite the cardinality definitions in terms of a tuple being removed from or added to a set.

First we give definition for removing a tuple from a set:

$$\begin{aligned}
&(\forall\, A)(\forall\, n)(\forall\, u)(\forall\, v)(\forall\, w)(\forall\, x)(\forall\, y)(\forall\, z)(\forall\, B) \\
&\quad ((Fun(A) \land \#A = n + 1 \land \\
&\qquad 6\,TUP(u, v, w, x, y, z) \in A \land \\
&\qquad\quad (\forall\, u1)(\forall\, v1)(\forall\, w1)(\forall\, x1)(\forall\, y1)(\forall\, z1) \\
&\qquad\qquad (6\,TUP(u1, v1, w1, x1, y1, z1) \in B \leftrightarrow \\
&\qquad\qquad\quad 6\,TUP(u1, v1, w1, x1, y1, z1) \in A \land u1 \neq u)) \\
&\qquad\qquad\qquad \longrightarrow \#B = n) \qquad (22)
\end{aligned}$$

Definition (22) could be explained as follows. For any given 6-tuple relations A and B, if A is a function and $\#A = n + 1$ then $\#B = n$, provided that B is obtained from A by performing a domain subtraction of a single tuple from A.

A resonance definition for adding a new tuple to a set is:

$$
\begin{aligned}
&(\forall A)(\forall n)(\forall x)(\forall B)\\
&\quad((\#A = n \wedge x \notin \operatorname{dom} A \wedge\\
&\quad\quad((\exists\, name)(\exists\, addr)(\exists\, pv)(\exists\, bv)(\exists\, bonus)\\
&\quad\quad\quad(\forall u1)(\forall v1)(\forall w1)(\forall x1)(\forall y1)(\forall z1)\\
&\quad\quad\quad\quad(6\,TUP(u1, v1, w1, x1, y1, z1) \in B \leftrightarrow\\
&\quad\quad\quad\quad\quad 6\,TUP(u1, v1, w1, x1, y1, z1) \in A \vee\\
&\quad\quad\quad\quad\quad\quad 6\,TUP(u1, v1, w1, x1, y1, z1) =\\
&\quad\quad\quad\quad\quad\quad\quad 6\,TUP(x, name, addr, pv, bv, bonus))))\\
&\quad\quad\longrightarrow \#B = n + 1)
\end{aligned}
\tag{23}
$$

Definition (23) states that for any sets A and B and any x where $x \notin \operatorname{dom} A$, if $\#A = n$ and B is obtained from A by adding a tuple with x as the first coordinate, then $\#B = n + 1$.

We also need a resonance definition of a domain:

$$
\begin{aligned}
&(\forall R)(\forall u)\\
&\quad(u \in \operatorname{dom} R \leftrightarrow\\
&\quad\quad(\exists v)(\exists w)(\exists x)(\exists y)(\exists z)\\
&\quad\quad\quad(6\,TUP(u, v, w, x, y, z) \in R))
\end{aligned}
\tag{24}
$$

These changes allow OTTER to find a short proof for (15) in just *1.78 seconds*. The input to the theorem prover appears in Appendix B.

8 Summary of operations

In line with the Established Strategy we summarise all robust MLM operations.

Register $\widehat{=}$

Register_with_upline $\vee$ *Register_no_upline* $\vee$ *No_more_codes*

Order_Product $\widehat{=}$ *Order* $\vee$ *Unknown_distributor*

Bonus $\widehat{=}$ (*Gross_Bonus* $\fatsemi$ *Nett_Bonus*) $\vee$ *Unknown_distributor*

Del_Dist $\widehat{=}$ *Del_Root* $\vee$ *Del_Not_Root* $\vee$ *Unknown_distributor*

9 Enhancing the specification

Some enhancements can be made to our specification: Distributors sometimes pose queries which need not go through their uplines, but which may be addressed to the company directly. The company appoints some 'distributor relations coordinators' for this purpose and these coordinators are assigned to the top-level uplines. If the group of distributors belonging to a top-level upline is not too large (according to the judgement of the company), then only one coordinator is assigned to such a group of distributors, but larger groups normally have more than one coordinator assigned to them.

One way of incorporating coordinators in our specification is to define a new function

$$NCoord : ID \rightarrowtail \mathbb{P}\, Coord$$

where *Coord* represents the set of coordinators employed by the company. For any distributor identity code d, if $d \notin \operatorname{dom} NCoord$ then d is not a top-level upline and therefore d's top-level upline has to be determined first, whereafter the coordinator set for d is obtained. The set could be a singleton for a small network as mentioned above. Note that the variable *NCoord* would become a fifth component of the state, possibly further complicating proof attempts when reasoning about properties of the state.

An enhancement can also be made to the bonus calculation defined in schema *Gross_bonus*. Inevitably in a MLM business there are additional, more advanced bonus structures. For example, one such structure involves the idea of a '4000 point unit': A distributor is paid a bonus of 5% of the business volume[7] of a 4000 point unit on his or her first level, 2% on the business volume of such a unit on the second level, and 1% on the business volume of a similar unit on the third level. These percentages advance on a sliding scale with the top values fixed at 7%, 5%, and 2% respectively. (See e.g. the section *The Marketing Plan*, pages 5 — 6 in [7].)

As far as our proof structure is concerned, the following critical proof obligation arises from operation *Nett_bonus* above:

> The sum of the gross bonuses allocated to *first-level* downline distributors is subtracted from the gross bonus allocated to their parent distributor in the network. Show, therefore, that the sum of the gross bonuses allocated to first-level downline distributors is *less than or equal to* the gross bonus allocated to their immediate upline.

Formally, consider a distributor, say q, with a number of first-level downlines, represented by a set, say D, such that

$$(\forall d)(d \in D \leftrightarrow (q, d) \in NUplines)$$

[7]In the MLM world a business value is often called a *business volume*.

for *NUplines* a component of the MLM state space. Suppose the gross bonus allocated to a distributor, say i, is indicated by $Bonus_i$. The proof obligation is to show that:

$$Bonus_q \; - \sum_{(q,d) \in NUplines} Bonus_d \; \geq \; 0 \qquad (25)$$

OTTER fails to find any proof of the above property. Hence more work is needed to develop a fresh set of heuristics for reasoning about arithmetic expressions, reminiscent of the heuristics developed in [17] for reasoning about expressions in set theory.

Of course a rather simple way to solve this problem in one fell swoop in **Z** is to include additional conditions in the state invariant in schema *MLM* requiring that the value of every distributor's bonus be non-negative and greater than or equal to the sum of the bonuses of that distributor's immediate downlines. Nevertheless, the sensible specifier will always check (using some proof mechanism) that all relevant operations respect this property.

10 Summary and future work

This paper presented for the first time a formal specification of part of a realistic multi-level marketing business in **Z**. The specification was constructed using the Established Strategy for presenting a **Z** specification. In essence, specifying such a business boils down to specifying properties of trees and forests and operations on these. **Z** appears to be a suitable vehicle for this task. We illustrated how some proof obligations arising from the specification may be stated and discharged using an appropriate reasoning assistant. In the process we demonstrated the utility of two reasoning strategies, namely avoiding equality and using resonance.

However, the specification of operations on recursive structures like forests and trees are non-trivial, as is evident from the predicates in *Gross_bonus* and *Nett_bonus*. We saw how this complexity is underlined by the difficulty encountered in finding a proof of (25) and that more work is needed before we can attempt a proof of an arithmetic expression with embedded recursion such as (25), using a resolution-based reasoner like OTTER. Alternatively, one can investigate the feasibility of using a well-known interactive term-rewriting reasoner, e.g. CaDiZ [15] or Z-Eves [11].

Appendix A

The precondition of operation *Register_with_upline* is denoted by pre *Register_with_upline*. Define *preRegister_with_upline* = pre *Register_with_upline*.

Step 1: Calculate *preRegister_with_upline* by restating *Register_with_upline*, existentially quantifying all output and after state variables:

$$\begin{array}{|l}
\textit{preRegister\_with\_upline} \\
\hline
MLM \\
q? : ID;\ name? : Name;\ addr? : Address \\
\hline
\exists\, MLM';\ p! : ID;\ mes! : Message \bullet \\
\quad p! \notin known \land q? \in known \land \\
\quad known' = known \cup \{p!\} \land \\
\quad NUplines' = NUplines \cup \{q? \mapsto p!\} \land \\
\quad NDist' = NDist \cup \\
\qquad \{p! \mapsto (name?, addr?, 0, 0.0, 0.0)\} \land \\
\quad mes! = New\_distributor\_added \\
\hline
\end{array}$$

Step 2: Expand the reference to MLM' above:

$$\begin{array}{l}
\exists\, known', NRoots' : \mathbb{P}\, ID;\ NUplines' : ID \leftrightarrow ID; \\
\quad NDist' : ID \rightarrowtail \\
\qquad Name \times Address \times PV \times BV \times Bonus; \\
\quad p! : ID;\ mes! : Message \bullet \\
\quad known' = \mathrm{dom}\, NDist' \land \\
\quad \mathrm{dom}\, NUplines' \cup \mathrm{ran}\, NUplines' \subseteq known' \land \\
\quad NRoots' = known' \setminus \mathrm{ran}\, NUplines' \land \\
\quad Inj(NUplines') \land \\
\quad (\forall\, Netw) \\
\qquad (Netw \subseteq known' \land NRoots' \subseteq Netw \land \\
\qquad\quad ClosedBy(Netw, NUplines') \\
\qquad\qquad \longrightarrow Netw = known') \land \\
\quad p! \notin known \land q? \in known \land \\
\quad known' = known \cup \{p!\} \land \\
\quad NUplines' = NUplines \cup \{q? \mapsto p!\} \land \\
\quad NDist' = NDist \cup \\
\qquad \{p! \mapsto (name?, addr?, 0, 0.0, 0.0)\} \land \\
\quad mes! = New\_distributor\_added
\end{array}$$

Step 3: Eliminate constants and redundant invariant parts, and instantiate dashed variables where possible:

$$\begin{array}{l}
\exists\, NRoots' : \mathbb{P}\, ID;\ p! : ID \bullet \\
\quad known \cup \{p!\} \in \mathbb{P}\, ID \land \\
\quad NUplines \cup \{q? \mapsto p!\} \in ID \leftrightarrow ID \land \\
\quad NDist \cup \{p! \mapsto (name?, addr?, 0, 0.0, 0.0)\} \in \\
\qquad ID \rightarrowtail Name \times Address \times PV \times BV \times Bonus \land \\
\quad known \cup \{p!\} = \mathrm{dom}\,(NDist \cup \\
\qquad \{p! \mapsto (name?, addr?, 0, 0.0, 0.0)\}) \land \\
\quad \mathrm{dom}\,(NUplines \cup \{q? \mapsto p!\}) \cup \\
\qquad \mathrm{ran}\,(NUplines \cup \{q? \mapsto p!\}) \subseteq known \cup \{p!\} \land \\
\quad NRoots' = \\
\qquad (known \cup \{p!\}) \setminus \mathrm{ran}(NUplines \cup \{q? \mapsto p!\}) \land \\
\quad p! \notin known \land q? \in known
\end{array}$$

Step 4: Verify the correctness of and thereafter remove the first 3 type definitions above.

$$\begin{array}{l}
\exists\, NRoots' : \mathbb{P}\, ID;\ p! : ID \bullet \\
\quad known \cup \{p!\} = \mathrm{dom}\,(NDist \cup \\
\qquad \{p! \mapsto (name?, addr?, 0, 0.0, 0.0)\}) \land \\
\quad \mathrm{dom}\,(NUplines \cup \{q? \mapsto p!\}) \cup \\
\qquad \mathrm{ran}\,(NUplines \cup \{q? \mapsto p!\}) \subseteq known \cup \{p!\} \land \\
\quad NRoots' = \\
\qquad (known \cup \{p!\}) \setminus \mathrm{ran}(NUplines \cup \{q? \mapsto p!\}) \land \\
\quad p! \notin known \land q? \in known
\end{array}$$

Step 5: Simplify using properties of dom and ran:

$$\begin{array}{l} \exists\, NRoots' : \mathbb{P}\, ID;\ p! : ID \bullet \\ \quad known \cup \{p!\} = \mathrm{dom}\ NDist \cup \{p!\} \land \\ \quad \mathrm{dom}\ NUplines \cup \{q?\} \cup \\ \qquad \mathrm{ran}\ NUplines \cup \{p!\} \subseteq known \cup \{p!\} \land \\ \quad NRoots' = \\ \qquad (known \cup \{p!\}) \setminus \mathrm{ran}(NUplines \cup \{q? \mapsto p!\}) \land \\ \quad p! \notin known \land q? \in known \end{array}$$

Step 6: Simplify using invariant on before state and the restriction $q? \in known$:

$$\begin{array}{l} \exists\, NRoots' : \mathbb{P}\, ID;\ p! : ID \bullet \\ \quad NRoots' = \\ \qquad (known \cup \{p!\}) \setminus \mathrm{ran}(NUplines \cup \{q? \mapsto p!\}) \land \\ \quad p! \notin known \land q? \in known \end{array}$$

Hence the precondition is $p! \notin known \land q? \in known$. The 1st conjunct is true exactly when *known* is not the whole of *ID*; then and only then will there exist a suitable $p!$ not in *known*. Therefore the precondition changes to:

$$known \neq ID \land q? \in known$$

Note that the after state condition $NRoots' = (known \cup \{p!\}) \setminus \mathrm{ran}(NUplines \cup \{q? \mapsto p!\})$ becomes a proof obligation.

Appendix B

Below is the input to OTTER for a proof of:

$$\#NDist' = \#NDist$$

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Prove #NDist' = #NDist after %%
%% overriding an element.       %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
set(neg_hyper_res).
set(factor).
set(ur_res).
%%
%% Paramodulation.
%% ---------------
set(para_from).
set(para_into).
%%
set(order_eq).
%%
%% Restriction strategies:
%% -----------------------
set(process_input).
clear(para_from_right).
clear(para_into_right).
set(dynamic_demod_all).
set(back_demod).
%%
weight_list(pick_and_purge).

weight(x,5).

end_of_list.
%%
clear(print_given).
clear(print_kept).
clear(print_back_sub).

formula_list(usable).

%% Reflexivity.
%% ------------
(all x (x = x)).

%% Resonance domain definition.
%% ----------------------------
(all R u
 ( El(u,dom(R)) <->
  (exists v w x y z
    El(6TUP(u,v,w,x,y,z),R)) )).

%% ##############################
%% Resonance cardinality follows.
%% ##############################

%% Removing element - Domain subtraction:
%% -------------------------------------
(all A n u v w x y z B
 ( ( Card(A,$SUM(n,1)) &
     El(6TUP(u,v,w,x,y,z),A) &
     Fun(A) &
     (all u1 v1 w1 x1 y1 z1
      ( El(6TUP(u1,v1,w1,x1,y1,z1),B) <->
         (El(6TUP(u1,v1,w1,x1,y1,z1),A) &
          -(u1 = u)) )) )
   -> Card(B,n) )).

%% Adding a new element:
%% ---------------------
(all A n x B
 ( ( Card(A,n) &
     -El(x,dom(A)) &
     (exists name addr pv bv bonus
      (all u1 v1 w1 x1 y1 z1
       ( El(6TUP(u1,v1,w1,x1,y1,z1),B) <->
          (El(6TUP(u1,v1,w1,x1,y1,z1),A) |
          (6TUP(u1,v1,w1,x1,y1,z1) =
           6TUP(x,name,addr,pv,bv,bonus))))
       )) )
    -> Card(B,$SUM(n,1)) )).

%% NDist1 = DOMDIFF(NDist,{id}).
%% -----------------------------
(all u v w x y z
 ( El(6TUP(u,v,w,x,y,z),NDist1) <->
    (El(6TUP(u,v,w,x,y,z),NDist) &
     -(u = id)) )).

%% NDist2 =
%%  NDist1 u {(id,name,addr,pv,bv,bonus)}.
%% --------------------------------------
(all u v w x y z
 ( El(6TUP(u,v,w,x,y,z),NDist2) <->
    (El(6TUP(u,v,w,x,y,z),NDist1) |
     (6TUP(u,v,w,x,y,z) =
      6TUP(id,name,addr,pv,bv,bonus))) )).

%% NDist is functional.
%% --------------------
Fun(NDist).

%% El(id,NDist)).
%% --------------
El(6TUP(id,name,addr,pv,bv,bonus),NDist).

%% #NDist = k + 1.
%% ---------------
Card(NDist,$SUM(k,1)).

%% Fact about NDist1.
%% ------------------
-El(id,dom(NDist1)).

end_of_list.

formula_list(sos).

%% #NDist2 = k + 1.
%% ----------------
-Card(NDist2,$SUM(k,1)).

end_of_list.
```

References

[1] J.-R. Abrial. *The **B** Book: Assigning Programs to Meanings.* Cambridge University Press, August 1996.

[2] S. Baase and A. Van Gelder. *Computer Algorithms: Introduction to Design & Analysis.* Addison-Wesley, 2000.

[3] R. Barden, S. Stepney, and D. Cooper. ***Z** in Practice.* Prentice-Hall, 1994.

[4] R. Boyer, E. Lusk, W. McCune, R. Overbeek, M. Stickel, and L. Wos. Set Theory in First-Order Logic: Clauses for Gödel's Axioms. *Journal of Automated Reasoning*, 2(3):287 – 327, September 1986.

[5] A. Diller and R. Docherty. **Z** and Abstract Machine Notation: A Comparison. In J. Bowen and J. Hall, editors, ***Z** User Workshop: Proceedings of the Eighth **Z** User Meeting, Cambridge, U.K., 29 - 30 June 1994*, Workshops in Computing, pages 250 – 263. Springer-Verlag, 1994.

[6] H. Enderton. *Elements of Set Theory.* Academic Press, Inc., 1977.

[7] Golden Neo-Life Diamite International. *Distributor Business Guide: The Business Plan*, July 1997.

[8] W. W. McCune. *OTTER 3.0 Reference Manual and Guide.* Argonne National Laboratory, Argonne, Illinois, August 1995. ANL-94/6.

[9] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and **Z**.* Prentice-Hall, 2nd edition, 1996.

[10] A. Quaife. *Automated Development of Fundamental Mathematical Theories.* Automated Reasoning Series. Kluwer Academic Publishers, 1992.

[11] M. Saaltink. **Z** and Eves. In J. Nicholls, editor, ***Z** User Workshop: Proceedings of the Sixth Annual **Z** User Meeting, York, U.K., 16 - 17 December 1991*, pages 223 – 242. British Computer Society, Springer-Verlag, 1992.

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

[13] J. M. Spivey. *The **Z** Notation: A Reference Manual.* Prentice-Hall, London, 2nd edition, 1992.

[14] T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199 – 204, 1997.

[15] I. Toyn. CaDiZ Type Checker and Theorem Prover Home Page. http://www.cs.york.ac.uk/~ian-/cadiz/home.html.

[16] J. van der Poll. *Automated Support for Set-Theoretic Specifications.* PhD thesis, Department of Computer Science & IS, University of South Africa, June 2000.

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

[18] H. Waeselynck. Developing Safe Software: Software development using the **B**-method. LAAS-CNRS, France. Short course: University of the Witwatersrand, February 1999.

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

[20] J. Wordsworth. Practical experience of formal specification: a programming interface for communications. In J. McDermid and C. Ghezzi, editors, *2nd European Software Engineering Conference (ESEC)*, pages 140 — 158. University of Warwick, Coventry, UK, Springer-Verlag, 1989.

[21] J. Wordsworth. A **Z** Development Method. Draft version 0.11. Technical report, IBM, UK, Hursley Park, January 1989.

[22] L. Wos. The Resonance Strategy. *Computers and Mathematics with Applications*, 29(2):133 – 178, February 1995. (Special issue on Automated Reasoning).