Maxwell Language Design

F. Schuiki (fschuiki@student.ethz.ch)

February 18, 2014

Listing 1: Iterator interface definition

```
type Iterator interface {
  next (iterator * @) -> nil;
  get (iterator @) -> any;
  end (iterator @) -> Bool;
}
```

Contents

1 Miscellaneous

This section acts as a placeholder for random information.

1.1 Array Literals

An array literal is a constant, static array defined in source code. It is not intended to be modified and supports only a rather simplistic interface:

- get (index Int) -> A
- length -> Int

This corresponds to the *ConstArray* interface, which declares a function to access individual elements, and a function to obtain the overall length of the array.

1.2 Iterators

An iterator is a structure which walks through elements of a collection; i.e. an array, set, map or string. Walking through a linear collection such as a vector array is simple, as the elements may be accessed by a continuously increasing index. In case of more complex structures, such as sets or maps, stepping from one element to another is not as simple anymore. Iterators provide a way to abstract this operation, allowing each collection to implement their most efficient iteration technique.

The language shall define an *Iterator* interface as shown in listing 1. The *next* function advances the iterator to the next element, or does nothing if the iterator went past the last element of its collection. The *get* function provides access to the element the iterator is currently pointing at.

2 Sequential Expressions

Since everything is an expression in Maxwell, there are no statements. Blocks `{ ... }` in classic C-like languages are statements, i.e. they describe a computational step and do not have a value. In Maxwell, blocks also have a return value, depending on the syntax used for their definition. The following possibilities exist:

- *Implicit return value.* The basic C syntax for a block returns the last expression of the block as its value. `{ expr0; expr1; ... exprn; } = exprn;`
- *Explicit return value.* The block expression may be prefixed by a return variable name which may be assigned a value inside the block body. The variable may be used like any other variable or function argument, and will be returned as the block's value. `r { expr0; r = expr1; expr2; } = r;`

Listings