

The Whiley Language Specification

David J. Pearce
School of Engineering and Computer Science
Victoria University of Wellington, New Zealand
djp@ecs.vuw.ac.nz

December 31, 2013

Contents

1	Introduction	3
1.1	Overview	3
1.2	Goals	3
1.3	History	3
2	Lexical Structure	4
2.1	Indentation	4
2.2	Blocks	4
2.3	Whitespace	4
2.4	Identifiers	4
3	Compilation Units	5
3.1	Type Declarations	5
3.2	Constant Declarations	5
3.3	Function & Method Declarations	5
3.4	Visibility Modifiers	5
3.5	Packages	5
3.6	Imports	5
4	Types	6
4.1	Overview	6
4.2	Primitives	6
4.2.1	Any Type	6
4.2.2	Void Type	6
4.2.3	Null Type	7
4.2.4	Bool Type	7
4.2.5	Char Type	7
4.2.6	Int Type	7
4.2.7	Real Type	8
4.3	Collection Types	8
4.3.1	Set Type	8
4.3.2	Map Type	8
4.3.3	List Type	9
4.4	Union Types	9
4.5	Intersection Types	9
4.6	Negation Types	9
4.7	Reference Types	9
4.8	Subtyping	9
5	Expressions	10
5.1	Binary Expressions	10

6	Statements	12
6.1	Variable Declarations	12
6.2	Assign Statements	12
6.3	Return Statements	12
6.4	If/Else Statements	12
6.5	While Statements	12
6.6	Do/While Statements	12
6.7	For Statements	12
6.8	Switch Statements	12
6.9	Try/Catch Statements	12

Chapter 1

Introduction

1.1 Overview

1.2 Goals

1.3 History

Chapter 2

Lexical Structure

2.1 Indentation

2.2 Blocks

2.3 Whitespace

2.4 Identifiers

Chapter 3

Compilation Units

3.1 Type Declarations

3.2 Constant Declarations

3.3 Function & Method Declarations

3.4 Visibility Modifiers

3.5 Packages

3.6 Imports

Chapter 4

Types

4.1 Overview

Discuss syntactic versus semantic types.

4.2 Primitives

4.2.1 Any Type

```
AnyType ::= any // type any
```

Description.

Examples.

Semantics.

Notes.

4.2.2 Void Type

```
VoidType ::= void // type void
```

Description. The `void` type represents the type whose variables cannot exist! That is, they cannot hold any possible value. Void is used to represent the return type of a function which does not return anything. However, it is also used to represent the element type of an empty list or set.

Examples.

Semantics.

Notes. The void type is a subtype of everything; that is, it is bottom in the type lattice.

4.2.3 Null Type

```
NullType ::= null // type null
```

Description.

Examples.

Semantics.

Notes.

4.2.4 Bool Type

```
BoolType ::= bool // type bool
```

Description.

Examples.

Semantics.

Notes.

4.2.5 Char Type

```
CharType ::= char
```

Description.

Examples.

Semantics.

Notes.

4.2.6 Int Type

```
IntType ::= int
```

Description.

Examples.

Semantics.

Notes.

4.2.7 Real Type

$$\text{RealType} ::= \boxed{\text{real}}$$

Description.

Examples.

Semantics.

Notes.

4.3 Collection Types

4.3.1 Set Type

$$\text{SetType} ::= \boxed{\{ \text{Type} \}}$$

Description.

Examples.

Semantics.

Notes.

4.3.2 Map Type

$$\text{MapType} ::= \boxed{\{ \text{Type} \Rightarrow \text{Type} \}}$$

Description.

Examples.

Semantics.

Notes.

4.3.3 List Type

$$\text{ListType} ::= [\text{Type}]$$

Description.

Examples.

Semantics.

Notes.

4.4 Union Types

4.5 Intersection Types

4.6 Negation Types

4.7 Reference Types

4.8 Subtyping

Discussion or present subtyping algorithm?

Expr	::=	Cond [(&&) Expr]	// Expressions
Cond	::=	Append [Cop Expr]	// Condition Expressions
Append	::=	Range [++ Expr]	// Append Expressions
Range	::=	AddSub [.. Expr]	// Range Expressions
AddSub	::=	MulDiv [(+ -) Expr]	// Additive Expressions
MulDiv	::=	Index [(* / %) Expr]	// Multiplicative Expressions
Index	::=	???	// Index Expressions

Figure 5.1: Syntax for Binary Expressions

Chapter 5

Expressions

5.1 Binary Expressions

Term	::=	<i>// Terms</i>	
	<i>Constant</i>		<i>// Constant expressions</i>
	<i>Identifier</i>		<i>// Identifier expressions</i>
	$Expr_1 ([, Expr_i]^+)$		<i>// Tuple expressions</i>
	$([Expr])$		<i>// Bracketed expressions</i>
	$[Expr]$		<i>// Size expressions</i>
	$Identifier ([Expr_1 ([, Expr_i]^+)])$		<i>// Invocation expressions</i>
	$([- ! \sim \& *] Expr)$		<i>// Unary expressions</i>
	$new Expr$		<i>// Allocation expressions</i>
	$\{ [Expr_1 ([, Expr_i]^*)] \}$		<i>// Set expressions</i>
	$\{ [Expr_1 \Rightarrow Expr'_1 ([, Expr_i \Rightarrow Expr'_i]^*)] \}$		<i>// Map expressions</i>
	$[[Expr_1 ([, Expr_i]^*)]]$		<i>// List expressions</i>
	$\{ [n_1 : Expr_1 ([, n_i : Expr_i]^*)] \}$		<i>// Record expressions</i>

Figure 5.2: Syntax for Term Expressions

Constant	::=	<i>// Constants</i>	
	$([0 1])^+ [b]$		<i>// Boolean constants</i>
	$([0-9])^+$		<i>// Integer constants</i>
	$([0-9])^+ [.] ([0-9])^+$		<i>// Decimal constants</i>
	$null$		<i>// Null constant</i>

Figure 5.3: Syntax for Constant Expressions

Identifier	::=	$([- a-z A-Z] ([- a-z A-Z 0-9])^*)$	<i>// Identifiers</i>
-------------------	------------	---	-----------------------

Figure 5.4: Syntax for Identifiers

Chapter 6

Statements

6.1 Variable Declarations

6.2 Assign Statements

6.3 Return Statements

6.4 If/Else Statements

6.5 While Statements

6.6 Do/While Statements

6.7 For Statements

6.8 Switch Statements

6.9 Try/Catch Statements