

INTERNATIONAL  
STANDARD

**ISO/IEC**  
**13817-1**

First edition  
1996-12-15

---

---

**Information technology — Programming  
languages, their environments and system  
software interfaces — Vienna Development  
Method — Specification Language —**

**Part 1:**  
Base language

*Technologies de l'information — Langages de programmation, leurs  
environnements et interfaces logiciel système — Méthode de  
développement de Vienne — Langage de spécification —*

*Partie 1: Langage de base*



Reference number  
ISO/IEC 13817-1:1996(E)

## Contents

	Page
Foreword . . . . .	xi
Introduction . . . . .	xii
<b>1 Scope . . . . .</b>	<b>1</b>
<b>2 Normative References . . . . .</b>	<b>2</b>
<b>3 Definitions . . . . .</b>	<b>3</b>
3.1 Structure of Formal Definition . . . . .	4
3.2 Conventions . . . . .	4
3.2.1 Informative text . . . . .	4
<b>4 Conformity . . . . .</b>	<b>5</b>
4.1 Specifications . . . . .	5
<b>5 Basic Mathematical Notation . . . . .</b>	<b>6</b>
5.1 Logic Notation . . . . .	6
5.2 Basic Set Theory . . . . .	6
5.3 Cartesian Products . . . . .	7
5.4 Binary Relations and Functions . . . . .	8
5.5 Finite Sequences . . . . .	9
5.6 Finite Mappings . . . . .	10
5.7 Ordinal Numbers . . . . .	11
5.8 Definition by Transfinite Induction . . . . .	12
5.9 Cardinality and Cardinal Numbers . . . . .	12
5.10 Structured Expressions . . . . .	13
5.11 Semantic Function and Predicate Definitions . . . . .	15
5.12 Use of Recursion . . . . .	16
<b>6 Core Abstract Syntax . . . . .</b>	<b>17</b>
6.1 Document . . . . .	17
6.2 Definitions . . . . .	17
6.2.1 Type Definitions . . . . .	17
6.2.2 State Definition . . . . .	19
6.2.3 Value Definitions . . . . .	19
6.2.4 Function Definitions . . . . .	19
6.2.5 Operation Definitions . . . . .	20

© ISO/IEC 1996

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

6.3	Expressions . . . . .	21
6.3.1	Local Binding Expressions . . . . .	22
6.3.2	Conditional Expressions . . . . .	22
6.3.3	Unary Expressions . . . . .	22
6.3.4	Binary Expressions . . . . .	23
6.3.5	Quantified Expressions . . . . .	23
6.3.6	Iota Expression . . . . .	24
6.3.7	Set Expressions . . . . .	24
6.3.8	Sequence Expressions . . . . .	24
6.3.9	Map Expressions . . . . .	25
6.3.10	Tuple Constructor . . . . .	25
6.3.11	Record Expressions . . . . .	25
6.3.12	Apply Expressions . . . . .	25
6.3.13	Lambda Expression . . . . .	26
6.3.14	Is Expression . . . . .	26
6.3.15	Literals . . . . .	26
6.3.16	Identifiers . . . . .	27
6.4	State Designators . . . . .	28
6.5	Statements . . . . .	28
6.5.1	Local Binding Statements . . . . .	28
6.5.2	Block and Assignment Statements . . . . .	29
6.5.3	Conditional Statements . . . . .	29
6.5.4	Loop Statements . . . . .	29
6.5.5	Non-Deterministic Sequences . . . . .	30
6.5.6	Call and Return Statements . . . . .	30
6.5.7	Exception Handling Statements . . . . .	31
6.6	Patterns and Bindings . . . . .	31
7	Dynamic Semantic Domains . . . . .	34
7.1	The Domain Universe . . . . .	34
7.1.1	Basic Definitions . . . . .	34
7.1.1.1	Complete Partial Orders and Fixed Point Definitions . . . . .	34
7.1.1.2	Operators on Complete Partial Orders . . . . .	35
7.1.2	A Universe of Complete Partial Orders . . . . .	37
7.1.3	A Universe of Domains . . . . .	38
7.2	The Semantic Domains . . . . .	40
7.2.1	Basic Semantic Domains . . . . .	40
7.2.2	Extended Semantic Domains . . . . .	42
7.2.3	Semantic Domains for Evaluation Functions . . . . .	43
7.2.4	Auxiliary Semantic Domains . . . . .	45
8	The Dynamic Semantics . . . . .	46
8.1	Document . . . . .	46
8.2	Definitions . . . . .	48
8.2.1	Type Definitions . . . . .	48
8.2.1.1	The Verification Predicate for Types . . . . .	49
8.2.1.2	Evaluation Functions for Types . . . . .	50
8.2.2	State Definition . . . . .	57
8.2.3	Value Definitions . . . . .	57
8.2.3.1	The Verification Predicate for Value Definitions . . . . .	58
8.2.3.2	The Evaluation Function for Value Definitions . . . . .	61
8.2.4	Function Definitions . . . . .	63
8.2.4.1	Verification Predicates for Function Definitions . . . . .	63
8.2.4.2	Evaluation Functions for Polymorphic Functions . . . . .	67
8.2.5	Operation Definitions . . . . .	69
8.2.5.1	Verification Predicates for Operation Definitions . . . . .	69
8.2.5.2	Evaluation Functions for Operation Definitions . . . . .	70

8.3	Expressions . . . . .	72
8.3.1	Local Binding Expressions . . . . .	74
8.3.2	Conditional Expressions . . . . .	76
8.3.3	Unary Expressions . . . . .	78
8.3.3.1	Numeric Operations . . . . .	78
8.3.3.2	Logical Operation . . . . .	79
8.3.3.3	Set Operations . . . . .	79
8.3.3.4	Sequence Operations . . . . .	80
8.3.3.5	Map Operations . . . . .	81
8.3.4	Binary Expressions . . . . .	82
8.3.4.1	Numeric Operations . . . . .	82
8.3.4.2	Numeric Comparison Operators . . . . .	85
8.3.4.3	Equality Operations . . . . .	85
8.3.4.4	Logical Operations . . . . .	86
8.3.4.5	Logical Set Operations . . . . .	87
8.3.4.6	Set Operations . . . . .	88
8.3.4.7	Sequence Operation . . . . .	88
8.3.4.8	Sequence and Map Modification Operation . . . . .	89
8.3.4.9	Map Operations . . . . .	89
8.3.4.10	Compose and Iterate . . . . .	90
8.3.5	Quantified Expressions . . . . .	93
8.3.6	Iota Expression . . . . .	98
8.3.7	Set Expressions . . . . .	99
8.3.8	Sequence Expressions . . . . .	104
8.3.9	Map Expressions . . . . .	106
8.3.10	Tuple Constructor . . . . .	108
8.3.11	Record Expressions . . . . .	108
8.3.12	Apply Expressions . . . . .	110
8.3.13	Lambda Expression . . . . .	111
8.3.14	Is Expression . . . . .	112
8.4	State Designators . . . . .	113
8.5	Statements . . . . .	117
8.5.1	Underlying Theory For Statements . . . . .	117
8.5.2	The Statement Evaluation Functions . . . . .	119
8.5.3	Local Binding Statements . . . . .	120
8.5.4	Block and Assignment Statements . . . . .	122
8.5.5	Conditional Statements . . . . .	124
8.5.6	Loop Statements . . . . .	126
8.5.7	Non-Deterministic Sequences . . . . .	129
8.5.8	Call and Return Statements . . . . .	130
8.5.9	Exception Handling Statements . . . . .	131
8.6	Patterns and Bindings . . . . .	135
8.6.1	Patterns . . . . .	136
8.6.1.1	Auxiliary Functions . . . . .	141
8.6.2	Bindings . . . . .	144
8.7	Auxiliary Functions and Predicates . . . . .	144
8.7.1	Expansion Functions . . . . .	144
8.7.2	Functions for Extending the Environment . . . . .	147
8.7.3	Functions and Predicates to deal with the State . . . . .	151
8.7.4	Functions and Predicates to deal with Curried Functions . . . . .	153
8.7.5	Compute Functions . . . . .	154
8.7.6	Generate Functions . . . . .	156
8.7.7	Make Functions . . . . .	157
8.7.8	Get Functions . . . . .	159
8.7.9	Collector Functions . . . . .	163
8.7.10	Selector Functions . . . . .	166
8.7.11	Tag-processing Functions . . . . .	169

8.7.12	General Functions and Predicates	171
<b>9</b>	<b>The Mathematical Concrete Syntax</b>	<b>178</b>
9.1	Document	178
9.2	Definitions	178
9.2.1	Type Definitions	178
9.2.2	State Definition	179
9.2.3	Value Definitions	179
9.2.4	Function Definitions	179
9.2.5	Operation Definitions	179
9.3	Expressions	180
9.3.1	Bracketed Expressions	180
9.3.2	Local Binding Expressions	180
9.3.3	Conditional Expressions	180
9.3.4	Unary Expressions	181
9.3.5	Binary Expressions	181
9.3.6	Quantified Expressions	182
9.3.7	Iota Expression	183
9.3.8	Set Expressions	183
9.3.9	Sequence Expressions	183
9.3.10	Map Expressions	183
9.3.11	Tuple Constructor Expression	183
9.3.12	Record Expressions	183
9.3.13	Apply Expressions	183
9.3.14	Lambda Expression	183
9.3.15	Is Expressions	183
9.3.16	Names	183
9.4	State Designators	183
9.5	Statements	184
9.5.1	Local Binding Statements	184
9.5.2	Block and Assignment Statements	184
9.5.3	Conditional Statements	184
9.5.4	Loop Statements	184
9.5.5	Nondeterministic Statement	184
9.5.6	Call and Return Statements	185
9.5.7	Exception Handling Statements	185
9.5.8	Identity Statement	185
9.6	Patterns and Bindings	185
9.6.1	Patterns	185
9.6.2	Bindings	185
9.7	Lexical Specification	185
9.7.1	General	185
9.7.2	Characters	186
9.7.3	Symbols	188
9.8	Operator Precedence	188
9.8.1	The Family of Combinators	189
9.8.2	The Family of Applicators	189
9.8.3	The Family of Evaluators	189
9.8.4	The Family of Relations	190
9.8.5	The Family of Connectives	190
9.8.6	The Family of Constructors	190
9.8.7	Grouping	190
9.8.8	The Type Operators	191
<b>10</b>	<b>The Interchange Concrete Syntax</b>	<b>192</b>
10.1	Introduction	192
10.2	Lexis	192

10.3	Symbols	192
11	<b>The Outer Abstract Syntax</b>	196
11.1	Document	196
11.2	Definitions	196
11.2.1	Type Definitions	196
11.2.2	State Definition	197
11.2.3	Value Definitions	198
11.2.4	Function Definitions	198
11.2.5	Operation Definitions	198
11.3	Expressions	199
11.3.1	Bracketed Expressions	200
11.3.2	Local Binding Expressions	200
11.3.3	Conditional Expressions	200
11.3.4	Unary Expressions	201
11.3.5	Binary Expressions	201
11.3.6	Quantified Expressions	202
11.3.7	Iota Expression	202
11.3.8	Set Expressions	202
11.3.9	Sequence Expressions	203
11.3.10	Map Expressions	203
11.3.11	Tuple Constructor Expression	203
11.3.12	Record Expressions	203
11.3.13	Apply Expressions	203
11.3.14	Lambda Expression	204
11.3.15	Is Expressions	204
11.3.16	Names	204
11.4	State Designators	204
11.5	Statements	204
11.5.1	Local Binding Statements	205
11.5.2	Block and Assignment Statements	205
11.5.3	Conditional Statements	205
11.5.4	Loop Statements	206
11.5.5	Nondeterministic Statement	206
11.5.6	Call and Return Statements	206
11.5.7	Exception Handling Statements	206
11.5.8	Identity Statement	206
11.6	Patterns and Bindings	207
11.6.1	Patterns	207
11.6.2	Bindings	207
11.7	Lexical Specification	208
12	<b>The Syntax Mapping</b>	210
12.1	Structure and Style of the Definition	210
12.1.1	Division into Modules	210
12.1.1.1	Module "OAS2CAS"	210
12.1.1.2	Module "OAS"	210
12.1.1.3	Module "CAS"	210
12.1.1.4	Module "GetUnusedId"	210
12.1.2	Pre-conditions in the VDM-SL Definition of the Syntax Mapping	211
12.1.3	Transformation of a Document to CAS.Definitions	211
12.1.3.1	The Introduction of Additional Identifiers	211
12.1.3.2	The Generation of Quoted Pre- and Post-conditions for Functions	212
12.1.3.3	The Generation of Quoted Pre- and Post-conditions for Operations	212
12.1.3.4	The Transformation of Expressions	213
12.1.3.5	Pre-conditions of Operations	213
12.1.3.6	Guards of Error Handlers	214

12.1.3.7	Quoted Post-conditions of Implicit Operations	214
12.1.3.8	The Transformation of Type Definitions	214
12.1.3.9	The Transformation of Value Definitions	214
12.1.3.10	The Transformation of the State	214
12.1.4	Notational Conventions	215
12.2	Syntaxes and Auxiliary Functions	215
12.2.1	Module "OAS"	215
12.2.2	Module "CAS"	215
12.2.3	Module "GetUnusedId"	216
12.3	The Syntax Mapping Functions	216
12.3.1	Document	217
12.3.2	Definitions	220
12.3.2.1	Type Definitions	220
12.3.2.2	State Definition	224
12.3.2.3	Value Definitions	225
12.3.2.4	Function Definitions	225
12.3.2.5	Operation Definitions	232
12.3.3	Expressions	237
12.3.3.1	Bracketed Expressions	239
12.3.3.2	Local Binding Expressions	239
12.3.3.3	Conditional Expressions	240
12.3.3.4	Unary Expressions	241
12.3.3.5	Binary Expressions	241
12.3.3.6	Quantified Expressions	241
12.3.3.7	Iota Expression	241
12.3.3.8	Set Expressions	242
12.3.3.9	Sequence Expressions	242
12.3.3.10	Map Expressions	242
12.3.3.11	Tuple Constructor Expression	243
12.3.3.12	Record Expressions	243
12.3.3.13	Apply Expressions	244
12.3.3.14	Lambda Expression	244
12.3.3.15	Is Expressions	245
12.3.3.16	Names	245
12.3.4	State Designators	246
12.3.5	Statements	246
12.3.5.1	Local Binding Statements	247
12.3.5.2	Block and Assignment Statements	248
12.3.5.3	Conditional Statements	248
12.3.5.4	Loop Statements	249
12.3.5.5	NonDeterministic Statement	250
12.3.5.6	Call and Return Statements	250
12.3.5.7	Exception Handling Statements	250
12.3.5.8	Identity Statement	251
12.3.6	Patterns and Bindings	251
12.3.6.1	Patterns	251
12.3.6.2	Bindings	252
12.3.7	Lexical Specification	253
12.3.7.1	General	253
12.3.7.2	Characters	253
12.3.7.3	Symbols	253
13	The Static Semantic Domains	255
13.1	Type Representations	255
13.1.1	Special Subclasses of Type Representations	256
13.2	Environments	257
13.2.1	Accessing Environments	257

13.2.2	Updating Environments	258
13.3	Well-formedness Classifications	258
13.4	Type Relations	259
13.4.1	Subtypes	259
13.4.2	Overlapping Subtypes, Disjoint Types and Overlapping Types	264
13.4.3	Auxiliary Type Relations and Functions	267
13.5	Extended Abstract Syntax	273
14	The Static Semantics	273
14.1	Documents	274
14.1.1	Auxiliary Well-formedness Requirements	275
14.2	Definitions	276
14.2.1	Type and State Definitions	276
14.2.1.1	Types and Type maps	277
14.2.1.2	Extraction of Type Representations	279
14.2.2	Value, Function, and Operation Definitions	280
14.2.2.1	Simultaneous Definitions	280
14.2.2.2	Ordered Definition Groups and Definition Sequences	281
14.2.2.3	Definitions	282
14.2.2.4	Definition Groups and their Ordering	285
14.2.2.5	Ordering of Definitions	286
14.2.2.6	Asserted Visible Environments	287
14.2.2.7	Pre-Conditions	289
14.3	Expressions	289
14.3.1	Expression Characteristic Predicates	290
14.3.2	Relaxations and Restriction of Predicates	291
14.3.2.1	The Subsumption Rule	291
14.3.2.2	The InType Rule	291
14.3.2.3	Union Close	292
14.3.2.4	Sub-Environment Based Checks	292
14.3.3	Well-Formedness of Expressions	294
14.3.4	Bracketed Expression	295
14.3.5	Local Binding Expressions	295
14.3.5.1	Let Expression	295
14.3.5.2	Let Be ST Expression	295
14.3.5.3	Def Expression	296
14.3.6	Conditional Expressions	296
14.3.6.1	If Expression	296
14.3.6.2	Cases Expression	297
14.3.7	Unary Expressions	298
14.3.7.1	Numeric Operations	299
14.3.7.2	Logical Operation	300
14.3.7.3	Set Operations	301
14.3.7.4	Sequence Operations	302
14.3.7.5	Map Operations	303
14.3.7.6	Map Inverse Expression	304
14.3.8	Binary Expressions	304
14.3.8.1	Numeric Operations	305
14.3.8.2	Numeric Comparison Operators	307
14.3.8.3	Equality operations	308
14.3.8.4	Logical Operations	308
14.3.8.5	Logical Set Operations	310
14.3.8.6	Set Operations	311
14.3.8.7	Sequence Operation	311
14.3.8.8	Sequence and Map Modification Operation	312
14.3.8.9	Map Operations	313
14.3.8.10	Compose and Iterate	315

14.3.9	Quantified Expressions	316
14.3.10	Iota Expression	317
14.3.11	Set Expressions	318
14.3.12	Sequence Expressions	319
14.3.13	MapExpressions	320
14.3.14	Tuple Constructor	321
14.3.15	Record Expressions	322
14.3.16	Apply Expressions	323
14.3.17	Lambda Expressions	324
14.3.18	Is Expressions	326
14.3.19	Names	326
14.3.20	Literals	326
14.3.21	Auxiliary Well-formedness Predicates	327
14.4	State Designators	327
14.5	Statements	328
14.5.1	Local Binding Statements	329
14.5.2	Block and Assignment Statements	330
14.5.3	Conditional Statements	332
14.5.4	Loop Statements	333
14.5.5	Non-Deterministic Statement	334
14.5.6	Call and Return Statements	335
14.5.7	Exception Handling Statements	335
14.5.8	Identity Statements	337
14.6	Patterns and Bindings	337
14.6.1	Patterns	338
14.6.1.1	Pattern Characteristic Predicates	338
14.6.1.2	Relaxation of Pattern Characteristic Predicates	340
14.6.1.3	Well-formedness of Patterns	340
14.6.2	Bindings	343
14.6.3	Value Environment	346
14.7	Auxiliary Functions	347
14.7.1	Dependency Relations	347
14.7.1.1	Free and Defined Names	348
14.7.1.2	Syntactical Sub-Components	356
14.7.1.3	The Definitional Basis of Types	361
14.7.2	Syntax Transformations	361
14.7.3	Substitutions	363
14.7.4	Indirectly Defined Functions	368
14.7.5	Classification Functions	371
A	Extensions	372
B	Tool Conformity	373
B.1	Semantic conformity of tools	373
C	Modules	375
C.1	Overview	375
C.2	Requirements of Modularization	375
C.2.1	Language Facilities	375
C.2.1.0.1	Syntactic Separation	375
C.2.1.0.2	Explicit Import	375
C.2.1.0.3	Explicit Export	375
C.2.1.0.4	Parameterization	376
C.2.2	Semantics	376
C.3	Extant Approaches to Structuring Specifications	376
C.3.1	The Syntactic Approach	376
C.3.2	The Z Approach	376

C.3.3	The Manchester Approach (Fitzgerald & Jones) .....	376
C.3.4	The VVSL Approach (Middelburg) .....	377
C.3.5	The RAISE Approach .....	377
D	<b>Cross-references</b> .....	379
D.1	Cross References for the Dynamic Semantics .....	379
D.1.1	Naming and Typesetting Conventions Used .....	379
D.1.2	Listing of Functions/Predicates: Alphabetic (uses) .....	380
D.2	Cross References for the Concrete Syntax .....	385
D.3	Cross References for the Abstract Syntax .....	388
D.4	Cross References for the Syntax Mapping .....	390
D.5	Cross References for the Static Semantics .....	393
E	<b>Bibliography</b> .....	397

## Figures

1	Structure of the syntax mapping .....	210
---	---------------------------------------	-----

## Tables

1	Character set .....	187
2	Interchange syntax: representation of symbols .....	192

## Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 13817-1 was prepared by Joint Technical Committee, ISO/IEC JTC 1, *Information technology, Subcommittee SC 22, Programming languages, their environments and system software interfaces*.

ISO/IEC 13817 consists of the following parts, under the general title *Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language*:

— *Part 1: Base language*

Additional parts will specify modules and the development method.

Annexes A to E of this part of ISO/IEC 13817 are for information only.

## Introduction

### Historical background of the Vienna Development Method (VDM)

VDM was developed at the IBM Laboratory in Vienna. The Laboratory came into existence in 1961 when Professor Heinz Zemanek of the Technical University in Vienna decided to move his whole group to an industrial home.<sup>1)</sup> They had previously developed a computer called Mailüfterl at the Technical University. From 1958 the group had been increasingly involved in software projects including the construction of one of the early compilers for the ALGOL 60 programming language. As time went on they found it difficult to get adequate support for their projects and eventually joined IBM. In the mid-1960s, IBM decided to develop a new programming language for which the ambition was to replace both FORTRAN and COBOL. The language, which was at first called New Programming Language (until the National Physical Laboratories in the UK objected to the acronym — the language became known as PL/I), was clearly going to be large and it was decided that it would be useful to try to apply formal techniques to its description.

Based on their own work — and influenced by research work by Cal Elgot, Peter Landin and John McCarthy — the Vienna group developed an operational semantics definition of PL/I which they called ULD-3 ('Universal Language Description'; ULD-2 was the name applied to the IBM Hursley contribution to this effort — the language itself was being developed mainly from Hursley along with the early compilers; ULD-1 was a term applied to the natural language description of the language). The description of PL/I in ULD-3 style ran through three versions. These are very large documents. Operational semantics is now seen as unnecessarily complicated as compared to denotational semantics. However, to make the principles of denotational semantics applicable to a language like PL/I with arbitrary transfer of control, procedures as arguments, complicated tasking, etc. required major theoretical break-throughs and a considerable mathematical apparatus not available at the time. The effort of the formal definition uncovered many language problems early and had a substantial influence on the shape of the language.

Towards the end of the 1960s serious attempts were made to use the ULD-3 description as the basis of compiler designs. Many problems were uncovered: in general, one could say that the over-detailed mechanistic features of the operational semantics definition considerably complicated the task of proving that compiling algorithms were correct. But again one should be clear that the work was a technical achievement: a series of papers was published that described how various programming language concepts could be mapped into implementations which could be proved correct from the description (e.g. in JL71). In addition, a series of proposals was made which could simplify the task of developing compilers from a semantic description. One of these was an early form of an exit construct (HJ70) which led to an interesting difference between the Vienna flavour of denotational semantics and that used in Oxford. Another VDM-like idea that arose at this time was Peter Lucas' *twin machine* proof (Luc68), and subsequently the observation that the ghost variable treatment in the twin machine could be replaced by retrieve functions (Jon70) as a simpler way of proving most cases of data development are correct. It is worth noting that Lucas' *twin machine* idea has been re-invented several times since: the generalization of retrieve functions to relations can be seen as equivalent to twin machines with invariants.

Hans Beckič spent some time in England with Peter Landin at Queen Mary College and was the person who first pushed the Vienna group in the direction of denotational semantics. (Until his untimely death in 1982, Hans Beckič

---

<sup>1)</sup>This is not intended to be a history of the Vienna Laboratory: citations are limited to those concerning VDM itself.

had published relatively little of his scientific research; some of his **important** papers were published posthumously in Bek84.) Another crucial stimulus was the visit to the Vienna laboratory by Dana Scott in 1969 (see dBS69).

During the period from 1971 to 1973, the Vienna group was diverted **into** other activities not really related to formal description. Cliff Jones at this time went back to the Hursley Laboratory and worked on a *functional* language description (ACJ72) and other aspects of what has become known as VDM. In particular he published a development of Earley's recogniser (Jon72) which is one of the first reports to use **data** reification. In late 1972 and throughout '73 and '74 the Vienna group (Cliff Jones returned and Dines Bjørner was recruited) had the opportunity to work on a PL/I compiler for what was then a very novel machine architecture. **They** of course decided to base their development for the compiler on a formal description of the programming language. PL/I was then undergoing ECMA/ANSI standardization (ANS76). The Vienna group chose to write a denotational semantics for PL/I (BBH<sup>+</sup>74); this is the origin of the VDM<sup>1</sup> work on language description techniques.

During the same period, at the IBM Laboratory in Hursley, an investigation into the use of Meta-IV to formally describe five of the major languages supported by IBM was carried out. The languages were PL/I, BASIC, FORTRAN, APL and COBOL. Sketches for parts of FORTRAN and APL were written, and a full description of minimal BASIC was produced. This work ceased when the language work was moved from Hursley.

Cliff Jones and Dines Bjørner took upon themselves the task of **making** sure that something other than technical reports existed to describe the work that had gone on on the language aspects of VDM: Be78 is a first book-length description of that work. In ESRI, Cliff Jones also developed the work on those aspects of VDM not specifically related to compiler development and the first book on what is now generally thought of as VDM is Jon80. Both of these books have now been superceded: the language description work is best accessed in Be82 and — in its second edition — the non-language work is best seen in Jon90 and also in AI91.

Within IBM, from 1978, a number of projects used VDM (for other than language descriptions). It was during this period that specifications of several large systems were carried out at the Böblingen Laboratory. (These included a file system for DOS and an fault report tracking system; a data reification for the file system was attempted to show that the proof techniques were viable in an industrial environment.) This work was carried out as part of a technology transfer program. Later the IBM Program Product Development Centre in Rome became involved, and work was done to formally specify a hotel management system. This work showed that VDM was suitable for large-scale projects; unfortunately little has been published on this experience.

Dines Bjørner's group at the Technical University of Denmark strenuously pursued the use of VDM for language description and he and his colleagues were responsible for descriptions of the CHILL programming Language and a major effort to document the semantics of the Ada programming language (BO80).

Language work was also continued at Leicester University where a formal definition of the full Pascal language was written (AH82) and later a formal definition of the programming language Modula-2, which became a Draft International Standard (AeWO).

The non-language, specification, aspects of VDM were taken up by the STL laboratory in Harlow and, partly because of their industrial push, the British Standards Institute (BSI) was persuaded to establish a standardization activity. This activity has not been easy because of the differences between the pressures of those interested in the language description aspects of VDM and those who are more interested in pre- and post-conditions, data reification and operation decomposition. It is to the credit of both the BSI and ISO Standards committee that they have managed to bear in mind the requirements of both types of user and come up with a standard that embraces such a wide scope of technical ideas. STL was responsible for funding the first formal semantics of VDM-SL and the work that was done at Manchester University by Brian Monahan (Mon87) was used as the starting point of the formal description of the specification language. A new formal description based on this work was produced because of the necessity of merging together the two aspects of the specification language.

The outcome of the standardization effort initiated by STL through the *British Standards Institution (BSI)* was the formation of a panel (BSI IST/5/-/50) whose membership was also open to participants from non-British organizations.

<sup>1</sup>)It is worth getting some acronyms sorted out: VDL stands for Vienna Description Language and was a term used to describe the operational semantics (ULD-3) notation; VDM stands for Vienna Development Method and relates to the post-1973 work; the specification language part of VDM is sometimes known as 'Meta-IV'; it is now known as VDM-SL.

## ISO/IEC 13817-1 : 1996(E)

In 1991 the need for a VDM-SL standard was also recognized by *ISO/IEC JTC1*<sup>2)</sup> by the formation of a Working Group, SC22/WG19. The actual work on the standard is done by the members of the BSI Panel.

Work on the Standard encouraged the building of computer-based tools to support both the language and the development method. Adelard produced 'SpecBox', a tool that provided syntax and type checking of VDM specifications. Manchester University and the Rutherford Appleton Laboratories wrote 'mural' (JJLM91), a prototype tool that supports both the specification language and the development method, providing a proof engine to help with data reification and operation decomposition. The Technical University of Denmark and the National Physical Laboratory have configured the Cornell Synthesizer (a structured editor tool) to support VDM-SL with some type-checking facilities — this tool was used to produce the static semantics of VDM-SL. IFAD have produced a tool for syntax and type checking together with an interpreter for an executable subset to allow rapid prototyping.

During its history, the Vienna Development Method, together with its specification languages has had a profound influence on both the specification of programming languages and the specification and development of systems. The ideas in VDM have influenced several other specification languages including RAISE, COLD-K and VVSL.

---

<sup>2)</sup>Joint Technical Committee 1 of the International Standards Organization and the International Electro-technical Commission.

---

# Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language —

## Part 1: Base language

### 1 Scope

This part of ISO/IEC 13817 specifies the model based specification language VDM-SL (Vienna Development Method — Specification Language). It specifies:

- two representations: the mathematical and interchange;
- the syntax;
- the static semantics;
- the dynamic semantics;
- conformity for specifications and tools.

It does not specify:

- the proof obligations;
- the reification rules;
- the size or complexity of a specification that will exceed the capacity of any specific data processing system or the capacity of a particular tool, nor the actions to be taken when the corresponding limits are exceeded;
- the minimal requirements of a data processing system that is capable of supporting an implementation of a tool;
- the method that tools use for reporting errors.

## 2 Normative References

The following standards contain provisions which, through reference in this text, constitute requirements of this part of ISO/IEC 13817. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 13817 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 8859-1:1987<sup>1)</sup>, *Information processing — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1.*

ISO/IEC 14977:1996, *Syntactic Metalanguage — Extended BNF.*

---

1) Currently under revision.