Proceedings of the Third Annual **Z** Users Meeting

at the

Department of External Studies
Rewley House, 1 Wellington Square, Oxford

on

Friday, 16th December 1988

compiled, edited and written by
Jonathan Bowen

Programme Committee

Jonathan Bowen
John Nicholls
Jim Woodcock

Oxford University Computing Laboratory
Programming Research Group
8–11 Keble Road
Oxford OX1 3QD

Further copies of this Proceedings, PRG monographs and other **Z** related material are available. For more information, please contact:

The Librarian
Oxford University Computing Laboratory
Programming Research Group
8–11 Keble Road
Oxford OX1 3QD
England

Tel: 0865-273837 (librarian)
0865-273838 (general enquiries)
Fax: 0865-273839
Email: `library@uk.ac.oxford.prg` (JANET)

Contents

1 Introduction

On 16th December 1988, the third annual **Z** Users Meeting was held at 1 Wellington Square, Oxford. John Nicholls, a Research Officer in the Programming Research Group (PRG) at Oxford University, chaired the meeting, as last year. Over 90 delegates from industry, academia and other research establishments were present representing a good cross-section of those who use and are interested in **Z**.

1.1 Chairman's address *John Nicholls, PRG*

John Nicholls welcomed delegates to this, the third **Z** Users Meeting. There were several new aspects of this meeting, including the organisation of *two* Tutorials (one on refinement, the other on proof) and a Workshop on "**Z** in the development process". Feedback from attendees at these events indicated that they had been successful.

One purpose of the **Z** Users Meetings is to exchange information on progress and achievements by members of the **Z** community. This year, we are pleased to welcome a number of new users, and also to see that established users have made significant advances.

The meetings also provide an opportunity for attendees to express opinions or concerns about the direction of **Z**. In previous meetings, concerns have been expressed about the need to stabilise and standardise the notation, the inclusion of facilities for concurrency and refinement, and the provision of **Z** tools. In the past year, progress has been made on all of these topics, and a particularly noteworthy event has been the completion and publication of the **Z** *Reference Manual*, an important step towards the standardisation of **Z**. At the time of the meeting, discussions were being held about an IED project for **Z** standards and **Z** tools, and a brief review of this project is made in the meeting.

Coming fresh from attending the workshop at which the software development process was being discussed, (the speaker John Nicholls) was tempted to consider the process by which a notation and method such as **Z** is developed. We can distinguish the following stages:

1. "It exists." The objectives and philosophy are established.
2. "It is thus." There is then a period of stabilisation and consolidation.
3. "It is used like this."
4. "And it has this value." Finally there is an assessment phase.

Z has firmly passed the first stage, is well into the second stage, but has only started to enter the last two stages.

1.2 Keynote speech *Martyn Thomas, Praxis Systems plc*

Martyn Thomas, the Chairman of Praxis Systems plc, was invited to give the keynote speech for the day. He started by thanking the audience for being at the meeting and apologising for having to leave immediately after his speech to attend another prior engagement.

His talk was entitled

> "The future of Formal Methods"

Professionalism

He presented the audience with a quotation:

> " ...these formal methods are the key to writing much better software. Their widespread use will revolutionise software writing, and the economic benefits will be considerable – on a par with those of the revolution in civil engineering during the last century."
>
> Brian Oakley
> "Alvey Achievements"
> June 1987

In the future we must be more precise. Only four years ago people involved with formal methods were regarded as "loonies". Now things are different; people apologise for *not* being formal at conferences. In the UK and Europe we have a world competitive edge in the area of formal methods.

Formal Methods are not a panacea

- We shall never be able to achieve certainty. There are good philosophical reasons for this.
- We cannot quantify the probability of error. There is no sign that we shall be able to do this before the end of the century.
- We must convince others by demonstration, not argument. The cost benefit must be shown. We need publishable evidence of improvements in quality and productivity. Please write up and publish such evidence – or send it to Martyn Thomas who will do this for you!

We need a technical intrastructure

- Tools – which should themselves be verified. These may be simple (type-checkers, cross reference generators, etc.) or complex (e.g. support for rigorous development). Why use formal methods for development and then use an unverified optimising compiler? We also need verified compilers.
- Standards – defining levels of use of formal methods, to be used in procurement.
- The infrastructure is important – we need a programme of developing key, verified tools and adopting them. If we don't act as if we feel formal methods are important, why should anyone else? We must use formal methods ourselves and formal verify software which we generate if we are to convince others of the benefit of the approach.
- We must maintain balance. Using formal methods without a QMS[1] is unbalanced. Using formal methods without conventional methods is unbalanced.

[1] Quality Management Standard.

Safety – a major issue

Safety-related computer systems are a major policy issue. Formal methods are important for safety – this is a useful pressure point on industry. If we cannot win the argument here, then we cannot win anywhere, particularly in the short term. The current state of play is lamentable; formal methods are almost unused for safety-critical systems apart from the CEGB[2] and for military applications.[3] Project managers are conservative – using unproven technology is avoided for good reasons. Even if formal methods were just used for specification then the rate of production of safety critical systems would be reduced drastically. In addition, formal methods are not politically acceptable yet.

We need:

- A definition of the acceptable minimum development practices;
- A definition of realistic best practice;
- Mandatory registration of safety-related systems and mandatory incident-reporting.

What can we realistically expect? Formal methods should be mandatory.

A long-term goal

We should aim to have formal methods in widespread use. We need a Trojan horse, for example, a formal SSADM, to bring semantic checking to a wider user base. There should be more information in the specification and more rule checking.

In the 1970s, strong typing and high level languages made right-first-time programming a realistic goal for many programmers. I would like us to provide the methods and tools for right-first-time specification and design, for the wider community, in the 1990s.

Questions

Q: There is a boot-strapping problem with verification. How many times do you need to do this to be sure? (For self-compiling compilers you can check for stabilisation of the binary code.)

A: This is a philosophical problem – but possibly not a practical problem.

Q: For RISC[4] machines, the verification problem has been moved from the hardware to the compilers. On USENET there has recently been a study of optimising compilers, all of which where found to be breakable. Any comments?

A: Many compilers are very complicated and bound to be bug-ridden. Programming in full Ada is an enormous act of faith which would stagger most monks! C is full of problems. However even assuming the compiler is correct, we still depend on the hardware. This is not necessarily reliable; for example, most PCs do not have memory parity checking.

[2] Central Electricity Generating Board.
[3] Rolls Royce are also using formal methods for safety critical systems.
[4] Reduced Instruction Set Computer.

2 Z standards, tools and education

2.1 Z standards *Rosalind Barden, Logica*

Rosalind Barden of the Advanced Software Engineering Group at Logica Cambridge Ltd gave this presentation instead of Tim Hoverd as previously advertised. The brief of this group is broadly to investigate and research areas which may be useful in 2–5 years' time. Rosalind described an IED[5] project proposal with the objective to produce a **Z** standard. This would use Mike Spivey's "The **Z** Notation: A Reference Manual" as a starting point.

ZEM – towards a method for the application of the Z notation

(or **Z** and the art of motorcycle maintenance!)

Overview

The partners are:

- Logica Cambridge Ltd
- IBM
- Programming Research Group

The proposal is for three year project at a total cost of £710,000.

Work Programme

It is felt that now is the time to strike for the standardisation of **Z**. The project aims to produce an evolving **Z** standard with regular reviews. It will use Mike Spivey's "The **Z** Notation: A Reference Manual" as input. A goal is to assist and extend the industrial use of **Z**.

The project would also be involved in the development of **Z**. It will undertake case studies in concurrency and the (formal) definition of concurrency extensions to **Z**. It will also investigate case studies in refinement and produce a refinement notation.

A methods hand-book will be written. This will be based on:

- A survey of experienced **Z** users,
- Case studies of ways of using **Z**.

Dissemination

The work will be in the public domain. There will be regular reviews of documents and a number of workshops will be held.

Volunteers will be required for

[5] Information Engineering Directorate – a successor to Alvey.

- The standards review body
- Case studies
- User survey

Please form an orderly queue!

In the question and answer session the following points were raised.

- It should be noted that this is only a *proposal* at the moment. It has passed the first stage and a full proposal has been submitted to the IED.
- This project might delay standardisation (being a three year project).
- There was a question as to what will happen if this project is not approved – will it happen anyway?

2.2 Tool requirements *Mike McMorran, IBM*

Mike McMorran of IBM, Winchester gave a talk on **Z** tools for the PS/2 personal computer. This work is in conjunction with Polytechnics. Around 20 people were surveyed to discover the user requirements for such tools. One requirement was to "stop changing the system" since minor changes in symbols causes problems with tools.

PS/2 tools

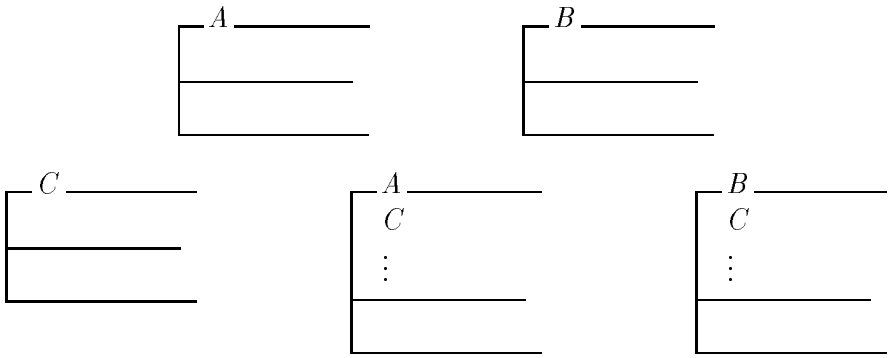
The basic PS/2 tools include:

- A live parsing editor "LPEX". This is a syntax driven editor which can handle C, REX, GML, etc. A parser for the GML mark up language with **Z** extensions was developed. In the future this may be extended to support the use of SGML. An investigation of the human factors involved in the entry of **Z** into a computer is seen as important. Entry aids to help with **Z** symbols and schema boxes are beneficial.
- Printing facilities. A number of special printer fonts are needed.
- Schema inclusion. This is a simple pragmatic tool for textual rather than mathematic manipulation. Area of interest include:
  - Schema windows.
  - Schema calculator. (Expanding schemas.)
  - Expression simplifier.
  - Consistency checker. (Avoiding *false* in predicates.)

Desirable tools

There are other areas which could benefit from mechanical tools. Some examples follow.

Aid with schema factoring is desirable. For example, given two schemas A and B, which parts can be factored out into a third schema C?



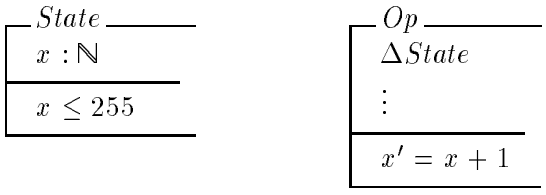
Checking for schema completeness is useful. For example in

$$Add \mathrel{\widehat=} AddOK \lor AddDep \lor AddError$$

are all situations covered by the three component schemas?

The calculation of pre- and post-conditions for an operation schema is useful. For example, the pre-condition may be calculated by hiding (existentially quantifying) all the output and *after* state components.

Help with proofs concerning pre- and post-operation predicates would be useful. E.g.:



Here of course $x' \leq 255$, so $x \leq 254$ is a pre-condition for Op to succeed. Such restrictions may not be so obvious in more complicated cases.

Design tools

A number of useful tools to aid the software design process have been identified:

- Refinement assistant, including refinement verification.
- Code sanity checker to identify code which *could* be a refinement and code which *cannot* be a refinement.
- Multi-parser, which will check the **Z** specification, the design notation and the code.

For successful design using formal methods, the whole project must be committed to their use. They are not effective on an individual basis.

Other tools

A number of other tools and facilities which could be helpful have been suggested:

- A tool to check for house style.
- A schema library.
- An informal text linker to relate the corresponding informal text with the formal text.
- A schema structure diagram generator.
- Measurement support.
- On-line reference manual (for example, Mike Spivey's "The **Z** Notation: A Reference Manual").
- Context sensitive help.

2.3 COMETT Z course *Jonathan Bowen, PRG*

Jonathan Bowen of the Programming Research Group described work currently being undertaken at the PRG on the distribution of material for a "COMETT"[6] **Z** course in the not-too-distant future. This is based on the existing **Z** course at Oxford, given at least every summer at an Oxford college, and also as demand dictates. It will include comprehensive printed material (using the document preparation system LaTeX[7] for which **Z** macros exist). The material will be designed for use by teachers of **Z** rather than end users. The idea is that teachers in industry and educational establishments should be able to obtain the material and use it as a basis for their own courses on **Z**.

Material

The material provided will consist of:

- Lecture notes – these are designed to be given to students on the course and covers what the teacher says during the lectures.
- Overhead projector foils – these will be based on the lecture notes.
- Teacher's notes – these include motivation for the lectures covered, points to be covered when presenting the foils and quick-fire questions which may be used to check that the students are following the material.
- Exercises and solutions.

[6] Although this has been nicknamed the "COMETT" course, I understand the funding is actually being provided by the UGC (University Grants Committee).

[7] This document has been prepared using the same text formatting system.

Sessions

The course consists of a number of $1\frac{1}{2}$ hour sessions (four per day is recommended). A typical breakdown would be:

Topic	Sessions
Motivation	1
Mathematics	4
Proofs	1
Structuring and use	5
Case studies	3
Exercises	3
Workshop	10

The workshop is optional, but it has been found to be very valuable in practice. The course participants are divided into teams of approximately 3–4 people and given a problem to specify in **Z**. Some refinement of the specification towards code may also be undertaken if desired, as time permits. In any case, it is recommended that anyone attending a **Z** course should attempt some sort of **Z** specification as soon as possible after the course to allow the ideas to be tried out in practice.

Material for each session is being prepared by members of the PRG and is reviewed by an least one other member of the PRG. The material has also been tested on the 1988 summer **Z** course at Oxford. All being well, the material will be made available sometime in 1989. Arrangements on how the material is to be distributed have not yet been finalised. It is intended that the material should be made available to anyone who wants it. The charge (as yet undecided) will be designed to cover costs rather than make a huge profit.

Details of the course will be sent to all **Z** Users Meeting participants as soon as it is available – if possible, please refrain from enquiries until this time!

Coffee break

3 Technical presentations

3.1 A Miscellany of Handy Techniques *Ruaridh Macdonald, RSRE*

Ruaridh Macdonald of RSRE,[8] Malvern, presented a number of techniques in **Z** which have been used at RSRE in the formal specification of part of a compiler. A paper covering the subject of his talk is included with these Proceedings.

3.2 Free Type Definitions *Mike Spivey, PRG*

The **Z** construct for "free type definitions" (sometimes called "data type definitions") provides a powerful way of describing recursive types. For example, here is a free type definition describing binary trees of numbers:

$$TREE ::= empty \mid fork\langle\!\langle \mathbb{N} \times TREE \times TREE \rangle\!\rangle.$$

[8] Royal Signals and Radar Establishment

The meaning of this definition can be explained by translating it into other **Z** constructs. We first introduce a new basic type $TREE$, the set of binary trees:

$$[TREE].$$

The empty tree is a constant of type $TREE$, and *fork* is a function taking a number and two trees and giving another tree:

$$\begin{array}{|l} empty : TREE \\ fork : \mathbb{N} \times TREE \times TREE \rightarrowtail TREE \end{array}$$

We use the injection arrow $\rightarrowtail$ to reflect the fact that *fork* makes different trees from different components.

Two axioms add more information about the type $TREE$. The first, a *disjointness* axiom, says that *empty* is not one of the trees that can be constructed with *fork*:

$$empty \notin \operatorname{ran} fork.$$

The second axiom is an *induction* principle:

$$\begin{array}{l} \forall S : \mathbb{P}\, TREE \bullet \\ \quad empty \in S \land \\ \quad fork(\!| \mathbb{N} \times S \times S |\!) \subseteq S \\ \quad \Rightarrow TREE \subseteq S. \end{array}$$

This axiom is the foundation for proof by structural induction on trees. To prove $\forall\, t : TREE \bullet P(t)$, we just need to show

1. $P(empty)$, and
2. if $P(t_1)$ and $P(t_2)$, then $P(fork(n, t_1, t_2))$.

To see the validity of this proof method, consider the set

$$S = \{\ t : TREE \mid P(t)\ \}.$$

Because of (1), *empty* is in S; and because of (2), if t_1 and t_2 are members of S and n is any natural number, then $fork(n, t_1, t_2)$ is in S: in symbols,

$$fork(\!| \mathbb{N} \times S \times S |\!) \subseteq S.$$

The induction axiom allows us to deduce that $TREE \subseteq S$, i.e. that

$$\forall\, t : TREE \bullet P(t).$$

Non-recursive free type definitions are often used to describe error codes:

$$RESULT ::= ok \mid none\_left,$$

or to give the effect of "variant records":

$$VEHICLE ::= car\langle\!\langle COLOUR \rangle\!\rangle \mid bike\langle\!\langle WEIGHT \rangle\!\rangle.$$

In these cases, the two axioms about the type can be merged into one *partitioning* axiom. For $RESULT$:

$$[RESULT]$$

$$ok, none\_left : RESULT$$
$$\langle \{ok\}, \{none\_left\} \rangle \text{ partition } RESULT$$

For $VEHICLE$:

$$[VEHICLE]$$

$$car : COLOUR \rightarrowtail VEHICLE$$
$$bike : WEIGHT \rightarrowtail VEHICLE$$
$$\langle \operatorname{ran} car, \operatorname{ran} bike \rangle \text{ partition } VEHICLE$$

The problem of consistency

A free type "definition" is no more than a *description* of a recursive type: there is no reason *a priori* to suppose that any recursive type exists which satisfies the description. To see that this is a potential source of inconsistency in specifications, consider the following description of the type of "objects":

$$OBJECT ::= file\langle\langle FILE \rangle\rangle \mid set\langle\langle \mathbb{P}\, OBJECT \rangle\rangle.$$

An object is either a simple file, or it is a packaged set of other objects.

Part of the meaning of this description is that set is an injection from $\mathbb{P}\, OBJECT$ to $OBJECT$:

$$set : \mathbb{P}\, OBJECT \rightarrowtail OBJECT.$$

But in fact, no such injection can exist! We can show this by assuming that an injection set exists and deriving a contradiction, namely a variant of Cantor's paradox in set theory.

Define a set C of objects by

$$C == \{\ V : \mathbb{P}\, OBJECT \mid set\ V \notin V \bullet set\ V\ \}.$$

For any set of objects V, we can ask whether the object $set\ V$ is a member of V itself. The set C contains those objects $set\ V$ for which the answer is "no". For any set $S : \mathbb{P}\, OBJECT$,

$$\begin{array}{lr}
set\ S \in C & \\
\quad \Leftrightarrow (\exists\, V : \mathbb{P}\, OBJECT \mid set\ V \notin V \bullet set\ S = set\ V) & [\text{def. of } C] \\
\quad \Leftrightarrow (\exists\, V : \mathbb{P}\, OBJECT \mid set\ V \notin V \bullet S = V) & [set \text{ is an injection}] \\
\quad \Leftrightarrow set\ S \notin S & [\text{one-point rule}]
\end{array}$$

So $set\ S \in C \Leftrightarrow set\ S \notin S$. Now substitute C for S: we obtain

$$set\ C \in C \Leftrightarrow set\ C \notin C,$$

a contradiction.

This contradiction does not mean that the **Z** construct for free type definitions is unsound; it simply highlights the responsibility of a specification author to check that any free type definitions he or she uses make sense, and underlines the need for general theorems which guarantee consistency.

Finitary constructions

In general, a free type definition looks like this:

$$
\begin{array}{rl}
T ::= & c_1 \mid c_2 \mid \ldots \mid c_m \\
\mid & d_1 \langle\!\langle E_1[T] \rangle\!\rangle \\
\mid & d_2 \langle\!\langle E_2[T] \rangle\!\rangle \\
\mid & \ldots \\
\mid & d_n \langle\!\langle E_n[T] \rangle\!\rangle
\end{array}
$$

Here c_1, c_2, ..., c_m are the constants of the type T, and d_1, d_2, ..., d_n are the constructors. The domains of the constructors are given by arbitrary set-valued expressions $E_1[T], E_2[T], \ldots, E_n[T]$ which may involve the type T being defined.

The consistency theorem guarantees that the recursive type described by this definition really exists by placing restrictions on the constructions $E_i[T]$ which may be used. In its simplest form, the theorem demands that all these constructions be *finitary*. A finitary construction E is one for which the result $E[S]$ of applying it to a set S is the same as the union of the sets $E[V]$ obtained by applying it to all *finite* subsets V of S. In symbols,

$$E[S] = \bigcup \{ \; V : \mathbb{F}\, S \bullet E[V] \; \}.$$

Broadly speaking, any construction of objects made from only a finite number of elements of T is finitary: for example,

- elements T,
- pairs $T \times T$,
- finite sequences seq T,
- finite sets $\mathbb{F}\, T$.

Moreover, any composition of finitary constructions is also finitary. But $\mathbb{P}\, T$ is not finitary, because it includes infinite subsets of T as well as finite ones. Here is the statement of the consistency theorem:

Theorem Any free type definition containing only finitary constructions has at least one model.

Summary

1. Free type definitions are an abbreviation for axiomatic descriptions of recursive types.
2. There is a need to prove that free type definitions are consistent.
3. By restricting ourselves to finitary constructions, we can be sure of consistency.

3.3 Brief introductions to demonstrations

During the lunch break a number of tools were demonstrated at the Programming Research Group in Keble Road.

4 Demonstrations

4.1 Fuzz *Mike Spivey, PRG*

*f*UZZ takes **Z** in the form of ASCII input which can also be processed by the LaTeX document preparation system to produce a formatted document suitable for output to a laser printer. The tool checks the conformance of the **Z** in the document by type-checking the contents. By default it uses the standard **Z** library as detailed in Mike's "The **Z** Notation: A Reference Manual". A listing showing schemas and types can be generated. Currently the tool runs on Sun 3 equipment under Unix and IBM PC compatible machines under MS-DOS. Other machines could be supported – the tool is written in C with portability in mind.

The tool is available on a commercial basis. For more information, contact Computing Science Consultancy, 2 Willow Close, Garsington, Oxford OX9 9AN.

4.2 FORSITE *Andy Ricketts, Racal*

This tool was jointly developed on the Alvey-funded FORSITE project by four sites[9] including Racal and the PRG. It is front-ended by the QED editor and includes facilities for the following:

- WYSIWYG[10] editing of **Z** documents.
- Parsing of **Z** documents.
- Type-checking of **Z** documents.
- Indexing of the schema and component names.
- Individual schema expansion.

The tool is written in C and the functional programming language ML. It runs under Unix on Sun 3 equipment under the SunView window system.

The FORSITE project is due to end at the end of March 1989. For more information on the availability of the FORSITE tool, contact Andy Ricketts, Racal Research Ltd., Worton Drive, Worton Grange Industrial Estate, Reading, BERKS RG2 0SB (Tel: 0734-868601).

4.3 Zebra *Bernard Sufrin, PRG*

This is another more recent **Z** type-checker. It is written in ML. It is less easy to port than Mike Spivey's tool and currently runs on Sun 3 equipment. It is normally interfaced to the WYSIWYG editor QED but could easily be adapted for other formats using conversion tables – internally the tool uses an ASCII form of the **Z** document. If anyone is intested in using this software, they should contact Bernard at the PRG – it is "more or less public domain".

[9] Pun intended I'm afraid!
[10] What You See Is What You Get.

4.4 B tool *Ib Sørensen, BP*

This proof assistant is being developed by Jean-Raymond Abrial in Paris in conjunction with the PRG. It is in use at the PRG and BP. The notation supported is similar to **Z** but does not include all the features of **Z**; in particular, schemas are not supported. This is a research tool and is still under development.

5 Project presentations

5.1 The use of Z in Tektronix *Kathleen Milsted, PRG*

Kathleen Milsted, currently a D.Phil. student at the PRG, spent July to October of 1988 working for Tektronix, Inc., Beaverton, Oregon, U.S.A.

TEKTRONIX, Inc.

Tektronix design, manufacture and sell high frequency oscilloscopes, computer peripherals, and other electronic equipment all around the world.

The Computer Research Laboratory have the following interests:

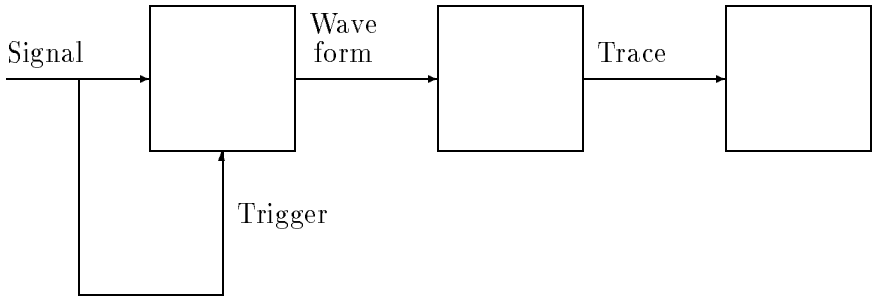
- Specification Environments
- Advanced Languages
- Knowledge Based Systems
- Visual Systems
- Application Languages: C, Smalltalk-80, C++, Scheme

Oscilloscope Design Project

Around half the people in the Tektronix Laboratory are involved in writing software, mainly in C. About a year ago they decided to try using **Z** (because of personal contact with the PRG) in the Specification Environments Group on an oscilloscope design project (one of their main products). Nowadays oscilloscopes contain more and more software and also multiprocessors.

The engineers were asked, "What is a waveform?" Their answer was often of the form "a 1K array of 8-bit digitised samples" – a rather implementation-oriented view!

The following block diagram gives an overview of an oscilloscope:



A signal can be considered as a (total) function of time (say nano-seconds) to voltage (say milli-Volts). A waveform can be thought of as some part of this – i.e. a partial function:

$$
\begin{array}{lcl}
Time & == & \mathbb{N} \\
Volts & == & \mathbb{Z}
\end{array}
$$

$$
\begin{array}{lcl}
Signal & == & Time \fun Volts \\
Waveform & == & Time \pfun Volts
\end{array}
$$

A waveform can be captured as a trace:

$$
\begin{array}{lcl}
Horiz & == & \mathbb{N} \\
Vert & == & \mathbb{Z}
\end{array}
$$

$$Trace == Horiz \pfun Vert$$

A trace can be triggered by user-specified events. There are a number of channel and trigger parameters to specify the desired configuration:

$$[ChanPrmtrs,\ TrigPrmtrs,\ Trigger]$$

$$TrigConfig : TrigPrmtrs \fun Signal \fun Trigger$$

$$ChanConfig : ChanPrmtrs \fun Trigger \fun Signal \fun Trace$$

An oscilloscope receives a signal. The channel and trigger parameters control the traces which are captured by the oscilloscope from the signal:

Oscilloscope

$$
\begin{array}{l}
s : Signal \\
cp : ChanPrmtrs \\
tp : TrigPrmtrs \\
ts : \mathrm{seq}\ Trace \\
\hline
\forall t : \mathrm{ran}\ ts \bullet \\
\quad \exists trig : TrigConfig\ tp\ s \bullet \\
\quad\quad t = ChanConfig\ cp\ trig\ s
\end{array}
$$

This modelling helps engineers to understand what an oscilloscope is in a more abstract way than they are used to.

ZEE – Z Engineering Environment

A support environment for the development of **Z** specifications called ZEE[11] is being developed. This has the following features:

- Implemented in Smalltalk,
- WYSIWYG editor,
- Parser,
- Type-checker,
- Hypertext capabilities,
- Proof environment (in design):
  - Simple schema manipulator,
  - Emphasis on support for user.

Reusable Component Catalogue

Work is also being undertaken on a reusable component catalogue based on a Smalltalk library. For more information, see: *Specifying reusable components using Z: Realistic Sets and Dictionaries*, by Ralph London and Kathleen Milsted. Specification styles for object-oriented applications are being investigated. Concerns include object–message orientation and single/multiple inheritance.

Overall, the management at Tektronix seem to be impressed by **Z**, so hopefully this experiment will be continued.

5.2 Working with CORE and Z: an evaluation *Brian Hepworth, British Aerospace*

Brian Hepworth of British Aerospace gave a talk at last year's **Z** Users Meeting. He summarised the progress in the use of **Z** at British Aerospace since then.

Software Technology Department – Formal Methods Group

The Software Technology Department consists of a number of specialist areas, undertaking research and development over a range of System and Software Engineering topics covering development of life-cycle tools and techniques through to performance evaluations of new hardware architecture, languages and concepts.

The Formal Methods Group undertakes research into methods for improving the specification and implementation of real-time software. Recent research has been targeted in the following areas:

[11] The name "ZEE" is an American answer to the insistence of pronouncing the name **Z** as "zed"!

Use of Z within CORE: CORE is a semi-formal requirements capture method and statement notation that is used extensively within British Aerospace on very large scale projects. Research into the use of **Z** applied at various levels within CORE has provided the basis for which formality will be introduced to BAe's current development process.

Project application of Z within CORE: Currently the Formal Methods Group is managing and providing technical support to a project representative pilot study using **Z** within CORE, in parallel with an on-line project. The objective of this study is to establish the benefits of applying formal techniques to the development process against those applied to the on-line and more conventional development process.

Prototyping Specifications: Many process-control applications specified in **Z** can be prototyped easily using functional programming languages. As an experiment, the language "Refine" has been used to produce prototypes of simple systems.

Tools: British Aerospace have developed the ZED editor which allows **Z** specifications to be created, modified and printed. Each user has access to the library of specifications which belong to the current project. Mathematical symbols are entered via keywords but displayed as symbols on the terminal.

Formal Methods Group Work

The research work concentrates on the following areas:

Utility Systems: Engines, Hydraulics, Fuel, etc. Case studies include:

- Hydraulics Control Systems
- Cabin Temperature Control Systems
- Engine Start, Restart, Shutdown Control Systems

Avionic Systems: Instrumentation, Navigation, Radar, Communications, etc. Case studies include:

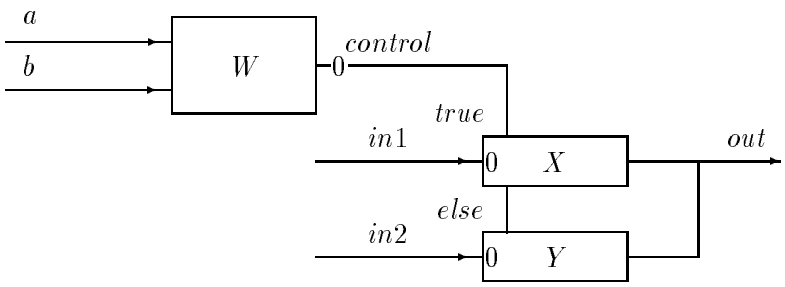
- Display Management Systems
- Navigation Database Systems

Flying Control Systems

A project on ground based test equipment (the AIDASS Hardware Diagnostics System) is also being undertaken to establish the benefits of using **Z**.

Case Studies

The initial aim in applying formal methods is to give a more definitive approach to writing process descriptions within CORE. Here is a simple example of a system described using **Z**:



W

$a? : GO\_STATE$
$b? : BOOLEAN$
$control! : BOOLEAN$

$a? = go \land b? = true$

$control! = true$

X

$in1? : \mathbb{N}$
$control? : BOOLEAN$
$out! : \mathbb{N}$

$control? = true$

$out! = 2 * in1?$

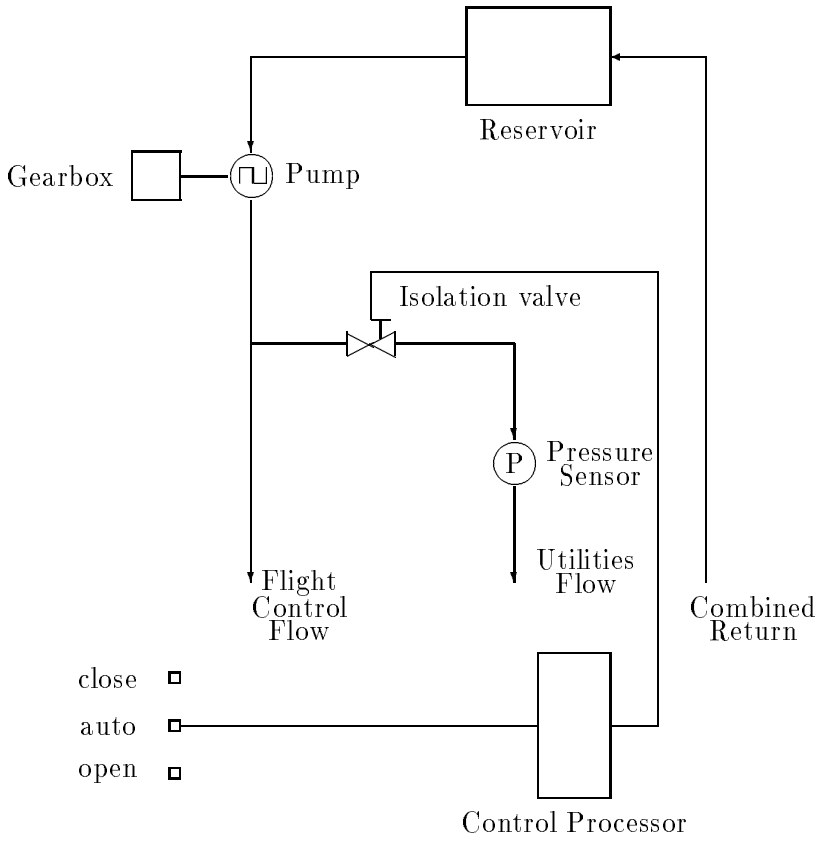
Y

$in2? : \mathbb{N}$
$control? : BOOLEAN$
$out! : \mathbb{N}$

$\neg\ (control? = true)$

$out! = 2 * in2?$

Here is another example of a control system taken from one of the case studies at British Aerospace:



SelectClosed

$valve\_select? : \{open,\ close,\ auto\}$

$valve\_control! : \{open\_vlv,\ close\_vlv\}$

$valve\_select? = close$

$valve\_control! = close\_vlv$

State & Abstract Data Representation

CORE has no defined representation of state and abstract data. However ad hoc development of CORE notations use the scheme where state data is input to and output from a process with its type defined informally within a data design note. Such problems are avoided when **Z** is used.

Future Work

British Aerospace plan to continue investigating the use of formal methods. The following work is planned:

- Formal program verification techniques and tools,
- Support for future **Z** tool developments,

- Refinement directed towards Ada code and subsets of Ada for safety-critical systems.

5.3 Formal definition of Information Systems *Ib Sørensen, BP*

Bernard Sufrin introduced Ib Sørensen as a former and future colleague – he is currently working at BP on leave from the PRG.

Objectives

Ib is currently interested in the software process and the associated quality concerns. The quality of specifications can be increased through the use of:

1. Mathematical based verification techniques,
2. Rapid prototyping.

The programmer's productivity can be increased through the use of:

1. Computer assisted design and code verification,
2. Reusable specifications, designs and algorithms.

In particular, the following objectives have been identified:

1. A notation and methods to be used in a computer aided formal development process should be developed – i.e. doing **Z** by computer.
2. Systems for the automatic generation of proof obligations are needed. It is useful to check the consistency of the specification, the correctness of the representation and the correctness of the algorithm used.
3. Systematic procedures for the semi automatic discharge of proof obligations should be developed. Libraries of mathematical laws and reusable proof-strategies are needed.
4. Systematic procedures for the semi automatic generation of designs, and code are required. Help with determining the weakest concrete design, a "Pascalizer" and automatic code generation are desirable.
5. A practical tool for 1–4 should be developed.

Overall Approach

The approach being used is along the following lines:

1. A mathematical basis is used – e.g. data refinement rules from the PRG and generalised substitution languages.
2. Experimental computer-based assistants are being investigated (if we know how to get the computer to do it – e.g. using the B tool, which is currently playing a central role[12]).

[12] The B tool is currently implemented in c8,000 of (subset) Pascal. BP intend to reimplement it formally using B.

3. Case Studies are being undertaken. e.g.:

   (a) Development of some simple linear access structures from specification to algorithm (e.g. stacks, queues, double-ended queues, etc.),

   (b) Development of medium scale information systems from specification to execution,

   (c) Specification of small real-time control systems.

   Currently dealing with concurrency can be a problem.

   As an example of the size of the case studies, a specification could consist of 50–100 predicates in total; each (expanded) operation uses around 10–20 predicates. Typically, around 5 operations can be verified in a morning using 10–20 step proofs. This gives you great confidence – you can go to lunch, feel happy, and not worry about it again.

In conclusion, the concern is what to do with a (**Z**) specification once it has been formulated. For practical development computer assistance is needed; in the real world it is not enough just to use pen and paper.

Break

5.4 Specifying a component of the SAA interface with Z *John Wordsworth, IBM*

John Wordsworth of IBM, Hursley Park, Winchester presented the problems encountered in specifying a part of the IBM SAA[13] interface in **Z**. In IBM, and SAA in particular, there are many TLA's[14] – in fact there are now so many that sometimes ETLA's[15] are needed!

SAA is designed to present a standard interface across different IBM machine architectures – e.g. CPIC.[16] A formal specification of this interface (largely in **Z**) has been produced. This is layered, so **Z** was particularly suitable since promoted operations could be used. At the bottom level there are around 30 operations on the "conversation state". These can be promoted to programs and then to nodes and new operations can be defined at each level.

All this can be done with "classical" **Z**. However there is a problem with a particular operation called "*confirm*". This was solved using a suggestion by Peter Lupton:[17]

$$
\begin{array}{l}
\mathit{confirm\_atomic} \\
[\!] \\
\mathit{confirm\_front\_end} \\
\quad \rightarrow \ \mathit{confirm\_back\_end}
\end{array}
$$

The pre-condition of *confirm_atomic* or *confirm_front_end* must be *true* for the operation to be performed. If both are *true*, either could be executed non-deterministically. If *confirm_atomic* is executed, the operation exhibits atomic behaviour. If *confirm_front_end* is executed then *confirm_back_end* is executed subsequently once its pre-condition becomes *true*. Other operations may interleave between the execution of *confirm_front_end* and *confirm_back_end*.

[13] Systems Application Architecture
[14] Three Letter Abbreviations
[15] Extended Three Letter Abbreviations!
[16] Common Programming Interface Communications
[17] Known as the "Luptonian" triad!

6 Panel on future directions

The day finished with a question and answer session by a number of **Z** experts from the PRG: Carroll Morgan, Mike Spivey, Bernard Sufrin and Jim Woodcock. The panel session ended around 5.15p.m.

7 General

7.1 Posters

Delegates were invited to display and view posters describing research related to the use of **Z** on the notice boards at the back of the auditorium. Copies of these are included with these Proceedings for delegates at the meeting.

7.2 Questionnaire

A questionnaire was circulated at the meeting to help in making improvements to future meetings. Delegates were asked to mark boxes for "usefulness", "right level", "right mix", "organisation" and "likelihood of reattendance" as 5 excellent, 4 good, 3 fair, 2 poor, or 1 bad for events which they attended. 44 completed forms were received.

Space was also provided for comments. The marks and comments were very variable. The average marks were all around 4. Things that some people did not like were often counterbalanced by people who did like them, so hopefully overall the balance was right.

Acknowledgements

Thank you to all the speakers for making the 1988 **Z** Users Meeting a worthwhile day for all the participants. In particular, thank you to those who provided foils, notes and even LaTeX source to aid the production of this record – Mike Spivey's account of his talk is included virtually verbatim.

Please accept the author's apologies for any mistakes in the transcription from verbal to written form. John Nicholls and others at the PRG gave valuable comments on the first draft of the Proceedings. A copy has also been circulated on the **Z** Forum electronic newsletter.

A special note of thanks must go to the organising secretary, Joanna Pulley, for making the day run so smoothly. Thank you too for the assistance of Joan Arnold in preparing these notes and mastering **Z** and diagrams in LaTeX.

Finally, thank you to the audience for making the meeting a success. We look forward to seeing you again next year – the date of the next **Z** Users Meeting will be Friday 15th December 1989.