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

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

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 of 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 (where as 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 compliment) using an auxillary function (e.g. `Byte.toInt()`).

4.2.6 Char Type

```
CharType ::= char
```

Description. Represents a unicode character.

Examples.

Semantics.

Notes.

4.2.7 Int Type

```
IntType ::= int
```

Description. Represents the set of (unbound) integer values.

Examples.

Semantics.

Notes. Since integer types in Whiley are unbounded, there is no equivalent to Java's `MIN_VALUE` and `MAX_VALUE` for **`int`** types.

4.2.8 Real Type

```
RealType ::= real
```

Description. Represents the set of (unbound) rational numbers.

Examples.

Semantics.

Notes.

4.3 Tuple Types

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

Description. A tuple type describes a compound type made up of two or more subcomponents. It is similar to a record, except that fields are effectively anonymous.

Examples.

Semantics.

Notes.

4.4 Record Types

```
RecordType ::= { Type Ident ( , Type Ident )* [ , ... ] }
```

Description. A record is made up of a number of fields, each of which has a unique name. Each field has a corresponding type. One can think of a record as a special kind of "fixed" map (i.e. where we know exactly which entries we have).

Examples.

Semantics.

Notes.

4.5 Reference Types

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

Description. Represents a reference to an object in Whiley.

Examples.

Semantics.

Notes.

4.6 Nominal Types

```
NominalType ::= Ident
```

Description. The existential type represents the an unknown type, defined at a given position.

Examples.

Semantics.

Notes.

4.7 Collection Types

4.7.1 Set Type

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

Description. A set type describes set values whose elements are subtypes of the element type. For example, `{1,2,3}` is an instance of set type `{int}`; however, `{1.345}` is not.

Examples.

Semantics.

Notes.

4.7.2 Map Type

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

Description.

Examples.

Semantics.

Notes.

4.7.3 List Type

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

Description. A list type describes list values whose elements are subtypes of the element type. For example, `[1,2,3]` is an instance of list type `[int]`; however, `[1.345]` is not.

Examples.

Semantics.

Notes.

4.8 Union Types

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

Description. A union type represents a type whose variables may hold values from any of its ”bounds”. For example, the union type `null|int` indicates a variable can either hold an integer value, or `null`.

Examples.

Semantics.

Notes. There must be at least two bounds for a union type to make sense.

4.9 Intersection Types

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

Description.

Examples.

Semantics.

Notes.

4.10 Negation Types

NegationType	::=	$\boxed{\texttt{!}}$ `Type`

Description. A negation type represents a type which accepts values *not* in a given type.

Examples.

Semantics.

Notes.

4.11 Abstract Types

4.11.1 Recursive Types

4.11.2 Effective Tuples

4.11.3 Effective Records

4.11.4 Effective Collections

4.12 Subtyping

Discussion or present subtyping algorithm?

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

Figure 5.1: Syntax for Binary Expressions

Chapter 5

Expressions

5.1 Binary Expressions

$$
\begin{array}{llll}
\text{Term} & ::= & \textit{// Terms} & \\
& \mid & Constant & \textit{// Constant expressions} \\
& \mid & Identifier & \textit{// Identifier expressions} \\
& \mid & Expr_1 \; (\boxed{\texttt{,}} \; Expr_i \;)^+ & \textit{// Tuple expressions} \\
& \mid & \boxed{\texttt{(}} \; Expr \; \boxed{\texttt{)}} & \textit{// Bracketed expressions} \\
& \mid & \boxed{\texttt{|}} \; Expr \; \boxed{\texttt{|}} & \textit{// Size expressions} \\
& \mid & Identifier \; \boxed{\texttt{(}} \; [\; Expr_1 \; (\boxed{\texttt{,}} \; Expr_i \;)^+ \;] \; \boxed{\texttt{)}} & \textit{// Invocation expressions} \\
& \mid & (\boxed{\texttt{-}} \mid \boxed{\texttt{!}} \mid \boxed{\texttt{\~{}}} \mid \boxed{\texttt{\&}} \mid \boxed{\texttt{*}} \;) \; Expr & \textit{// Unary expressions} \\
& \mid & \boxed{\texttt{new}} \; Expr & \textit{// Allocation expressions} \\
& \mid & \boxed{\{} \; [\; Expr_1 \; (\boxed{\texttt{,}} \; Expr_i \;)^* \;] \; \boxed{\}} & \textit{// Set expressions} \\
& \mid & \boxed{\{} \; [\; Expr_1 \; \boxed{\texttt{=>}} \; Expr'_1 \; (\boxed{\texttt{,}} \; Expr_i \; \boxed{\texttt{=>}} \; Expr'_i \;)^* \;] \; \boxed{\}} & \textit{// Map expressions} \\
& \mid & \boxed{\texttt{[}} \; [\; Expr_1 \; (\boxed{\texttt{,}} \; Expr_i \;)^* \;] \; \boxed{\texttt{]}} & \textit{// List expressions} \\
& \mid & \boxed{\{} \; [\; n_1 \boxed{\texttt{:}} \; Expr_1 \; (\boxed{\texttt{,}} \; n_i \; \boxed{\texttt{:}} \; Expr_i \;)^* \;] \; \boxed{\}} & \textit{// Record expressions}
\end{array}
$$

Figure 5.2: Syntax for Term Expressions

$$
\begin{array}{llll}
\text{Constant} & ::= & \textit{// Constants} & \\
& \mid & (\boxed{\texttt{0}} \mid \boxed{\texttt{1}} \;)^+ \; \boxed{\texttt{b}} & \textit{// Boolean constants} \\
& \mid & (\boxed{\texttt{0-9}} \;)^+ & \textit{// Integer constants} \\
& \mid & (\boxed{\texttt{0-9}} \;)^+ \; \boxed{\texttt{.}} \; (\boxed{\texttt{0-9}} \;)^+ & \textit{// Decimal constants} \\
& \mid & \boxed{\texttt{null}} & \textit{// Null constant}
\end{array}
$$

Figure 5.3: Syntax for Constant Expressions

$$
\text{Identifier} \quad ::= \quad (\boxed{\texttt{\_}} \mid \boxed{\texttt{a-z}} \mid \boxed{\texttt{A-Z}} \;) \; (\boxed{\texttt{\_}} \mid \boxed{\texttt{a-z}} \mid \boxed{\texttt{A-Z}} \mid \boxed{\texttt{0-9}} \;)^* \qquad \textit{// Identifiers}
$$

Figure 5.4: Syntax for Identifiers

Chapter 6

Statements