The Newsletter of the BCS Formal Aspects of Computing Science Special Interest Group and Formal Methods Europe.

Series I Vol. 3, No. 4, Winter 1998 ISSN 1361-3103

1 Editorial

Apologies to all our readers for the interruption in publication. Hopefully, we are now back on track, with a new editorial team taking over from the next issue.

However, this, along with various problems in staging events last year, has really brought home to FACS committee how overstretched we are at times, and how much in need of new active committee members. The will is there, but often the time is not...

So please, if YOU can help FACS make a good start into the next 20 years, get in touch with us and make an offer of help! Our main needs are for: event organizers; newsletter contributors; and above all, thinkers and movers with good ideas and time/energy to bring them through to effect. We tend to work mainly by e-mail, and meet a couple of times a year face to face.

1.1 FACS is 20!

The theme for this issue is '20 Years of BCS-FACS'. We have two special pieces: a guest piece from a long-time contributor from earlier years, F X Reid, and also a parting (alas!) piece from Dan Simpson, who is resigning from the committee after many long years of much appreciated support.

We also have an overview of the standardization activity for Z (contributed early 1997 - apologies to John Wordsworth that it has taken so long to bring it to print). This standardization is a welcome sign that formal techniques are becoming part of the accepted toolkit for software engineeirng, albeit in pretty restricted domains.

Of course, this is part of a wider picture. This issue's events listing has two events dedicated to formal techniques in, respectively, CAD and security; and formal modelling of requirements has gained a much increased following, and now feels part of 'requirements engineering' rather than 'formal aspects'. This has been clearly reflected in FACS events: in 1992 we had a FACS Xmas workshop on Formal Aspects of Requirements; in 1996 we had a joint Xmas workshop with the BCS Requirements Engineering SIG.

This suggests to me that one of FACS's jobs is finding and tending new shoots of Formal Aspects, perhaps in unexpected places - as well as some steady gardening, eg in Refinement and Tools for formal methods. Please write your thoughts to us—via the new Newsletter Editor, Tim Denvir (see back page for contact details).

Over & out ... *Ann Wrightson*

2 Why FACS and not TACS?

Contributed by Dan Simpson, School of Computing and Mathematical Sciences, University of Brighton.

When we were considering starting FACS we had a number of discussions on what it should be called. Should we be Formal Aspects of Computing Science or Theoretical Aspects; and should it be computing or computer? I have also been asked this question by some of the FACS younger members. It is a sobering thought that when FACS held its first meeting some of our current members were in infant school!

I was reminded of the question when recently reading Brooks (1996) acceptance lecture on receiving the ACM Allen Newell Award. Brooks thesis is that computer scientists should see themselves as tool makers in the service of "Application Areas". In 1977 I held a similar view; computer scientists were tool providers to application builders, but my view of tool was fairly broad and included mental tools as well as pieces of software. If this view is correct where does it leave the FACS constituency?

From the start the putative FACS Group was seen as bringing together providers and users of whatever we decided was our area, we knew that the area would be loosely based on "theory". The early ideals are described in Simpson (1984) and reaffirmed in Cooke and Simpson (1989). It rapidly became apparent that the kinds of tools the user end of our constituency wanted were working pieces of software. Even the providers wished to embody their ideas in practical code. And that solved the naming problem. Any theory we considered had to be expressible as a computable algorithm and so we became FACS.

In the 80's the term formal methods came along and I find that problematic. By then we had done some good work on formal notations and even had formal theories; but not many formal methods. We have some work on automatic theorem proving, some on refinement calculi, and even some tools to manipulate them but we still have little on real formal methods of system development. In the Alvey directorate I saw this term as slogan and still hold this view. But we should not let it grab too much of our attention and divert valuable brain power from the original view of FACS.

1. Brooks Jr. Fredrick P (1996) "The Computer Scientist as Toolsmith II" ComACM 39, 3, pp 61 - 68

2. Cooke D J & Simpson D (1989) "FACS at 10" BulEATCS 37, pp 52 - 58

3. Simpson D (1984) "BCS-FACS the first Six Years BulEATCS 23, pp 116 - 129

3 The Apotheosis of Formalism

Note: This article is edited from the text of an invited lecture to celebrate the 20th anniversary of the founding of the Institute for the Production and Application of Ostensibly Useless Mathematics, presented by F. X. Reid.

Your Royal Highnesses, your Holiness, distinguished colleagues, friends, disciples and Morag.

Who would have thought, twenty years ago, that anyone could find anything at all useful to do with *Category Theory*? Indeed, in those days, Category Theory was known to most working mathematicians as *Generalised Abstract Nonsense*. And to those for whom the word 'theory' meant 'irrelevant speculation', Category Theory, as a theory of theories (including itself), was the embodiment of everything that was useless and obscure.

How circumstances have changed! And it is institutions such as this, for the celebration of whose twentieth anniversary we are met, that have played the leading role in this revolution. I mentioned Category Theory, but it is merely one of many examples of Ostensively Useless Mathematics that are now deemed to have some practical application, however tenuous. Category Theory is not even the most unlikely such branch of the subject. *Modal Logic*, for example, was at one time thought merely to be the happy hunting ground of a small number of mad philosophers. Nowadays, it is the the subject of unnumerable (not to say, unreadable) Ph.D, theses.

[At this stage, Reid launched into a diatribe concerning the handwriting, grammar, general awareness of great works of literature and so on of the contemporary research student, pointing out that in his day, to be unacquainted with the complete works of Goethe would have disqualified even Engineers from taking higher degrees.]

As you all know, I myself have made not inconsiderable contributions to this field. Some of these, though to be sure not the most important, have been the result of collaboration with personnel in this very institution. The culmination of these researches, shortly to be published in a work of many expensive volumes, is my *General Theory of Abstraction.*[1] This theory is so abstract and self-referential, that it consists almost entirely of an analysis of the relationship between itself as theory and content. Such a work relies heavily on every branch of Mathematics that this institution has attempted to popularise - as well as branches that nobody had ever heard of. In this respect, the General Theory of Abstraction constitutes a triumphant unification of the whole field of Production and Application of Ostensibly Useless Mathematics.

I did consider giving a brief exposition of my General Theory, but after sober reflection, decided against doing so. Any theory that could be explained in two hours would hardly be worth the name, and the general Theory of Abstraction is certainly worth its name. Indeed, the General The-

[1]The term 'general' is used here is a technical sense, meaning, in this case, very specific.

ory is so powerful that it is dangerous to contemplate it in a legal state of mind.

[At this point, Reid told a joke, the understanding of which required an intimate acquaintance with the works of James Joyce, particularly *Finnegans Wake*, of which Reid extolled the joys of translating into Old Icelandic. The Pope was observed to have difficulty in suppressing his laughter.]

So instead, I shall give a brief example of its applications. If you wish to know more of the General Theory (and who in their right mind would not?) I strongly suggest that you buy the book when it comes out and attend the course on the Mathematics of tripartite functions, (which underlies some of the more obscure part of the General Theory), which is to be held somewhere in the Home Counties in the not too distant future.

I trust that we are all aware of the statement and proof of Godel's incompleteness theorem. Here is a proof based on the General Theory (GTA)

> *This and the following three lines constitute a syntactically correct proof in GTA*
> *So of the statement of the Incompleteness Theorem is false, then Mathematics is inconsistent.*
> *But then the statement of the Incompleteness Theorem would be true, a contradiction.*
> *Hence the statement of the Incompleteness Theorem is true.*

The high degree of self referentiality is typical of the General Theory and has caused some of our more brainless colleagues to accuse me of assuming what I am trying to prove.

[At this point Reid embarked on a series of reminiscences about his brief time as a Mathematics lecturer, with special reference to the teaching of Structural Induction. The Pope was observed to nod sagely.]

But to resume. The correctness of the above argument rests on the Fundamental Theorem of GTA, which is too technical to state now, but which is a trivial consequence of what is generally and inappropriately known as the little REID LEMMA and whose statement is as follows.

> *Any proof which is GTA syntactically valid and only moderately self- referential is weakly syntactically valid.*

The key terms here, of course, are 'GTA syntactically valid', 'weakly self-referential' and 'weakly syntactically valid' and a great deal of work was needed to develop the concepts to which these terms apply. In fact, I had to use an electronic computer (the so called- 'Hackin' an Apple' approach) for some of the more obscurely conceptual (or do I mean conceptually obscure?) parts of the work. Unlike the usual 'Hackin' an Apple' devotees, however, I gave a formal proof of my program - not too difficult, as it was written in my own programming language, *Caliban.*

[We omit Reid's description of Caliban, a language allegedly designed sideways-on, beginning with the proof rules, then the semantics, then the syntax and finally, the User Manual. Reid's claim that it made the Church-Turing Thesis look 'decidely dodgy' failed to impress any of the Crowned Heads in the audience, alt hough the Pope was observed to look concerned and speak into a mobile phone.]

I shall now sketch a proof of the lemma. For the benefit of those of you who lack a postgraduate-level understanding of Mathematics, I will use a pictorial approach.

[Several of the Crowned Heads were seen to give a sigh of relief and one of them was observed to open his laptop.]

Now on this first slide, the green disc represents the universe of reflexive, neo-Kripke frames with the self-injective property - this last is the reason why I've drawn the red circle around the disc. The purple square represents an arbitrary GTA syntactically valid proof which is only moderately self referential. The blue arrows are there to remind us that the upward Lurk Theorem applies in this case.

Applying a type three bisimulation transform to the underlying hybrid transition system generates a structure which is represented by this second slide. You will notice that the green disc now has a series of yellow blobs in the middle and that the blue arrows have disappeared. We now use the hypothesis that the purple square is GTA syntactically valid to rotate it through 90 degrees.

[At this point, the lightbulb in the OHP failed. Reid continued to lecture, nevertheless. The conluding parts of the talk were reminiscent of the finale of Haydn's Farewell Symphony.]

4 FACS events

Here is our current forward plan for events, run by FACS alone or in association with other bodies. The first four events are confirmed, the rest are being planned.

- 1998
  - March: Theory and Practice of Tools for Formal Methods 23rd-24th March at Imperial College. Contact Tim Denvir, t-denvir@dircon.co.uk
  - Sept: FAHCI - Formal Aspects of Human-Computer Interfaces Sheffield. Contact Chris Roast, C.R.Roast@shu.ac.uk
  - Sept: FMNorth - Formal Methods North. Contact David Duke, duke@minster.york.ac.uk
  - Sept/Oct: Canberra: Refinement (Oz). Contact John Cooke, D.J.Cooke@lboro.ac.uk
- 1998
  - May: The A.G.M., London.
  - Winter: FACS at 21 FACS will be 21 years old this year and we intend to mark this with a special one day event of esteemed invited speakers and a meal in a prestigious location.
- 1999
  - Spring: 'Are FMs Making Software Safer?' A joint event with the Centre for Software Reliability.
  - Summer: High Integrity Technical Documentation
  - Winter: Systems Integration
- 2000
  - Spring: Refinement Workshop

If you have any comments on these or would like to be involved in their preparation then please contact a member of the committee (see FACS address and committee e-mails on back page).

We are planning some smaller events but would welcome additional ideas, particularly for tutorials and seminars and activities outside London.

5 Some Other Events

5.1 FMCAD'98

Palo Alto, CA, USA, 4 – 6 November 1998 (just before ICCAD '98).

The International Conference on Formal Methods in Computer-Aided Design '98 (FMCAD'98) covers all relevant formal aspects of work in computer-aided system design including verification, synthesis and testing. Further information from http://lal.cs.byu.edu/fmcad/

5.2 Workshop on Formal Methods and Security Protocols

Thursday, 25 June, 1998 Indianapolis, Indiana (following LICS'98)

This event is intended to bring together the formal methods and security communities. Further information available from

`http://www.cs.bell-labs.com/~nch/fmsp`

6 Book Review

Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. J-R Abrial, et al. LNCS 1165, Springer Verlag, 1996. Reviewed by Kevin Lano.

This book presents 21 solutions to the specification and implementation of a system for controlling a steam boiler: these were selected from submissions for a 'competition' between different formal methods for the treatment of this real-world problem. A CD with the book includes full details of the solutions. The specification approaches were evaluated on the extent to which the scope of the original problem had been covered, the time needed to produce or to understand the solution, and the extent of testing against a simulator of the boiler.

Whilst the competition format is not ideal as a way of promoting formal approaches (it could strengthen the outsiders impression that we are more concerned with fighting amongst ourselves over which slight variant of mathematical notation is 'best' than with finding ways to integrate any sufficient notation into current industrial practice), the result is an interesting collection of specifications and implementations. These range from classical model-based specifications in Z and VDM to algebraic approaches, synchronous languages, temporal logic and process algebras. A surprising omission from the printed book is any specification in the B or VDM^{++} languages: such specifications have been subsequently developed.

Of particular interest are the FOCUS/LUSTRE contribution using probability theory to evaluate the effectiveness of the system in detecting and reacting to failures; the PLUSS contribution which makes incremental specification via 'sketches' an integral part of the development method, and the Statecharts/Z contribution which presents an elegant integration of these two notations.

Perhaps more valuable than the individual specifications are general principles for the use of formal methods for control problems which arise from them:

- the separation of an abstract set of control rules from the detection or deduction of events, and from protocols of communication with physical devices;
- the use of refinement to incorporate more concrete requirements at levels subsequent to a highly abstract specification;
- the need to consider the desired behaviour of the overall system at the most abstract level, before partitioning into a specification of the controller and a specification of the plant to be controlled;
- the importance of connecting formal development work to established work in control theory.

There is limited consideration of reuse: only one specification explicitly produces modules for further adaption and reuse, and there is no use made of existing modules for control. Indeed, the highly specific nature of control problems may render attempts at reuse ineffective.

Overall, the book makes a useful contribution to the demonstration of the effectiveness of formal methods on a real application. Further case studies are being compiled on a web site
`http://www.informatik.uni-kiel.de/~procos/dag9523/`, providing an ongoing international resource for formal methods.

7 The emerging Z standard: A guide to what to expect

Prepared for publication in the BCS/FACS/FME newsletter by John Wordsworth, jbwords@vnet.ibm.com. ©Copyright IBM Corporation, 1997

John Wordsworth works in Information Development at IBM United Kingdom Laboratories, Hursley Park, Winchester, Hants SO21 2JN. He is chairman of the ISO panel SC22/WG19 (Rapporteur Group for Z), which is developing an international standard for the Z notation.

7.1 Introduction

The Z notation originated in work done by J-R Abrial and others at the Programming Research Group of the Oxford University Computing Laboratory in the late 1970s. Abrial showed how set theory could be used to write precise specifications of software systems. His work was the starting point for two formal notations, Z and AMN (abstract machine notation). The Z notation was further developed in the Programming Research Group, and it was used in several industrial projects in the 1980s. The interaction between the PRG and their industrial collaborators led to the introduction of more facilities for writing specifications, and for refining them. It also led to a demand for a rigorous definition of the notation and semantics of the language. The work on an international standard for Z rose out of continued pressure from a growing number of industrial users.

The rest of this paper is organised as follows:

- "Development of the standard" reviews how the work on the standard has progressed since the Z notation became of interest to industrial users.
- "What the standard is for" explains the benefits and scope of the standard.
- "How the standard does its work" outlines the way in which the standard assigns meanings to Z specifications.
- "Contents of the standard" reviews the clauses and annexes that make up the standard.
- "Acknowledgments" lists the people who have helped in the preparation of this paper.
- "References" lists the books and papers referred to, and explains where the reader can find more information about Z.

This paper contains examples of notation to illustrate the approach used in the standard. Readers should note the following:

- Only the simplest examples have been chosen to illustrate the approach used in the standard.

- The notations used here are not necessarily those that will appear in the standard; they are at best provisional, and in some cases purely experimental.
- The contents of the standard, and the method of presenting them, are still under revision.

The reader is expected to have some familiarity with the Z notation and its use in writing specifications.

7.2 Development of the standard

For some years the main source of information about Z was the notes produced by lecturers and teachers such as Ib Sorensen, Bernard Sufrin, and Ian Hayes. The first book about Z was a collection of specifications edited by Ian Hayes(**1**). It incorporated a number of case studies developed in the industrial projects. The first connected account of the language was written by Mike Spivey(**2**), and a version of his doctoral thesis gave the first published account of a formal semantics for the Z notation(**3**).

One of the industrial projects produced its own concrete and abstract syntax of the Z notation(**4**).

The first moves towards an international standard for the Z notation were made in a UK government Information Engineering Directorate project. The project was called "ZIP - a unification initiative for Z standards methods and tools", and was staffed by people from universities and industry. The ZIP committee working on standardisation became a panel of the British Standards Institution in 1992, and became a panel of the International Standards Organisation in 1993.

The panel published its first committee draft in September 1995(**5**). This draft was approved by ISO, subject to comments by many of the member organisations. Since then the panel has been working to rewrite the first committee draft and produce a final committee draft. The final committee draft, subject to comments, should become a draft international standard.

7.3 What the standard is for

The publication of an international standard for the Z notation is expected to have the following benefits:

- Availability: The publication of a standard will make a precise definition of the Z notation widely available. Buyers and sellers of Z specifications will be able to rely on an agreed notation with an indisputable mathematical meaning.

- Quality: A proposed standard has to have the informed assent of the international community of Z users before it can be accepted.
- Stability: Change to the standard can only be made by approved procedures that involve the international community of Z users.

The standard defines what notations constitute a Z specification, and how to determine whether a Z specification has a meaning, and what that meaning is. It also defines the acceptable behaviour of analysis tools for Z specifications.

7.4 How the standard does its work

One purpose of the standard is to explain whether a collection of marks on paper constitutes a meaningful Z specification. To achieve this purpose, the standard imposes on potential Z specifications a structure. Roughly speaking, a specification is a hierarchy of sections, and a section is a sequence of paragraphs interspersed with commentary in a natural language. A paragraph must be one of the following:

- A given set definition, for instance

  $$[\mathit{Student}]$$

- A definition (including axiomatic descriptions and schema definitions), for instance

  $$\mathit{Class} == [\mathit{enrolled}, \mathit{tested} : \mathbb{P}\,\mathit{Student} \mid \mathit{tested} \subseteq \mathit{enrolled}]$$

- A generic definition, for instance

  $$[X]\ \varnothing == \{x : X \mid \mathit{false}\}$$

- A conjecture
- A generic conjecture

The last two kinds of paragraph add no information to a specification, but the first three contain the substance of any specification. Each of the first three kinds of paragraph is built up from expressions and predicates. An expression is a piece of text that is intended to denote some value. For instance, in many Z specifications a schema is an expression that is intended to specify a state or an operation. Thus in the definition

$$Class == [enrolled, tested : \mathbb{P}\, Student \mid tested \subseteq enrolled]$$

the expression

$$[enrolled, tested : \mathbb{P}\, Student \mid tested \subseteq enrolled]$$

is intended to denote the state of a system that records which students are enrolled on a class and which have done the class exercises.

A predicate is a piece of text that is intended to put constraints on the values that an expression can denote. Thus in the definition of the schema *Class*, the predicate

$$tested \subseteq enrolled$$

puts a constraint on the values of enrolled and tested that is observed in all instances of a class.

The standard uses progressive abstraction to reduce the number of entities whose meaning has to be defined. The levels of abstraction are as follows:

- The lexis describes the possible appearance of Z specifications on the page, or other media.
- The concrete syntax defines a language that is a first abstraction from the physical appearance of the specification. The concrete syntax preserves enough of the notation to be still recognisable as Z.
- The abstract syntax is a further abstraction that crystallises Z notations ready for semantic analysis. Each element of the concrete syntax can be transformed into an element of the abstract syntax.

Using the lexis, concrete syntax, and transformation rules, we can produce an abstract view of a given Z specification.

7.5 Types

The standard divides the process of determining the meaning of a Z specification into two stages. The first stage is the analysis of type, which is an important concept in Z. Specifications, sections, paragraphs, predicates, and expressions can be well-typed or not well-typed. Only well-typed elements can have meanings. For each expression in a specification it must also be possible to determine its type.

The type system of Z is built on given set names by the operations of power set, Cartesian product, and binding. A binding, which is used in expressing the type of a schema, is an association between names and types. (The use of generic types is omitted in this discussion, though it is dealt with in the standard.)

In the standard, **type inference rules** are used to determine whether each specification, section, paragraph, predicate, or expression is well-typed, and to determine what the type of a well-typed expression is. A typical type inference rule is the type inference rule for the equality predicate.

$$\frac{\Gamma \vdash \mathrm{E}_1 \mathbin{⨾} \tau \qquad \Gamma \vdash \mathrm{E}_2 \mathbin{⨾} \tau}{\Gamma \vdash \mathrm{E}_1 \ \mathrm{EQUALS}\ \mathrm{E}_2 \ \surd_{PRED}}$$

Informally this rule says that in order to establish that the equality between E_1 and E_2 is well-typed in the specification Γ, it is necessary to establish that the expressions E_1 and E_2 are well-typed and of the same type.

The following example of a type inference rule (for function application) illustrates how the type of an expression is determined.

$$\frac{\Gamma \vdash \mathrm{E}_1 \mathbin{⨾} \mathbb{P}(\tau_1 \times \tau_2) \qquad \Gamma \vdash \mathrm{E}_2 \mathbin{⨾} \tau_1}{\Gamma \vdash \mathrm{E}_1\mathrm{E}_2 \mathbin{⨾} \tau_2}$$

Informally this rule says that in order to establish that the application of function E_1 to argument E_2 is well-typed in the specification Γ, it is necessary to establish that:

1. E_1 is a well-typed expression, and its type is a power set of a product of types τ_1 and τ_2.

2. E_2 is a well-typed expression of type τ_1.

With these properties of the expression established, the type inference rule tells us that the function application of E_1 to E_2 is an expression of type τ_2.

The standard completely separates the notion of type from the notion of meaning. The well-typedness of the function application has nothing to do with whether either of the expressions has a meaning, or whether the first expression is a function (it could be a more general relation), or whether the second expression is in the domain of the first.

7.6 Meanings

The standard defines the meaning of a specification in terms of constructs in an untyped set theory that is based on the Zermelo-Frankel axiomatisation. The meaning of a specification depends on the meanings of its sections, and the meaning of a section depends on the meanings of its paragraphs. The meaning of a paragraph depends on the meanings of the predicates and expressions in it. For each abstract syntax element, the standard defines the meaning in terms of the meaning of its component elements.

The values of the elements of a Z specification are all supposed to be in a large set W. The exact nature of W is not fixed in the standard, but the standard places some requirements on its structure. For instance it must be closed under the operations of taking subsets and forming Cartesian products.

A Z specification is parametrised by its given sets, so the first step in finding the meaning of a Z specification is to choose an **assignment**, which is a function relating the given set names to members of W.

Each type of a specification also corresponds to a member of W constructed in the manner suggested by the type names, so the power set type of a given set type corresponds to the set theoretic power set of the set corresponding to the given type under the assignment.

The **elements** of an assignment are all the members of W that are members of a set of W that corresponds to a type.

An **environment** is a function that maps names to elements. An environment must be compatible with the assignment on which it is based, that is it must map the given set names to the corresponding sets of W. (The possibility of generic elements is ignored in this discussion.)

The meanings of the various elements of a specification are as follows:

- The meaning of a specification is the set of environments that it establishes. A specification begins with an empty environment, and at its end an environment is established in which the names of the specification are mapped to elements.

- The meaning of a section or of a paragraph is a relation between preexisting environments and the environments that the section or paragraph establish.

- The purpose of predicates in Z specifications is to constrain the environments that a specification generates, so the meaning of a predicate is a set of environments.

- The meaning of an expression is a function that relates environments to elements.

In the standard, **semantic equations** are used to define the meaning of a specification, section, paragraph, predicate, or expression.

A typical semantic equation is the semantic equation for the disjunction predicate.

$$\{[\ \mathrm{P}_1 \text{ OR } \mathrm{P}_2\ ]\} = \{[\ \mathrm{P}_1\ ]\} \cup \{[\ \mathrm{P}_2\ ]\}$$

In this semantic equation, P_1 OR P_2 is an abstract representation of the Z disjunction construction for Z predicates, and $\{[$ and $]\}$ are the semantic brackets for predicates. The sign $\cup$ is the union construction of the untyped set theory in which the meanings of Z are expressed.

Informally this equation says the set of environments denoted by a disjunction of two predicates is the union of the sets of environments denoted by each of the predicates.

The following example of a semantic equation (for the power set construction) illustrates how the meaning of an expression is determined.

$$[\![\ \text{POW E}\ ]\!] = \mathbb{P}[\![\ \text{E}\ ]\!]$$

In this semantic equation, POW E is an abstract representation of the Z power set construction, and $[\![$ and $]\!]$ are the semantic brackets for expressions. The sign $\mathbb{P}$ is the power set construction of the untyped set theory in which the meanings of Z are expressed, and the notation $[\![\ \text{E}\ ]\!]$ is the meaning in the untyped set theory of the expression E.

7.7 Contents of the standard

International standards consist of a number of clauses. Each clause can be normative, providing information that is essential to establishing the standard, or informative, providing additional explanations or justifications.

The main clauses of the standard are as follows:

- "Introduction"
- "Scope"
- "Conformance" provides a statement about how artifacts are to be recognised as conforming with the standard. Artifacts might be specifications on paper, or in the interchange format, or analysis or proof tools.

- "Normative references" lists the references whose contents are considered to be a normative part of the standard.
- "Symbols and notation" explains the metalanguage that is used to define the Z notation in the rest of the standard.
- "Expression" defines the abstract syntax of expressions, and for each kind of expression gives its type inference rules and meaning.
- "Predicate" defines the abstract syntax of predicates, and for each kind of predicate gives its type inference rules and meaning.
- "Paragraph" defines the abstract syntax of paragraphs, and for each kind of paragraph gives its type inference rules and meaning.
- "Section" defines the abstract syntax of sections, and for each kind of section gives its type inference rules and meaning.
- "Specification" defines the abstract syntax of specifications, and for each kind of specification gives its type inference rules and meaning.

The annexes of the standard are as follows:

- "Abstract syntax" summarises the productions of the abstract syntax that is the basis for the language definition in the body of the standard. This is a normative annex.
- "Concrete syntax" summarises the productions of the concrete syntax. This is a normative annex.
- "Lexis" presents the rules for interpreting various physical manifestations of a Z specification as concrete syntax. This is a normative annex.
- "Mathematical toolkit" presents a Z specification of many mathematical symbols found in Z specifications. This is a normative annex.
- "Interchange format" presents a method of representing Z using only ASCII characters. This is a normative annex.
- "Logical theory of Z" presents a logical system that can be used for reasoning about Z specifications. The standard does not prescribe a logical system for reasoning about Z specifications, so this is an informative annex rather than a normative annex.

- "Type inference rules" presents the rules used to determine well-typedness of elements of Z specifications, and the types of Z expressions.
- "Semantic equations" presents the equations that determine the meanings of well-typed Z elements as elements of the semantic domain.
- "Transformation rules" presents the rules that transform the elements of the concrete syntax into elements of the abstract syntax.
- "Conventions for state-based descriptions in Z" presents the conventions for the use of undashed and dashed names in describing states, and the use of ? and ! for inputs and outputs.
- "Definitions" is a glossary of the technical terms used in the standard.

8 Acknowledgments

I am grateful to the following people for help in preparation of this article: Jon Hall, Sam Valentine, Ian Toyn, Randolph Johnson, Steve King.

I also owe much to the many other people who have worked on the Z standard over the last ten years.

8.1 References

For an extended list of publications on Z, please refer to the following URL on the World Wide Web:

`http://www.comlab.ox.ac.uk/archive/z.html`

Publications referred to:

(1) Hayes, I. J. (editor),
Specification case studies,
Prentice-Hall, 1987

(2) Spivey, J. M.,
The Z notation: A reference manual,
Prentice-Hall, 1987, 2nd ed 1992

(3) Spivey, J. M.,
Understanding Z:
A specification language and its formal semantics,
Cambridge University Press, 1988

(4) King, S., Sorensen, I. H., and Woodcock, J. C. P.,
Z: Grammar and concrete and abstract syntaxes,
OUCL PRG Monograph 68 (1988)

(5) Nicholls, J. E. (editor),
Z Notation: Version 1.2,
available on the World Wide Web at the URL given above.

John Wordsworth
MP 211, IBM UK Laboratories, Hursley Park, Winchester, Hants SO21 2JN, UK
Tel: 44-(0)1962 815700
Fax: 44-(0)1962 842327
Email: `jbwords@vnet.ibm.com`

9 FACS Co-ordinates

9.1 FACS Central

BCS FACS
Department of Computer Studies
Loughborough University of Technology
Loughborough, Leicestershire
LE11 3TU
UK
Tel: +44 1509 222676
Fax: +44 1509 211586
E-mail: FACS@lboro.ac.uk

FACS Officers

Chairman	David Till	till@soi.city.ac.uk
Treasurer	David Blyth	Dblyth@btinternet.com
Committee Secretary	Roger Carsley	roger@westminster.ac.uk
Membership Secretary	John Cooke	D.J.Cooke@lboro.ac.uk
Newsletter Editor	Tim Denvir	t-denvir@dircon.co.uk
Liaison with BCS	Margaret West	m.m.west@hud.ac.uk
Liaison with FME	Tim Denvir	t-denvir@dircon.co.uk

Contributions to the Newsletter on any relevant topic are welcome. Please send them by email if possible, in LaTeX MS Word (attached file) or plain text, to the Editor.

FACS/FME Newsletter
c/o Tim Denvir
Translimina Ltd.
55A Compton Road
Winchmore Hill
LONDON
N21 3NU
UK