



ATLANTIS:  
A TOOL FOR LANGUAGE DEFINITION  
AND INTERPRETER SYNTHESIS

Michael John Oudshoorn, B.Sc.(Hons.)

A THESIS SUBMITTED FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
IN THE DEPARTMENT OF COMPUTER SCIENCE  
UNIVERSITY OF ADELAIDE

August 1992

# Abstract

Programming language semantics are usually defined informally in some form of technical natural language, or in a very mathematical manner with techniques such as the Vienna Definition Method (VDM) or denotational semantics. One difficulty which arises from serious attempts to define language semantics is that the resulting definition is generally suitable for a single limited kind of reader. For example, the more formal kind of definition may suit a compiler writer or a language designer, but will be less convenient for other potential classes of reader, such as programmers. The latter frequently make use of some completely separate description (e.g., an introductory text book on the language); not surprisingly, inconsistencies between these separate descriptions and the language definition are commonplace.

This thesis develops a technique for the definition of programming language semantics which is suitable for a wide range of potential readers. This technique employs an operational semantic model which is based on the algebraic specification of abstract data types; the semantic model manipulates multi-layer descriptions of language semantics and supports multiple passes in these descriptions.

The semantic technique described in this thesis lends itself to the semi-automatic generation of an interpreter from the language definition, a fact which acts as an

incentive to language designers to produce a formal definition of any new programming language, since the prototype implementation allows experimentation with new language features and their semantics. The system which generates an interpretive implementation from a language definition is called ATLANTIS, A Tool for LANguage definiTion and Interpreter Synthesis, and is also described in this thesis.

# Declaration

This is to certify that this thesis contains no material which has previously been accepted for the award of any degree or diploma in any University. To the best of my knowledge and belief it contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

If this thesis is accepted for the award of the degree, permission is granted for it to be made available for loan and photocopying.

Michael Oudshoorn

August 1992

# Acknowledgments

I wish to acknowledge all the help and encouragement that I have received throughout my candidature as a Ph.D. student. In particular, I am indebted to my supervisor, Prof. Chris Marlin, for his advice, thoughts, guidance and encouragement; and to Dr. David Bover who acted as my supervisor whilst Prof. Marlin was on study leave during the second half of 1988 and early 1989. I also wish to express my thanks to Prof. Peter King and Dr. Brian Molinari for their comments on this thesis. I am also indebted to my wife, Jo, and my son, Daniel, for having to tolerate many late nights, and often having to come second behind my study. Many thanks to my parents, Bep and Nick, who supported and encouraged my study.

I also wish to thank the other members of the staff, the postgraduate students and visiting speakers, who have all contributed to this thesis in many small ways. Of particular note in this regard are Michael McCarthy and Ali Mili. I also wish to acknowledge the contribution of Keith Ransom, who developed the Ada code in Section 5.5 and carried out the experiments referred to in Section 5.7.

I would also like to thank all those companies that allowed me time on their machines and gave me access to their Pascal compilers to obtain the results discussed in Chapter 1; I have been asked by many of these companies not to publish their names.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Declaration</b>	<b>iv</b>
<b>Acknowledgments</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Descriptions of Syntax . . . . .	2
1.2 Descriptions of Semantics . . . . .	3
1.2.1 The Informal Approach . . . . .	3
1.2.1.1 Historical Changes to the Pascal Definition . . . . .	6
1.2.1.2 Problems within the Current Pascal Definition . . . . .	8
1.2.2 Formal Definitions . . . . .	14
1.3 The Model Presented in this Thesis . . . . .	18
<b>2 Algebraic Specification of Abstract Data Types</b>	<b>22</b>
2.1 An Example of an Abstract Data Type . . . . .	23
2.2 Axiomatizations . . . . .	24
2.3 Error Conditions . . . . .	31
2.4 Verifying the Specification . . . . .	39

2.4.1	Consistency . . . . .	39
2.4.2	Sufficient-Completeness . . . . .	44
2.4.3	An Alternative Approach . . . . .	47
2.5	The Use of Algebraic Specification Techniques in this Thesis . . . . .	49
<b>3</b>	<b>Describing Sequential Languages</b>	<b>51</b>
3.1	Introduction . . . . .	51
3.2	The Structure of the Multi Layered Model . . . . .	53
3.3	A Model of Data Control . . . . .	58
3.3.1	Embedding Semantics Within a Syntactic Description . . . . .	59
3.3.2	Abstract Data Type Definitions – the Formal Foundation . . . . .	62
3.3.3	The Information Structures and Their Operations . . . . .	63
3.3.4	High Level Operations . . . . .	70
3.3.5	The Semantic Descriptions . . . . .	73
3.3.5.1	Local Declarations . . . . .	73
3.3.5.2	Scope Rules . . . . .	76
3.3.5.3	Parameters . . . . .	80
3.4	Observations . . . . .	87
<b>4</b>	<b>A Tool for Language Definition and Interpreter Synthesis</b>	<b>89</b>
4.1	Introduction . . . . .	89
4.2	General . . . . .	92
4.3	Lexical Analysis . . . . .	93
4.4	The Syntactic Component . . . . .	101
4.5	The Semantic Component . . . . .	103
4.6	Abstract Data Type Definitions . . . . .	108

4.7	Environment Variables . . . . .	116
4.8	High Level Operations . . . . .	118
4.9	Predefined Abstract Data Types . . . . .	124
4.10	Implementation of ATLANTIS . . . . .	125
4.11	Observations . . . . .	133
4.12	Difficulties . . . . .	138
<b>5</b>	<b>Describing Parallel Languages</b>	<b>139</b>
5.1	Introduction . . . . .	139
5.2	Mallgren's Approach to Shared Data Abstractions . . . . .	140
5.3	The Ada Rendezvous . . . . .	145
5.4	A Shared Data Abstraction Model for the Ada Rendezvous . . . . .	151
5.4.1	Modelling Interface Objects and Communication Lists . . . . .	156
5.4.2	Primitives . . . . .	161
5.4.3	Communication Events . . . . .	163
5.4.3.1	The Deterministic_Send Event . . . . .	164
5.4.3.2	The Conditional_Send Event . . . . .	166
5.4.3.3	The Deterministic_Receive Event . . . . .	168
5.4.3.4	The Nondeterministic_Receive Event . . . . .	169
5.4.4	Final Comments on Mallgren's Approach . . . . .	171
5.5	An Alternative Approach to Shared Data Abstractions . . . . .	172
5.5.1	Using Shared Data Abstractions . . . . .	189
5.5.2	Critical Regions . . . . .	192
5.5.3	The Waituntil Clause . . . . .	193
5.5.4	The Do Clause . . . . .	195



5.5.5	The Lock Clause . . . . .	196
5.5.6	General Comments . . . . .	200
5.6	An Alternative Model for the Ada Rendezvous . . . . .	201
5.6.1	Abstract Data Types for Modelling the Rendezvous Mechanism of Ada . . . . .	205
5.6.2	Deterministic Send . . . . .	206
5.6.3	Conditional Send . . . . .	208
5.6.4	Deterministic Receive . . . . .	210
5.6.5	Conditional Receive . . . . .	212
5.7	Summary . . . . .	212
<b>6</b>	<b>Summary and Conclusions</b>	<b>215</b>
6.1	Summary . . . . .	215
6.2	Conclusions . . . . .	219
6.3	Future Work . . . . .	220
<b>A</b>	<b>Predefined ADT's within ATLANTIS</b>	<b>227</b>
A.1	The Boolean Data Type . . . . .	227
A.2	The Integer Data Type . . . . .	228
A.3	The Floating Point Data Type . . . . .	229
A.4	The Character Data Type . . . . .	230
A.5	The String Data Type . . . . .	231
A.6	The Location Data Type . . . . .	233
<b>B</b>	<b>ADT implementation</b>	<b>235</b>
B.1	The Table Package . . . . .	235

<b>C</b>	<b>ADT's for the Neptune Definition</b>	<b>244</b>
C.1	The Stack Data Type . . . . .	244
C.2	The Table Data Type . . . . .	245
C.3	The Symbol_Table Data Type . . . . .	246
C.4	The Block_Info Data Type . . . . .	248
C.5	The Declaration Data Type . . . . .	249
C.6	The Initial_Value Data Type . . . . .	250
C.7	The Kind Data Type . . . . .	250
C.8	The Operation Data Type . . . . .	251
C.9	The Basic_Types Data Type . . . . .	252
C.10	The Attributes Data Type . . . . .	253
<b>D</b>	<b>HLO's from the Neptune Definition</b>	<b>256</b>
D.1	Procedure Initialize . . . . .	256
D.2	Procedure Reset_Environment . . . . .	256
D.3	Procedure Check_Block . . . . .	256
D.4	Procedure Create_Block . . . . .	257
D.5	Procedure Enter_Block . . . . .	257
D.6	Procedure Exit_Block . . . . .	257
D.7	Procedure Leave_Block . . . . .	257
D.8	Procedure Note_Identifier . . . . .	257
D.9	Function Recall_Identifier . . . . .	258
D.10	Procedure Note_Previous_Arguments . . . . .	258
D.11	Procedure Recall_Previous_Arguments . . . . .	258
D.12	Procedure Record_Type . . . . .	258

D.13 Procedure Record_Op . . . . .	258
D.14 Procedure Record_Integer . . . . .	258
D.15 Procedure Record_Real . . . . .	259
D.16 Procedure Record_Boolean . . . . .	259
D.17 Procedure Record_String . . . . .	259
D.18 Procedure Reverse_Ident_Stack . . . . .	259
D.19 Procedure Add_To_Symbol_Table . . . . .	259
D.20 Procedure Add_Procedure_To_Symbol_Table . . . . .	260
D.21 Procedure Add_Function_To_Symbol_Table . . . . .	260
D.22 Procedure Ensure_Within_Function . . . . .	260
D.23 Procedure Ensure_Within_Procedure . . . . .	260
D.24 Procedure Ensure_Within_Loop . . . . .	260
D.25 Procedure Start_Loop . . . . .	261
D.26 Procedure Finish_Loop . . . . .	261
D.27 Procedure Inherit_Via_Scope_Rules . . . . .	261
D.28 Procedure Record_Start_Of_Block . . . . .	261
D.29 Procedure Return_To_Caller . . . . .	261
D.30 Function Reverse_Type_Stack . . . . .	262
D.31 Procedure Transfer_To_Argument_List . . . . .	262
D.32 Procedure Check_Call_To_Procedure . . . . .	262
D.33 Procedure Check_Call_To_Function . . . . .	262
D.34 Procedure Check_Call_To_Variable . . . . .	262
D.35 Procedure Check_Boolean_Type . . . . .	262
D.36 Procedure Check_Numeric_Type . . . . .	263
D.37 Procedure Negate_Expression . . . . .	263

D.38 Procedure Not_Expression . . . . .	263
D.39 Procedure Determine_Exit_Statement . . . . .	263
D.40 Procedure Determine_Branch . . . . .	263
D.41 Function Compatible_Types . . . . .	263
D.42 Procedure Equality_Test . . . . .	264
D.43 Procedure Inequality_Test . . . . .	264
D.44 Procedure And_Operation . . . . .	264
D.45 Procedure Or_Operation . . . . .	264
D.46 Procedure Less_Than_Operation . . . . .	264
D.47 Procedure Greater_Than_Operation . . . . .	265
D.48 Procedure Less_Or_Equal_Operation . . . . .	265
D.49 Procedure Greater_Or_Equal_Operation . . . . .	265
D.50 Procedure Plus_Operation . . . . .	265
D.51 Procedure Minus_Operation . . . . .	265
D.52 Procedure Mult_Operation . . . . .	265
D.53 Procedure Divide_Operation . . . . .	266
D.54 Procedure Evaluate_Expression . . . . .	266
D.55 Procedure Perform_Output . . . . .	266
D.56 Procedure Perform_Input . . . . .	266
D.57 Procedure Perform_Assignment . . . . .	266
D.58 Procedure Invoke_Block . . . . .	266
D.59 Procedure Invoke_Block_Or_Variable . . . . .	267

# List of Tables

4.1	The syntactic categories for the definition of special lexical elements.	99
4.2	EBNF symbols used in ATLANTIS. . . . .	99
5.1	Call and return events for the infinite queue shared data abstraction.	142
5.2	The operations applicable to a synchronous communication port. . . .	155
5.3	The operations applicable to an asynchronous communication port. . .	156
5.4	The operations applicable to an “In_Table_ADT” object. . . . .	158
5.5	The operations applicable to an “Out_Table_ADT” object. . . . .	159

# List of Figures

1.1	Two of the test programs developed. . . . .	10
2.1	Algebraic specification of the abstract data type “Table”. . . . .	27
2.2	Collapsing the abstract data type “Table”. . . . .	32
2.3	The type “boolean”. . . . .	36
2.4	Revised abstract data type specification for “Table”. . . . .	40
3.1	Layering an operational semantic model. . . . .	53
3.2	Repetition (while-loop) defined in terms of selection. . . . .	56
3.3	Repetition (repeat-until) defined in terms of selection. . . . .	56
3.4	The form and definition of the for-loop. . . . .	57
3.5	Enhancing syntax with semantics. . . . .	60
3.6	A Pascal example. . . . .	61
3.7	Two-pass semantic description. . . . .	61
3.8	An abstract data type describing a FIFO list. . . . .	64
3.9	The abstract data type <code>Link_Type</code> . . . . .	65
3.10	A Pascal program and its static environment. . . . .	68
3.11	The dynamic environment for a Pascal program. . . . .	70
3.12	The high-level operation <code>member_of_symbol_table</code> . . . . .	71
3.13	The high-level operation <code>add_new_info</code> . . . . .	72

3.14	Local declarations. . . . .	75
3.15	Scope rules for Pascal – the first pass. . . . .	77
3.16	Scope rules for Pascal – the second pass. . . . .	79
3.17	Variable parameters. . . . .	81
3.18	Value parameters. . . . .	83
3.19	Procedural and functional parameters. . . . .	85
4.1	The keyword section of an ATLANTIS definition. . . . .	96
4.2	The operator specification within the definition of Neptune. . . . .	98
4.3	The special token definitions within an ATLANTIS definition. . . . .	100
4.4	The syntactic definition of a Pascal block. . . . .	102
4.5	ATLANTIS definition of a Pascal block with semantic definitions. . . . .	104
4.6	An identifier definition for ATLANTIS. . . . .	105
4.7	Flow chart illustrating syntax and semantics. . . . .	107
4.8	The definition of a table for input to ATLANTIS. . . . .	109
4.9	Assignment of ATLANTIS environment variables. . . . .	117
4.10	A procedure high level operation. . . . .	119
4.11	A function high level operation. . . . .	120
4.12	The <b>if</b> action within ATLANTIS. . . . .	121
4.13	Semantics of the <b>exit</b> action from within a loop. . . . .	122
4.14	Structure of the <b>while</b> loop. . . . .	122
4.15	Structure of the <b>for</b> loop. . . . .	123
4.16	Display the contents of an identifier stack. . . . .	123
4.17	Overall structure of ATLANTIS. . . . .	127
4.18	A simple language definition. . . . .	129
4.19	The parse tree for the input “ACC”. . . . .	130

4.20	The rules and the corresponding tree. . . . .	131
5.1	The shared data abstraction specification of an infinite queue. . . . .	141
5.2	A sample Ada program. . . . .	153
5.3	Communication paths of Ada tasks. . . . .	153
5.4	Types used in the description of Ada tasks. . . . .	158
5.5	Types used in the description of Ada tasks. . . . .	162
5.6	The “Deterministic_Send” event for Ada. . . . .	165
5.7	The “Conditional_Send” event for Ada. . . . .	167
5.8	The “Deterministic_Receive” event for Ada. . . . .	169
5.9	The “Nondeterministic_Receive” event for Ada. . . . .	170
5.10	Sequential and parallel access to an abstract data type object. . . . .	173
5.11	An example of the application of the object access rules. . . . .	175
5.12	An abstract data type representing a queue. . . . .	176
5.13	The specification of the queue data type. . . . .	177
5.14	The SDA envelope specification for the queue data type. . . . .	180
5.15	The package body for the shared data abstraction envelope for the queue abstract data type. . . . .	182
5.16	The specification of the “sda_monitor” package. . . . .	186
5.17	Example of the use of a shared data abstraction. . . . .	190
5.18	The textual form of a critical region. . . . .	193
5.19	Definition of a critical region in terms of semaphores. . . . .	193
5.20	A diagrammatic representation of a critical region. . . . .	193
5.21	The textual form of a waituntil clause. . . . .	194
5.22	Definition of the waituntil clause using semaphores. . . . .	194
5.23	A diagram representing the waituntil clause. . . . .	195



5.24	The textual form of a do clause. . . . .	196
5.25	Definition of the do clause using semaphores. . . . .	197
5.26	A diagrammatic view of the do clause. . . . .	198
5.27	Syntactic and diagrammatic representation of locked regions. . . . .	200
5.28	A record describing each Ada task. . . . .	202
5.29	Information associated with each entry point within a task. . . . .	202
5.30	An Ada program fragment. . . . .	203
5.31	The definition of a communication port. . . . .	205
5.32	The deterministic send event. . . . .	206
5.33	The conditional send event. . . . .	209
5.34	The deterministic receive event. . . . .	211
5.35	The conditional receive event. . . . .	213