Translation of TTF Test Specifications into $\{log\}$

Maximiliano Cristiá
CIFASIS and UNR
Rosario, Argentina
cristia@cifasis-conicet.gov.ar

Gianfranco Rossi
Università degli studi di Parma
Parma, Italy
gianfranco.rossi@unipr.it

This document describes the translation rules to be applied when a test specification generated by Fastest must be translated into $\{log\}$.

The translation rules are given as follows:

$$\text{rule name}\,\frac{\textit{Z notation}}{\{log\}\ \mathsf{language}}$$

where the text above the line is some Z term and the text below the line is one or more $\{log\}$ formulas.

This document assumes the reader is familiar with the mathematics underlying either Z or B and with general notions and notations of Prolog and $\{log\}$.

1. Translation Strategy

1. The translation is applied to each test specification in isolation.

   For this reason, translation rules for the schema calculus are unnecessary. Recall that test specifications do not use schema calculus.

2. Each test specification is translated as a $\{log\}$ goal (i.e., a conjunction of either positive or negative atomic predicates).

3. Variable names are translated into the same name but the first letter is written in uppercase.

   If the translated name conflicts with another identifier then some algorithm to avoid name clashes must be applied.

4. Information about the type of a particular variable is seldom given through a type declaration but mainly by means of the expressions in which that variable participates in the test specification. For example, if a variable in a test specification is $x : T \times U$ for some types T and U, no type information will be given for it in $\{log\}$. However, if the predicate of that test specification says, for instance, $x.1 > 12$ then it will be translated as $fst(x) > 12$ which will guide $\{log\}$ in giving an ordered pair as the value for x.

5. Explicit type information of a variable is given at the $\{log\}$ level only if it helps or speedups the tool in finding a solution. A typical case are integer or natural variables.

6. As part of the preprocessing, all subexpressions are replaced for fresh variables and new equalities binding these fresh variables with the corresponding subexpressions, are added to the test specification. For example, the following predicate:

   $A \cup (B \cap C) \subseteq D \oplus F$

is replaced as follows:

$$\begin{aligned}&G_1 = D \oplus F\\&G_2 = A \cup G_3\\&G_3 = B \cap D\\&G_2 \subseteq G_1\end{aligned}$$

where G_1, G_2 and G_3 are fresh variables within the test specification. Another example is:

$$x * (y+z) \leq h \operatorname{\mathsf{div}} (j+z)$$

which is rewritten as:

$$\begin{aligned}&a_1 = y+z\\&a_2 = j+z\\&a_3 = x * a_1\\&a_4 = h \operatorname{\mathsf{div}} a_2\\&a_3 \leq a_4\end{aligned}$$

where a_1, a_2, a_3 and a_4 are all fresh variables.

This preprocessing must be done inside quantified predicates, lambda expressions and set comprehensions, taking care of the scope of each subexpression.

Note that the referential transparency property of Z specifications ensures that the modification shown above preserves the meaning of the test specification.

The equalities introduced by this process, conceptually, will be added to one of two lists. One list, called *set equalities*, will contain all these equalities whose left hand side is a set (note that in this category fall sequences and binary relations); the other one, called *arithmetic equalities*, will contain all these equalities whose left hand side is of type $\mathbb{Z}$.

Then, during the translation (i.e. after preprocessing) each equality will be translated as follows:

a) Set equalities. An equality of the form:

$$A = B \boxplus C$$

where $\boxplus$ represents any set operator, must be translated as:

$$boxplus(B, C, A)$$

where *boxplus* is the name of a $\{log\}$ operator. See the rules given below to find the right rule for each operator.

b) Arithmetic equalities. An equality of the form:

$$A = B \boxtimes C$$

where $\boxtimes$ represents any arithmetic operator, must be translated as:

$$A \ \mathsf{is}\ B \boxdot C$$

where $\boxdot$ is the $\{log\}$ representation of $\boxtimes$. See the rules given below to find the right rule for each operator.

7. If a variable declared in the declaration part of a test specification does not appear in its predicate part, then it will be translated as follows:

$$\text{unused variables}\frac{x : T}{x \;\mathsf{in}\; [\![T]\!]}$$

   where T is any Z type.

8. Auxiliary variables introduced during the translation. As it will be seen below, some of the translation rules require the introduction of new (fresh) variables. For example, rules named access to component, schema type, apply, $\not\subseteq$, $\cup$, $\cap$, set difference, $\bigcup$, $\bigcap$, dom, ran, etc. require the introduction of a new variable. For each of these variables, the translation rule corresponding to its Z type must be applied.

   For instance, in rule apply we have, as a result of the translation, the introduction of a new variable y whose Z type is the type of the range of the function which is being applied, say type T. Then, consider as if there is a declaration $y : T$ and look up if there is a translation rule for that declaration. If there is one, the apply it; otherwise do nothing.

All the predicates at the $\{log\}$ level are defined either within the interpreter, or within the standard library, or within an ad-hoc library, called the TTF library.

Notational conventions

At the Z level, we will use capital letters, such as A, B, X, Y, T, ..., to denote sets (in particular, types), while we will use lowercase letters, such as a, b, x, y, p ..., to denote single elements of any type (in particular, c_i will be used to denote constants).

The recursive application of all the rules given in this document is usually left implicit. However, when appropriate we will use the notation $[\![term]\!]$ to explicitly indicate the translation of *term* according to the rules.

In particular, we assume that Z variables and constants are always implicitly translated to the corresponding Prolog variables and constants. For the sake of simplicity, however, in this document we assume Z variables and constants are simply translated as they are, whereas in the concrete translation they will be rewritten according to the strategy described in item 3 above.

2. Ground Types

$$\mathbb{Z}\frac{\mathbb{Z}}{int(-2147483648, 2147483647)}$$

$$\text{Integer variable}\frac{x : \mathbb{Z}}{x \;\mathsf{in}\; [\![\mathbb{Z}]\!]}$$

$$\mathbb{N}\frac{\mathbb{N}}{int(0, 2147483647)}$$

$$\mathbb{N}_1\frac{\mathbb{N}_1}{int(1, 2147483647)}$$

$$\text{Natural variable}\ \frac{x : \mathbb{N}}{x\ \mathsf{in}\ [\![\mathbb{N}]\!]}$$

$$\text{Positive variable}\ \frac{x : \mathbb{N}_1}{x\ \mathsf{in}\ [\![\mathbb{N}_1]\!]}$$

For convenience, when a test specification is translated and if there are several uses of $\mathbb{Z}$ or $\mathbb{N}$, then it is possible to define, for example:

$$NAT = int(0, 21474836479)$$
$$NAT1 = int(1, 2147483647)$$
$$INT = int(-2147483648, 2147483647)$$

provided these names are not used within the test specification.

$$\text{given set}\ \frac{[X]}{set(X)}$$

$$\text{enumeration}\ \frac{X ::= c_1 | \ldots | c_n}{X = \{c_1, \ldots, c_n\}}$$

where $c_1, \ldots, c_n$ are constants.

$$\text{enumeration variable}\ \frac{x : T}{x\ \mathsf{in}\ [\![T]\!]}$$

where T is an enumerated type.

3. Set Extensions

$$\text{set extension}\ \frac{\{y_1, \ldots, y_m\}}{\{[\![y_1]\!], \ldots, [\![y_m]\!]\}}$$

where $y_1, \ldots, y_m$ are expressions that are in scope within the test specification being translated.

4. Set Comprehensions

Set comprehensions are first preprocessed as follows. Given a set comprehension of the following form:

$$\{x_1 : X_1;\ \ldots;\ x_n : X_n | P(x_1, \ldots, x_n) \bullet E(x_1, \ldots, x_n)\}$$

where P and E are a predicate and an expression, respectively.

First, the set comprehension is changed as follows:

$$\{x : X_1 \times \ldots \times X_n | P_F(x,1, \ldots, x.n) \bullet E_F(x,1, \ldots, x.n)\}$$

where E_F is a flattened version of E (i.e., all the subexpressions are replaced by fresh variables) and P_F is P conjugated with all the equalities used to flatten E.

Once this preprocessing has been done, the following rule must be applied.

$$\text{compr}\ \frac{\{x : X_1 \times \ldots \times X_n | P_F(x,1, \ldots, x.n) \bullet E_F(x,1, \ldots, x.n)\}}{\{x : \mathsf{exists}([x_1, \ldots, x_n], [\![x_1 : X_1]\!]\ \&\ \ldots\ \&\ [\![x_n : X_n]\!]\ \&\ [\![P_F(x_1, \ldots, x_n)]\!]\ \&\ x =_{\mathsf{is}} [\![E_F(x_1, \ldots, x_n)]\!]\}}$$

where $=_{\mathsf{is}}$ is either $=$ or is depending on the type of E_F (i.e. $=_{\mathsf{is}}$ is is if E_F is an arithmetic expression and $=$ otherwise).

Examples

The Z term:

$$\{x_1 : D \bullet x_1 + 1\}$$

is translated as:

```
{X : exists([Z], Z in D & X is Z + 1 )}
```

The Z term:

$$\{x_1 : D;\ x_2 : D \bullet x_1 \cup x_2\}$$

is translated as:

```
{X : exists([X1,X2,R], X1 in D & X2 in D & un(X1,X2,X))}
```

The Z term:

$$\{x_1 : D \bullet x_1 \mapsto 0\}$$

is translated as:

```
{X : exists([X1,X2], X1 in N & X2 in N & X2 is X1 + 1 & X = [X1,X2])}
```

The Z term:

$$\{x_1 : \mathbb{Z} | x_1 \mathsf{mod}\, 2 = 0 \bullet x_1 \mapsto x_1 + 1\}$$

is translated as:

```
{X :
  exists([X1,X2],
    X1 in N & X2 in N & 0 is X1 mod 2 & X2 is X1 + 1 & X = [X1,X2])}
```

5. Cross Products

In general, the type expression $X \times Y$ will not be translated because if a term belongs to such a type then it will participate in some expression involving operators of such type. If the term does not participate in such an expression then its structure is irrelevant for solving the goal. In this case when the result returned by $\{log\}$ is translated back to Z it can be replaced by a fixed constant of the type $X \times Y$.

Furthermore, if a variable is declared as $p : X \times Y$ and the test specification includes only a predicate like $p.1 \in A$, then $\{log\}$ will be able to determine only a value for the first component of p. The second component will remain a variable. For example, the result returned by $\{log\}$ can be $[a|G]$ where a is a constant that belongs to the set resulting from translating A and G is a list. In this case, when the results is translated back to Z, Fastest will replace G by any constant, y, of type Y and the net result will be $a \mapsto y$. This is sound because if $p.2$ does not appear in the test specification means that it is irrelevant for finding a test case, so any value will serve as input data.

The only cases where type information must be considered for translation are the following ones. Let t be an expression of type $X_1 \times \ldots \times X_i \times \ldots \times X_n$, and let the test specification contain an expression of the form $t.i$. If X_i is either $\mathbb{Z}$, or $\mathbb{N}$, or an enumerated type, or a binary relation, or a partial function, or a total function, then the corresponding translation rule must be applied to $t.i$. More formally,

$$\text{cross product } \pi \frac{t : X_1 \times \ldots \times X_i \times \ldots \times X_n \qquad t.i}{apply\ rule\ \pi\ to\ [\![t.i]\!]}$$

where π is any of the following translation rules, depending on X_i: rule $\mathbb{Z}$ if X_i is $\mathbb{Z}$, rule $\mathbb{N}$ if X_i is $\mathbb{N}$, rule “enumeration variable” if X_i is an enumerated type, rule $\leftrightarrow$ if X_i is a binary relation, rule $\nrightarrow$ if X_i is a partial function, rule $\rightarrow$ if X_i is a total function.

Ordered pairs are represented as lists of two elements.

$$\text{Ordered pair} \frac{x \mapsto y}{[[\![x]\!], [\![y]\!]]}$$

Therefore, access to components must be translated as follows:

$$\text{access to component} \frac{x : X_1 \times \ldots \times X_n \qquad x.i}{nth1(i, [\![x]\!], y)}$$

where y must be a new variable.

6. Power Sets

If a variable is declared as $A : \mathbb{P} X$ for some type X it is not necessary, in general, to translate this information to $\{log\}$ because it will deduce that type from the expressions where A appears.

The only cases where type information must be considered for translation are the following ones. Let t be an expression of type $\mathbb{P} X$, and let the test specification contain an expression r of type X. If X is either $\mathbb{Z}$, or $\mathbb{N}$, or a binary relation, or a partial function, or a total function, then the corresponding translation rule must be applied to r. More formally,

$$\text{power set } \pi \frac{t : \mathbb{P} X \qquad r}{apply\ rule\ \pi\ to\ [\![r]\!]}$$

where $r \in t$, and π is any of the following translation rules, depending on X: rule $\mathbb{Z}$ if X is $\mathbb{Z}$, rule $\mathbb{N}$ if X is $\mathbb{N}$, rule $\leftrightarrow$ if X is a binary relation, rule $\nrightarrow$ if X is a partial function, rule $\rightarrow$ if X is a total function.

If X is an enumerated type then apply the following rule:

$$\text{power set enumeration} \frac{t : \mathbb{P} X}{subset([\![t]\!], [\![X]\!])}$$

7. Schema Types

As with power sets and cross products it is not necessary to translate the type information of a variable whose type is a schema type. However, the operators that work on schema types must be translated. In particular, we consider the dot operator:

$$\text{schema type} \frac{x : [a_1 : T_1;\ \ldots;\ a_n : T_n] \qquad x.a_i}{nth1(i, [\![x]\!], a_i)}$$

where the components of the schema type are ordered alphabetically and $nth1(i,x,a_i)$ is true if x is a list and the i-th element of x unifies with a_i. Hence, schema types are translated as lists. At the $\{log\}$ level, a_i means a variable whose name is composed of the translation of a and i. Since lists in Prolog and $\{log\}$ are untyped, each of its elements can be of a different type.

The only cases where type information must be considered for translation are the following ones. Let x be an expression of type $[a_1 : T_1,\dots,a_i : T_i,\dots,a_n : T_n]$), and let the test specification contain an expression of the form $x.a_i$. If T_i is either $\mathbb{Z}$, or $\mathbb{N}$, or an enumerated type, or a binary relation, or a partial function, or a total function, then the corresponding translation rule must be applied to $x.a_i$. More formally,

$$\text{schema type } \pi \frac{[a_1 : T_1,\dots,a_i : T_i,\dots,a_n : T_n] \qquad x.a_i}{\textit{apply rule } \pi \textit{ to } [\![x.a_i]\!]}$$

where π is any of the following translation rules, depending on T_i: rule $\mathbb{Z}$ if T_i is $\mathbb{Z}$, rule $\mathbb{N}$ if T_i is $\mathbb{N}$, rule "enumeration variable" if T_i is an enumerated type, rule $\leftrightarrow$ if T_i is a binary relation, rule $\mathrel{\mkern3mu\joinrel\mapstochar\joinrel\rightarrow}$ if T_i is a partial function, rule $\rightarrow$ if T_i is a total function.

θ expressions are not translated because Fastest does not support them for the moment.

8. Binary Relations

If a set is a binary relation then the following rule must be applied:

$$\leftrightarrow \frac{R : X \leftrightarrow Y}{is\_rel(R)}$$

Note that the domain and range of the relation are irrelevant for the translation. These will be deduced from the expressions where the relation appears.

9. Partial Functions

If a set is a partial function then the following rule must be applied:

$$\mapstochar\rightarrow \frac{f : X \mapstochar\rightarrow Y}{\mathsf{is\_pfun}(f)}$$

Note that the domain and range of the function are irrelevant for the translation. These will be deduced from the expression where the relation appears.

Given that in $\{log\}$ all sets are finite, $\mapstochar\twoheadrightarrow$ is translated as $\mapstochar\rightarrow$.

10. Total Functions

If a set is a total function then the following rule must be applied:

$$\rightarrow \frac{f : X \rightarrow Y}{\mathsf{is\_pfun}(f)\ \&\ \mathsf{dom}(f,X)}$$

11. Function Application

Function application is translated as follows:

$$\mathsf{apply}\,\frac{f : X(\rightarrow\!\!\!\!+ \mid \rightarrow)Y \qquad f\,x}{\mathsf{apply}(f, [\![x]\!], y)}$$

where $\mathsf{apply}(f,x,y)$ is true if y is the image of x in f and y must be a new variable.

12. Sequences

$$\text{seq}\,\frac{s : \operatorname{seq} X}{\mathsf{list}(s)}$$

$$\text{seq}_1\,\frac{s : \operatorname{seq}_1 X}{\mathsf{list}(s)\ \&\ s\ \mathsf{neq}\ [\,]}$$

$$\text{explicit sequence}\,\frac{\langle x_1, \ldots, x_n\rangle}{[[\![x_1]\!], \ldots, [\![x_n]\!]]}$$

13. Equality

$$=\frac{x = y}{[\![x]\!] = [\![y]\!]}$$

$$\neq\frac{x \neq y}{[\![x]\!]\ \mathsf{neq}\ [\![y]\!]}$$

14. Set Expressions

$$\in\frac{x \in A}{[\![x]\!]\ \mathsf{in}\ A}$$

$$\notin\frac{\neg\, x \in A}{[\![x]\!]\ \mathsf{nin}\ A}$$

$$\emptyset\frac{\emptyset}{\{\}}$$

$$\subseteq\frac{A \subseteq B}{\mathsf{dsubset}(A, B)}$$

$$\not\subseteq\frac{\neg\, A \subseteq B}{dnsubset(A, B)}$$

$$\subset\frac{A \subset B}{dssubset(A, B)}$$

$$\not\subset\frac{\neg\, A \subset B}{dinters(A,B,C)\ \&\ C \neq A}$$

$$\cup\frac{A \cup B}{dun(A,B,C)}$$

where C is a fresh variable generated, along with the equality $C = A \cup B$, as a result of the preprocessing of the subexpression $A \cup B$.

$$\cap\frac{A \cap B}{dinters(A,B,C)}$$

$$\text{set difference}\frac{A \setminus B}{diff(A,B,C)}$$

$$\bigcup\frac{\bigcup A}{bun(A,S)}$$

$$\bigcap\frac{\bigcap A}{bdinters(A,S)}$$

15. Relational Expressions

$$\text{dom}\frac{R : X \leftrightarrow Y \qquad \operatorname{dom} R}{dom(R,D)}$$

$$\text{ran}\frac{R : X \leftrightarrow Y \qquad \operatorname{ran} R}{ran(R,D)}$$

$$\fatsemi\frac{Q : X \leftrightarrow Y \qquad R : Y \leftrightarrow Z \qquad Q \fatsemi R}{comp(Q,R,S)}$$

$$\circ\frac{Q : X \leftrightarrow Y \qquad R : Y \leftrightarrow Z \qquad Q \circ R}{comp(R,Q,S)}$$

$$\lhd\frac{R : X \leftrightarrow Y \qquad A : \mathbb{P}X \qquad A \lhd R}{dres(A,R,S)}$$

$$\rhd\frac{R : X \leftrightarrow Y \qquad A : \mathbb{P}Y \qquad R \rhd A}{rres(A,R,S)}$$

$$\ndres\frac{R : X \leftrightarrow Y \qquad A : \mathbb{P}X \qquad A \ndres R}{ndres(A,R,S)}$$

$$\nrres\frac{R : X \leftrightarrow Y \qquad A : \mathbb{P}Y \qquad R \nrres A}{nrres(A,R,S)}$$

$$\sim\frac{R^{\sim}}{inv(Q,R)}$$

$$(\!|\,|\!)\frac{R : X \leftrightarrow Y \qquad A : \mathbb{P}X \qquad R(\!|A|\!)}{rimg(R,A,B)}$$

$$\oplus\frac{R \oplus G}{oplus(R,G,S)}$$

16. Sequence Expressions

Although in Z sequences are just special sets, for reasons of efficiency we treat them as Prolog lists and whenever necessary we convert them to $\{log\}$ sets by the following rule:

$$\text{list to rel}\ \frac{s : \mathrm{seq} X}{list\_to\_rel(s,x)}$$

This rule must be applied every time a sequence participates in a set expression. However, if it participates in a purely sequence expression then it must be treated as a sequence.

Also for reasons of efficiency we have defined special predicates for the domain, range and application of a sequence.

$$\text{sequence domain}\ \frac{s : \mathrm{seq} X \qquad \mathrm{dom}\, s}{dom\_list(s,dom)}$$

$$\text{sequence range}\ \frac{s : \mathrm{seq} X \qquad \mathrm{ran}\, s}{list\_to\_set(s,ran)}$$

$$\text{sequence apply}\ \frac{s : \mathrm{seq} X \qquad s\ i}{nth1(i,s,y)}$$

$$\frown\ \frac{s \frown t}{append(s,t,u)}$$

$$\text{reversal}\ \frac{rev\ s}{reverse(s,rev)}$$

$$\text{head}\ \frac{head\ s}{nth1(1,s,head)}$$

$$\text{last}\ \frac{last\ s}{last(s,last)}$$

$$\text{tail}\ \frac{tail\ s}{drop(1,s,tail)}$$

$$\text{front}\ \frac{front\ s}{length(s,n)\ \&\ take(n-1,s,front)}$$

$$\upharpoonleft\ \frac{s : \mathrm{seq} X \qquad A : \mathbb{P}\mathbb{Z} \qquad A \upharpoonleft s}{extract(A,s,t)}$$

$$\upharpoonright\ \frac{s : \mathrm{seq} X \qquad A : \mathbb{P} X \qquad s \upharpoonright A}{filter(s,A,t)}$$

$$\text{squash}\ \frac{s : \mathbb{N}_1 \twoheadrightarrow X \qquad squash\ s}{squash(s,t)}$$

$$\text{prefix}\ \frac{s\ \mathsf{prefix}\ t}{append(s,\_,t)}$$

$$\text{suffix}\ \frac{s\ \mathsf{suffix}\ t}{append(\_,s,t)}$$

$$\mathsf{in}\ \frac{s\ \mathsf{in}\ t}{sublist(s,t)}$$

17. Arithmetic Expressions

$$\text{+}\,\frac{a+b}{a+b}$$

$$\text{-}\,\frac{a-b}{a-b}$$

$$*\,\frac{a*b}{a*b}$$

$$\text{div}\,\frac{a \mathsf{\,div\,} b}{div(a,b)}$$

$$\text{mod}\,\frac{a \mathsf{\,mod\,} b}{mod(a,b)}$$

$$\texttt{>}\,\frac{a \;\texttt{<}\; b}{a \;\texttt{<}\; b}$$

$$\texttt{>}\,\frac{a \;\texttt{>}\; b}{a \;\texttt{>}\; b}$$

$$\leq\,\frac{a \leq b}{a \;\texttt{=<}\; b}$$

$$\geq\,\frac{a \geq b}{a >= b}$$

$$\text{ranges}\,\frac{a\,..\,b}{int(a,b)}$$

$$\#\,\frac{A : \mathbb{F}X \qquad \#A}{size(A,N)}$$

$$\text{min}\,\frac{min\,A}{min(A,Min)}$$

$$\text{max}\,\frac{max\,A}{max(A,Max)}$$

18. Translation from $\{log\}$ to Z

A Z test specification is translated through the translation rules to a {log} goal. Executing this goal by means of the {log} interpreter will produce as its output a sequence of equalities $X_1 = v_1, \ldots, X_n = v_n$, where $X_1, \ldots, X_n$ are variables and $v_1, \ldots, v_n$ are either variables or constants, along with a (possibly empty) sequence of irreducible $\{log\}$ constraints of some specific forms. Variables occurring in the {log} output can be either *Z variables*, that is variables that correspond to a variable declared in the test specification, or *auxiliary variables*, that is variables that appear in the {log} output but are not Z variables. Constants occurring in the $\{log\}$ output can be either atomic values, e.g. integer numbers, or they can denote structured values, such as set or list expressions.

The translation rules from $\{log\}$ to Z are as follows:

1. Each *constant* returned by $\{log\}$ is a constant that comes from the translation from Z into $\{log\}$. Hence, the translation of a constant is simply its Z name.
2. Each *Z variable* must be translated into its Z name. If M is a variable at the $\{log\}$ level then $[\![M]\!]$ denotes its translation into Z.
3. If M is an *auxiliary variable*, then it must be treated as a constant of the corresponding Z type. For example, if at the Z level we have

   $$[ACCNUM, UID]$$

   along with a variable in a test specification such as $owners : UID \leftrightarrow ACCNUM$, and $\{log\}$ returns as part of its output the equality:

   $$Owners = \{[U,N]\},$$

   where U and N are not variables of the test specification, then U and N must be considered as constants of type UID and $ACCNUM$, respectively.

   In doing so we must consider the following cases:

   - The type of M at the Z level[1] is a *basic type* called T. Then the translation of M is the string TM.
   - The type of M at the Z level is $\mathbb{Z}$ *or* $\mathbb{N}$. Then the translation of M is any natural number.
   - The type of M at the Z level is an *enumerated type*. Then the translation of M is one of the constants of the type.
   - The type of M at the Z level is *any other type*. In this case build the translation of M by recursively applying the other rules.

     Note: if the type of M at the Z level is a schema type, then a constant of this type at the Z level is represented as follows:

     $$\lblot x_1 == c_1; \ldots; x_n == c_n \rblot$$

     where $x_1, \ldots, x_n$ are the components of the schema type and $c_1, \ldots, c_n$ are the constants generated during the translation[2].

[1]Note that although M is not a Z variable, its Z type can be deduced from the expression where it appears in the $\{log\}$ output.

[2]The right way of writing $\lblot$ in LaTeX is `\lblot` and $\rblot$ is `\rblot`.

This is true, in particular, for variables of the form *_Gnumber* which are automatically generated by $\{log\}$ (for instance, $\_G251$).

However, if M participates in an irreducible constraint of the form $M \neq t$ where t is either a variable or a constant, then see rule 4.

4. If M is a variable and it is never at the left hand side of one of the equalities returned by $\{log\}$, then its value is given by a set of zero or more so-called irreducible constraints. These variables must be translated before all the other Z variables. This is so because the variables not appearing at the left hand side can be part of the value of the other variables.

   In this case the following translation rules apply:

   - Zero irreducible constraint is given. In this case apply rule 3.
   - $list(M)$. Translate it as $[\![M]\!] = \langle\rangle$, only if M is a Z variable.
   - $set(M)$. Translate it as $[\![M]\!] = \emptyset$, only if M is a Z variable.
   - $integer(M)$. Translate it as $[\![M]\!] = 0$, only if M is a Z variable.
   - $\neq$. Assume the variable we are interested in is M. Therefore, in this case, there is a sequence of inequalities of the form $M \neq B_i$ where B_i are either variables or constants, for all i in some set I. In turn, these other terms B_i can participate in other inequalities where M does not participate. This set of inequalities is trivially solved if all the variables (M, all the B_i, and all the other variables participating in inequalities where a B_i also participates) are assigned different constants. Thus, the point is to provide translation rules as to generate different constants for each variable involved in an inequality.

     The translation rules are as follows. Let A be a variable or a component of a cross product or a schema type[3] participating in an inequality and let K_A be the set of all constants occurring in all inequalities which are directly or indirectly connected with A. For example, if we have the following sequence of inequalities $M \neq a, M \neq b, M \neq Z, Z \neq c, W \neq Z, W \neq d, N \neq e$, then the set K_M for the variable M is $\{a,b,c,d\}$.

     a) If the type of A at the Z level is a basic type apply the first case of rule in 3.
     b) If the type of A at the Z level is $\mathbb{Z}$ or $\mathbb{N}$, then assign to it a natural number n such that $n \notin K_A$.
     c) If the type of A at the Z level is an enumerated type, then assign to it a constant c of the enumerated type such that $c \notin K_A$.
     d) If the type of A at the Z level is a structured type (i.e. one obtained by applying a type constructor), then proceed as follows:

       I) If the type is $\mathbb{P}X$, $\mathbb{F}X$ or $\mathrm{seq}X$, then assign to A a set or a sequence s such that $\#s \neq \#r$ for each $r \in K_A$ of the same Z type as s.
       II) If the type is $X \times Y$ or a schema type, then consider one of the components, say X_i (preferentially one of type $\mathbb{Z}$ or $\mathbb{N}$), and the relevant set K_{X_i}, and assign to it a constant c such that $c \neq d$ for each $d \in K_{X_i}$ of the same Z type as c. Assign to the remaining components a fixed constant by applying the corresponding rule given in 3.

[3] If A is a component then we mean the same component of the same type. For example, if we have $x : X \times Y$ and $y : X \times Z$ and we take A as $x.1$ then we do not talk about $y.1$ although they have the same type. That is, consider components as named variables.

Consider the following examples. If at the Z level we have:

$$[X]$$

and a test condition as:

$$\begin{array}{l} A : \mathbb{P}(\mathbb{P}\,X) \\ \#A > 2 \end{array}$$

then $\{log\}$ will return:

```
A = {_G2676,_G2661,_G2646},
N = 3
Constraint: _G2676 neq _G2661, _G2676 neq _G2646, _G2661 neq _G2646
```

Then the test case at the Z level will be:

$$A = \{\emptyset, \{X1\}, \{X1, X2\}\}$$

where $X1$ and $X2$ are assumed to be two elements of X.

As another example say we have the same test condition but X now is as follows:

$$\boxed{\begin{array}{l} X \\ \hline x : \mathbb{N} \\ y : \operatorname{seq} \mathbb{Z} \end{array}}$$

then, since the translation of the test specification will be the same, the answer returned by $\{log\}$ will be the same, too. However, in this case the translation from $\{log\}$ to Z will yield:

$$A = \{\langle\!| x : \{0\};\ y : \{\langle\rangle\} |\!\rangle, \langle\!| x : \{1\};\ y : \{\langle\rangle\} |\!\rangle, \langle\!| x : \{2\};\ y : \{\langle\rangle\} |\!\rangle\}$$

because the type of A at the Z level is different in the two examples.

5. If M is a Z variable at the left hand side of one of the equalities returned by $\{log\}$ whose type is a cross product or a schema type, then consider the following cases:

   a) The type of M is $X_1 \times \ldots \times X_n$. At the right hand side of the equality there will be a $\{log\}$ list of the form $[G_1, \ldots, G_m]$ with $m \leq n$. Then the translation must be as follows:

   $$[\![M]\!] = ([\![G_1]\!], \ldots, [\![G_m]\!], c_{m+1}, \ldots, c_n)$$

   where $c_{m+1}, \ldots, c_n$ are constants whose types are, respectively, $X_{m+1}, \ldots, X_n$. These constants must be built as indicated in rule 3. Note that $G_1, \ldots, G_m$ may be auxiliary variables so rules 3 and 4 may apply to them.

   If $n = 2$ the translation can be $[\![G_1]\!] \mapsto [\![G_2]\!]$, that is mapplet is used in place of parenthesis.

b) The type of M is $[a_1 : T_1;\ \dots;\ a_n : T_n]$. At the right hand side of the equality there will be a $\{log\}$ list of the form $[G_1, \dots, G_m]$ with $m \leq n$. Then the translation must be as follows:

$$[\![M]\!] = \langle\!| a_1 : \{[\![G_1]\!]\};\ \dots;\ a_m : \{[\![G_m]\!]\};\ a_{m+1} : \{c_{m+1}\};\ \dots;\ a_n : \{c_n\} |\!\rangle$$

where $c_{m+1}, \dots, c_n$ are constants whose types are, respectively, $T_{m+1}, \dots, T_n$. These constants must be built as indicated in rule 3. Note that $G_1, \dots, G_m$ may be auxiliary variables so rules 3 and 4 may apply to them.

6. *Sequences of equalities* of the form:

$$M_1 = M_2, \dots, M_{n-1} = M_n, M_n = c$$

where $M_1, \dots, M_n$ are n variables and c is a constant, are going to be translated as follows:

$$\begin{array}{l} [\![M_1]\!] = [\![c]\!] \\ [\![M_2]\!] = [\![c]\!] \\ \dots \\ [\![M_n]\!] = [\![c]\!] \end{array}$$

7. *Set extensions* of the form $\{c_1, \dots, c_n\}$ are translated as $\{[\![c_1]\!], \dots, [\![c_n]\!]\}$. If $n = 0$ and thus the set is empty, the translation can be $\{\}$ or $\emptyset$.
8. *List expressions* of the form $[c_1, \dots, c_n]$ are translated as $\langle [\![c_1]\!], \dots, [\![c_n]\!] \rangle$, if they are not used to represent cross products or schema types (in which case rule 5must be applied). If $n = 0$ and thus the list is empty, the translation is $\langle\rangle$.
9. *Integer numbers* are translated as they are.
10. The translation of *set expressions* of the form $\{c_1, \dots, c_n \backslash S\}$ is $[\![\{c_1, \dots, c_n\}]\!]$ (i.e. S is ignored).
11. The translation of *list expressions* of the form $[c_1, \dots, c_n \mid S]$ is $[\![\,[c_1, \dots, c_n]\,]\!]$ (i.e. S is ignored).