

The Whiley Language Specification

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

January 2, 2014

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	7
4.2.2	Void Type	7
4.2.3	Null Type	7
4.2.4	Bool Type	7
4.2.5	Byte Type	8
4.2.6	Char Type	8
4.2.7	Int Type	8
4.2.8	Real Type	9
4.3	Tuple Types	9
4.4	Record Types	9
4.5	Reference Types	10
4.6	Nominal Types	10
4.7	Collection Types	10
4.7.1	Set Type	10
4.7.2	Map Type	10
4.7.3	List Type	11
4.8	Union Types	11
4.9	Intersection Types	11
4.10	Negation Types	12
4.11	Subtyping	12

5	Expressions	13
5.1	Binary Expressions	13
6	Statements	15
6.1	Variable Declarations	15
6.2	Assign Statements	15
6.3	Return Statements	15
6.4	If/Else Statements	15
6.5	While Statements	15
6.6	Do/While Statements	15
6.7	For Statements	15
6.8	Switch Statements	15
6.9	Try/Catch Statements	15

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.

```
Type ::=
      | TermType
      | UnionType
      | IntersectionType
```

```
TermType ::=
        | PrimitiveType
        | TupleType
        | RecordType
        | ReferenceType
        | NominalType
        | CollectionType
        | NegationType
```

4.2 Primitives

```
PrimitiveType ::=
              | AnyType
              | VoidType
              | NullType
              | BoolType
              | ByteType
              | CharType
              | IntType
              | RealType
```

4.2.1 Any Type

```
AnyType ::= any
```

Description. The type **any** represents the type whose variables may hold any possible value.

Examples.

Semantics.

Notes. The **any** type is top in the type lattice. That is, it is the supertype of all other types.

4.2.2 Void Type

```
VoidType ::= 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
```

Description. The null type is a special type which should be used to show the absence of something. It is distinct from void, since variables can hold the special **null** value (whereas there is no special "void" value).

Examples.

Semantics.

Notes. With all of the problems surrounding **null** and `NullPointerException`s in languages like Java and C, it may seem that this type should be avoided. However, it remains a very useful abstraction to have around and, in Whiley, it is treated in a completely safe manner (unlike e.g. Java).

4.2.4 Bool Type


```
BoolType ::= bool
```

Description. Represents the set of boolean values (i.e. `true` and `false`).

Examples.

Semantics.

Notes.

4.2.5 Byte Type

```
ByteType ::= byte
```

Description. Represents a sequence of 8 bits.

Examples.

Semantics.

Notes. Unlike for many languages, there is no representation associated with a byte. For example, to extract an integer value from a byte, it must be explicitly decoded according to some representation (e.g. two's complement) using an auxiliary function (e.g. `Byte.toInt()`).

4.2.6 Char Type

```
CharType ::= char
```

Description.

Examples.

Semantics.

Notes.

4.2.7 Int Type

```
IntType ::= int
```

Description.

Examples.

Semantics.

Notes.

4.2.8 Real Type

```
RealType ::= real
```

Description.

Examples.

Semantics.

Notes.

4.3 Tuple Types

```
TupleType ::= ( Type ( , Type )* )
```

Description.

Examples.

Semantics.

Notes.

4.4 Record Types

```
RecordType ::= { Type Identifier ( , Type Identifier )* }
```

Description.

Examples.

Semantics.

Notes.

4.5 Reference Types

```
ReferenceType ::= & Type
```

Description.

Examples.

Semantics.

Notes.

4.6 Nominal Types

```
NominalType ::= Identifier
```

Description.

Examples.

Semantics.

Notes.

4.7 Collection Types

4.7.1 Set Type

```
SetType ::= { Type }
```

Description.

Examples.

Semantics.

Notes.

4.7.2 Map Type

```
MapType ::= { Type => Type }
```

Description.

Examples.

Semantics.

Notes.

4.7.3 List Type

```
ListType ::= [ Type ]
```

Description.

Examples.

Semantics.

Notes.

4.8 Union Types

```
UnionType ::= IntersectionType ( | IntersectionType )+
```

Description.

Examples.

Semantics.

Notes.

4.9 Intersection Types

```
IntersectionType ::= TermType ( & TermType )+
```

Description.

Examples.

Semantics.

Notes.

4.10 Negation Types

$$\text{NegationType} ::= \boxed{!} \text{Type}$$

Description.

Examples.

Semantics.

Notes.

4.11 Subtyping

Discussion or present subtyping algorithm?

Expr	::=	Cond [($\boxed{\&\&}$ $\boxed{ }$) Expr]	// Expressions
Cond	::=	Append [Cop Expr]	// Condition Expressions
Append	::=	Range [$\boxed{++}$ Expr]	// Append Expressions
Range	::=	AddSub [$\boxed{..}$ Expr]	// Range Expressions
AddSub	::=	MulDiv [($\boxed{+}$ $\boxed{-}$) Expr]	// Additive Expressions
MulDiv	::=	Index [($\boxed{*}$ $\boxed{/}$ $\boxed{\%}$) 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