Animation of Z with Prolog – A Survey

Formal Specifications and Logic Programming,

AFCET, June 26th 1997

Margaret West[a]

School of Computing and Mathematics, University of Huddersfield

email : `M.M.West@hud.ac.uk`

http://www.hud.ac.uk/schools/comp+maths/staff/mmwest.htm

[a] IMPRESS project.

Talk Structure

1. Animation methods - requirements
2. Methods: Stepney et al., SuZan project, Ez, Structure Simulation
3. Correctness criteria
4. Abstract approximation
5. Logic Programming Language – Declarative Semantics
6. Small Example
7. Evaluation in the LP: Schemas and Expressions
8. Approximation

Animation for Validation

Validation – ‘validate’(Latin) ‘to confirm or ratify’, requiring an informed decision to be made.

Rushby [Rus93] – *challenging the specification.*

> The idea of a challenge is to pose a question that an adequate specification should be able to answer.

. . . hence animation.

Requirements for Z Animation

Surveys on animation of formal specification (including Z to Prolog) conducted in [BB94, GH96, DN91].
Breuer and Bowen have identified some requirements for executing Z. They include *coverage* (of the Z grammar), *sophistication* (less likely to go into an infinite loop), *efficiency* (performance) and *correctness* (which has received the least attention). Criteria for correctness are presented later in this talk.

A declarative language such as Prolog is particularly suitable for the animation of Z, for a range of "what if" queries can be used to test and hence explain the specification.

Access Control System

1987 [SL87]: a line-by-line translation of a specific system. There are not many details – a simplified e.g. :

$$Unsuccessful\_login \mathrel{\widehat=} [\, p? : AE;\ q? : QOS;\ sid! : SID \mid p? \notin AE \wedge sid! = Null \,]$$

```
unsuccessful_login(P, S, QOS, SID) :-
    enty(E), not(member(P, E)), SID = sid0,
    report(‘‘person unknown, login abandoned’’).
```

Some clauses provide a screen display e.g. ‘report’.

The Alvey ‘SuZan’ Project (method)

1989 [DKC89]: ‘Pure’ Prolog and ‘generate and test’ method, e.g. if [*PHONE*] is a basic type, then a connection is defined: $CON \mathrel{\widehat{=}} \mathbb{P}\,PHONE$. In Prolog this is:

```
phones([1,2,3]). %% set is finite list
connections(Cons):- phones(Ps), powerset(Ps, Cons).
```

‘`powerset`’ generates $\mathbb{P}\{1,2,3\}$.
$Rel : X \leftrightarrow Y$:

```
%% If sets X, Y are ground, the following will
generate a relation Rel, or test Rel:


is_a_relation(Rel, X, Y) :-
   cart_prod(X,Y,CP),   has_subset(Rel,CP).
```

'SuZan' Project –efficiency and coverage

efficiency improved by utilising correctness preserving program transformations:

- re-ordering code;
- fold/unfold and removal of duplicate clauses;
- transformation of the Z

Coverage – large library of Prolog for set operators, but for schema calculus only a suggested implementation.

‘SuZan’ Project – telephone network example

(from [KK88].)

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \; TN \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
reqs, cons : \mathbb{P}\, CON \\
\hline
cons \subseteq reqs \land \mathsf{disjoint}\ CON \\
\hline
\end{array}
$$

The use of $\forall\, s : \mathbb{P}\, X \bullet s \subseteq X$ produces a simpler translation.

```
tn(Reqs, Cons):-
   connections(CON), has_subset(Reqs, CON),
   has_subset(Cons, CON), has_subset(Cons, Reqs),
   disjoint(Cons).
```

[DN91] also use ’generate and test’ in a similar method.

Structure Simulation – West and Eaglestone

Z animation in Prolog ([WE92]).

1. *Formal Program Synthesis* - The transformation of standard logic to its clausal form and (hence) Prolog formed the basis of the original translation technique. This was an attempt at a correct animation – it was abandoned as being infeasible at that time (1988).

2. *Structure Simulation* - more successful. This was an attempt to 'simulate' the structure of Z, using Prolog.

Variables and schema are *named* so they can be referenced by other schema. The set-theory operations were contained in a separate collection or library of Prolog code which had been created for the purpose.

Structure Simulation – Example

$$
\begin{array}{|l}
\multicolumn{1}{l}{\text{\_\_ } \mathit{AddOne} \text{ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_}} \\
x, x' : 0 \mathrel{..} 10 \\
\hline
x' = x + 1 \\
\hline
\end{array}
$$

```
schema_type([X, X1 ], addone) :-
     givenset(T, one_to_ten),     %% given sets
     varname(x, X), member(X, T), %% variable bindings
     varname(d(x), X1), member(X1, T), %% and types
     X1 is X + 1.                      %% predicate
```

This allows the linking of variables and (hence) schema calculus. Variables in each list `L` of `schema_type(L, N)` are instantiated via `varname` using some suitable test strategy.

Structure Simulation – EC DRIVE 1 Project
Pedestrian Crossing Case Study

The method was used to animate a Z specification of a pedestrian crossing (1991) – described in [WHJ95] .
The UK Department of Transport had been having problems with the equipment: there had been reports of the audible signal continuing to sound when the red and green lights had been turned *off* (in the case of bulb failure for example.)
We were able to aid the authorities in solving the problem and creating a new equipment specification.

Criteria for Correctness

Proof basis for correct animations of Z [BB94] based on the notion of *abstract interpretation*, viz. *Abstract Approximation.* The proof basis is (informally) – an animation of a Z specification must not contain any more information than the original Z specification, as it may mislead. A computation may result in an output which is incomplete, viz enter an infinite loop. This answer is less *refined* than that obtainable via set theoretic considerations. E.g.

$$\{1, 2, 3\}_{\cup\perp} \textit{ is refined by } \{1, 2, 3, 4\}$$

Abstract Approximation - Breuer and Bowen

- Concrete interpretation *refines* the abstract.
- *ZF* interpretation, in the Z domain *ideal* - evaluation in set-theoretic terms of *schemas, expressions* and *predicates* (Other parts of Z expanded out - schema calculus.)
- Non-termination or incomplete information, is possible in abstract domain, *output*; a partial element $\perp$ is introduced.
- Correspondingly, the sets of *ideal* are 'lifted" partial element $\perp$
- Sets can be *incomplete*, denoted $S_{\cup\perp}$. $\sqsubseteq$ on *ideal* is equality on integers, and co-ordinatewise on tuples. It uses a 'standard' powerdomain ordering on subsets.

D_Z and D_{LP}

- Concrete Z domain D_Z corresponds to *ideal* and the abstract domain is D_{LP}, the output of an animation of a Z specification via a logic program.
- A 'concretisation function' (denoted γ) is introduced so that evaluations in D_{LP} and D_Z can be compared.
- $\gamma : D_{LP} \nrightarrow D_Z$, is chosen so that it is appropriate for the *declarative semantics* of logic programming.

Approximation Diagram for LP and Z domains

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\rho_{LP} \arrow[r, "\mathcal{E}_{\mathcal{LP}}[\![\epsilon]\!]\_"] \arrow[d, dashed, "\gamma \circ \_"'] & D_{LP} \arrow[d, dashed, "\gamma"] \arrow[d, phantom, "\sqsupseteq"{pos=0.9, xshift=2em}] \\
\rho_{Z} \arrow[r, "\mathcal{E}_{\mathcal{Z}}[\![\epsilon]\!]\_"'] & D_{Z}
\end{tikzcd}
$$

If ϵ is a syntactic Z expressions, the interpreted evaluation in the output LP domain of ϵ is denoted $\mathcal{E}_{\mathcal{LP}}[\![\epsilon]\!]\rho_{\mathcal{LP}}$, where ρ_{LP} is the LP 'environment'. The evaluation in the Z domain is similarly denoted.

Abstract Approximation - LP (work in progress)

Some preliminary work has been done on Abstract Approximation using a *Logic Programming Language* (LP) and its *declarative semantics*, and a pilot study undertaken, in implementing the declarative semantics of the animation in the LP, Gödel [Wes95].

The logic programming language (LP) is assumed

- to allow *negation*,
- to have built in constructor functions which allow for *set terms*.
- Equality (unification) on sets allows for duplication and permutation of elements as in the LP Gödel.
- It is assumed that the (sequential or parallel) implementation is sound with respect to the semantics.

Example of Gödel Code 1

In Gödel a predicate is defined by a declaration specifying the type(s) of its arguments, and a set of statements of the form `Head <- Body`, where `Head` is an atom, `Body` is a formula in FOL, `<-` means "if", upper case - constants, lower case - variables. Schema defined previously is animated in Gödel:

```
PREDICATE SchemaType: Set(BindVar) * SchNames.
FUNCTION    BindInt : SetNames * Integer ->  BindVar.
FUNCTION Pr : SetNames  -> SetNames.

SchemaType(binding, AddOne)  <-
      binding = {BindInt(X, x), BindInt(Pr(X), x1)} &
      x In {y: 0 =< y =< 10} &  x1 In {y: 0 =< y =< 10} &
        x1 = x + 1.
```

Example of Gödel Code 2

```
%% Type constructor OP allows the definition of Goedel function
%% OrdPair. This is a device to provide for ordered pairs and
%% functions (a set of ordered pairs) in Z.
CONSTRUCTOR OP/2.
FUNCTION    OrdPair : a * b ->  OP(a,b).

%%%%% declaration of partial function %%%%%
PREDICATE   PF  : Set(OP(a,b)) * Set(a)  * Set(b).
%%%%% definition of partial function %%%%%
 PF(pf, s1, s2) <- ALL [z,x,y] (z In pf &
                                 (z = OrdPair(x,y))
              ->  (x In s1) & (y In s2)  &
              ALL [u] (OrdPair(x, u) In pf -> u = y)).
```

```
[Demo4] <- binding = {BindInt(X,1), BindInt(Pr(X),2)}  &
                     SchemaType(binding, AddOne) .

binding = {BindInt(X,1),BindInt(Pr(X),2)} ?
Yes

[Demo4] <-  SchemaType(binding, AddOne) .
   binding = {BindInt(X,0),BindInt(Pr(X),1)} ?
Yes
[Demo4] <- binding = {BindInt(X,1), BindInt(Pr(X),3)}  &
                                SchemaType(binding, AddOne) .

No
```

Evaluation in the LP 1

$$Env ::= (x_1 = t_1) \wedge \ldots \wedge (x_m = t_m) \wedge true$$

where, as in [And90], $\mathrm{dom}\,\rho_{LP}$ is $\{x_1, \ldots, x_m\}$ and $t_1, \ldots, t_m$ are terms of D_{LP}.

E.g. predicate $P(z, x_1, \ldots, x_k)$ could interpret a set operation such as set union, then
$?\, Env_{LP} \,\&\, P(z, x_1, \ldots, x_k)$ 'evaluates' z:

$$\exists\, z(Env_{LP} \,\&\, P(z, x_1, \ldots, x_k))$$

Evaluation in the LP 2

$\mathcal{E}_{\mathcal{LP}}[\![\epsilon]\!]$ gives the interpretation in D_{LP} of syntactic expressions ϵ in an environment ρ_{LP}. If z denotes ϵ then $\mathcal{E}_{\mathcal{LP}}[\![\epsilon]\!]$ is defined:

$$\mathcal{E}_{\mathcal{LP}}[\![\epsilon]\!]\rho_{LP} = t \Leftrightarrow (z = t) \wedge Env \wedge P(z, x_1, \ldots, x_k)$$

$$\mathcal{E}_{\mathcal{LP}}[\![x_i]\!]\rho_{LP} = t_i \Leftrightarrow (x_i = t_i) \wedge true,\ (x_i \in \mathrm{dom}\,\rho_{LP}).$$

Environments in Z and in LP

- Z expressions are evaluated (using set-theoretic considerations) in environment $\rho_Z : VAR \rightarrowtail D_Z$.
- The output (D_{LP}) involves an environment $\rho_{LP} : VAR \rightarrowtail D_{LP}$
- If $x : VAR$; $val : D_Z$; $t : D_{LP}$
  $\gamma(\rho_{LP}(x)) = \gamma(t) = val$
  $\rho_Z = \gamma \times \rho_{LP}$.

Concretisation function γ

γ maps recursively as follows:

- integers in the LP output map to integers in Z ,
- set terms map to sets,
- functions of arity n maps to n-tuples,
  e.g. $\gamma(T2(t_1, t_2)) = T2(\gamma(t_1), \gamma(t_2))$
- The non-termination value $\perp$ in D_{LP} maps to $\perp$ in D_Z.
  (Note that sets, function, and so on can involve elements which are incomplete.)

The representation of schemas in D_{LP} will be explained later.

Schemas in Z

$Schema \mathrel{\widehat{=}} [\, D_1;\ \ldots;\, D_n \mid CP_1;\ \ldots;\, CP_m \,]$
where each D_i is a declaration of the form $x_i : \tau_i$, where τ_i is set valued, and each CP_i is a predicate constraining the variables of the schema as in the example.

(See the Z draft standard.)

Evaluation in the LP of Z Schemas

In particular a *schema* evaluates to a set expression, of bindings of variables to values. A schema is interpreted in the LP via. its characteristic predicate: $Schema \Leftarrow A_1 \wedge \ldots \wedge A_s$, where the variables of $Schema$ correspond to the declared schema variables and each A_i is an atom. For each declaration D_i, there is a corresponding atom, declaring the variable type; other atoms interprate the CP_i in the schema predicate.

Evaluation in the LP of Z Expressions and Predicates

For each $i, 1 \leq i \leq s$, the predicate P_i in A_i is defined by a set of statements of the form $A \Leftarrow \mathcal{F}$ in the program, where $\mathcal{F}$ is a formula in FOL and A is an atom with P_i. Atoms can either be used to *check* variable values from ρ_{LP}, or, constructively to evaluate them and augment ρ_{LP}.

Schema Binding in an Environment

Assume *Env* is a formula of the form (cf Andrews):.

$$Env \;:= (x_{i1} = t_{i1}) \wedge \ldots \wedge (x_{im} = t_{im}) \wedge true$$

where $t_{i1}, \ldots, t_{im}$ are terms of D_{LP}.

A goal (query)

$$?Env \wedge Schema$$

has, a (possibly empty) set of answer substitutions. Each of these corresponds to a binding σ for the schema so that $(Env \wedge Schema)\sigma$ is a logical consequence of the program.

Schema Predicate

Recall that each CP_i in the schema predicate is interpreted by P_i in the LP. $P(z, x_1, \ldots, x_k)$ could interpret an operation such as addition. Assuming that a correct answer for $P(z, t_1, \ldots, t_k)$ is a binding, $z = t$:

$$\exists\, z(Env \wedge P(z, x_1, \ldots, x_k))$$

where t is a term of D_{LP}.

Schema Declaration

$\mathcal{D}[\![x : \tau]\!]$ gives the interpretation of declarations $x : \tau$, τ is set-valued.

τ is a *Given Set:*

$$\mathcal{D}[\![x : \tau]\!]\rho_{LP} = t \iff$$

$$(x = t) \wedge x \in \tau \wedge Env$$

τ is a *Cartesian Product*, $\tau_1 \times \tau_2$:

$$\mathcal{D}[\![x : \tau_1 \times \tau_2]\!]\rho_{LP} = T2(t_1, t_2) \iff$$

$$(x = T2(t_1, t_2)) \wedge (t_1 = \mathcal{D}[\![z_1 : \tau_1]\!]\rho_{LP}) \wedge$$

$$(t_2 = \mathcal{D}[\![z_2 : \tau_2]\!]\rho_{LP})$$

Approximation 1

Suppose $\mathcal{E}_{\mathcal{Z}}[\![\epsilon]\!]\rho_Z$ interprets ϵ in D_Z, using set-theoretic considerations. In order to prove correctness it is necessary to show that the interpretation in D_{LP} is built recursively for each operator of Z represented above. Criteria also include the following approximation rule:

$$\gamma(\mathcal{E}_{\mathcal{LP}}[\![\epsilon]\!]\rho_{LP}) \sqsubseteq \mathcal{E}_{\mathcal{Z}}[\![\epsilon]\!](\gamma \times \rho_{LP}).$$

Approximation 2

For example evaluating "$2x$" where x has value n, in an implementation where $2n$ exceeds the largest integer available in the system may result in non-termination:
$\perp = \gamma(\perp) = (\gamma(\mathcal{E}_{\mathcal{LP}}[\![2x]\!]\rho_{LP})) \sqsubseteq 2n$ (the value obtained when interpreting in D_Z, the concrete interpretation).

Future Work

Involves:

1. Other parts of Z syntax e.g. given sets
2. Proofs -approximation
3. Gödel implementation
4. Automated translation - aided by Gödel facility for meta programming to expand Z expressions using schema calculus.

References

[And90] J H Andrews. Proof-Theoretic Characterisations of Logic Programming. In *Proceedings of the 14th International Symposium on the Mathematical Foundations of Computer Science, vol. 379 of LNCS*, pages 145–154. Springer, 1990.

[BB94] P T Breuer and J Bowen. Towards Correct Executable Semantics for Z. In *Z User Workshop, Cambridge, June 1994*, pages 185–209. Springer-Verlag, 1994.

[BN92] S M Brien and J E Nicholls. Z Base Standard. Technical Report PRG-107, Oxford University Computing Laboratory, Oxford, Nov 1992.

[CC92] P Cousot and R Cousot. Abstract Interpretation and Application to Logic Programs. *The Journal of Logic Programming*, 13:103–179, 1992.

[DKC89] A J Dick, P J Krause, and J Cozens. Computer Aided Transformation of Z into Prolog. In *Proceedings of the 4th Annual Z*

Users Meeting, Oxford University Computing Laboratory PRG, pages 71–85. Springer-Verlag, December 1989.

[DN91] V Doma and R Nicholl. EZ: A System for Automatic Prototyping of Z Specifications. In S Prehn and W J Toetnel, editors, *VDM '91, Formal Software Development Methods, Fourth International Symposium of VDM Europe, October 21-25 1991, Noordwijkerhout, The Netherlands*, volume 1, pages 189–203. Lecture Notes in Computer Science 551, Springer-Verlag, Germany, 1991.

[GH96] A Gravell and P Henderson. Executing formal specifications need not be harmful. *Software Engineering Journal*, 11(2):104 –110, March 1996.

[HL94] P M Hill and J W Lloyd. *The Gödel Programming Language*. MIT Press, 1994.

[KK88] R D Knott and P J Krause. On the Derivation of an Effective Animation: Telephone Network Case Study. Technical Report A1.3, Alvey Project SE/065, University of Surrey, 1988.

[Llo87] J W Lloyd. *Foundations of Logic Programming (Second, Extended Edition)*. Springer-Verlag, Berlin, 1987.

[Mn92] K Marriot and H Søndergaard. Bottom-up Dataflow Analysis of Normal Logic Programs. *The Journal of Logic Programming*, 13:181–204, 1992.

[Rus93] J Rushby. Formal Methods and the Certification of C ritical Systems. Technical Report CSL-93-7, Computer Science Laboratory, SRI International, Menlo Park CA 94025 USA, December 1993.

[Sch86] D A Schmidt. *Denotional Semantics, A Methodology for Language Development*. Allyn and Bacon, Inc, 7 Wells Avenue, Newton, Massachusetts, US, 1986.

[SL87] S Stepney and S P Lord. Formal specification of an access control system. *Software - Practice and Experience*, 17(9):575 – 593, 1987.

[WE92] M M West and B M Eaglestone. Software Development: Two Approaches to Animation of Z Specifications Using Prolog. *Software Engineering Journal*, 7(4):264–276, July 1992.

[Wes95] M M West. Types and Sets in Gödel and Z. In J P Bowen and M G Hinchey, editors, *ZUM'95 – 9th International Conference of Z User's, September 1995, Limerick, Ireland*, pages 389–407. Lecture Notes in Computer Science 967, Springer-Verlag, Heidelberg, 1995. A version is available as technical report 94.33, ftp://agora.leeds.ac.uk/scs/doc/reports/1994.

[WHJ95] M M West, K M Hobley, and P H Jesty. The Use of Formal Methods in ATT Systems, a Safety-critical Case Study. Technical Report 95.29, School of Computer Studies, University of Leeds, October 1995. ftp://agora.leeds.ac.uk/scs/doc/reports/1995.