Homework1 – Patryk Konieczny

Using the Language

In order to evaluate these following expressions use .eval at the end of the expression.

Assign(name : “String”, set : Set[Any]) => Set[Any]

The Assign expression creates a binding between a string and a set of type Set[Any]. This binding is stored using a map within the language. The first parameter takes a string, which is the key to the map, and the second parameter is a set, which is the value of the map. The user can nest operation within the second parameter if those operation return a set. The assign method returns the set that it is creating a binding for.

Insert(value: Any*)=> Set[Any]

The Insert expression creates a Set of objects and returns that set. The objects that are stored in that Set are passed as parameters by the user. The user can insert as many parameters as possible and of Any type. Set returned by insert can only be bound is used within the Assign expression.
Delete(name: “String” , value: Any) => Set[Any]

The Delete expression finds a set that has a binding and deletes a value from that set. The first parameter is the name of the set that should have an element deleted, and the second parameter is value that should be deleted from the set. The set without the item is returned.

Union(set1: Set[Any], set2: Set[Any]) => Set[Any]

The Union expression takes two parameters that are of types Set[Any]. It returns the set that has elements from set1 and from set2. Other expressions can be nested into Union if they return a set of type Set[Any].

Intersect (set1: Set[Any], set2: Set[Any]) => Set[Any]

The Intersect expression takes two parameters that are of type Set[Any]. It returns a set of elements that exist both in set1 and in set2. Other expressions can be nested into Intersect if they return a set of type Set[Any]..

SetDifference(set1: Set[Any], set2: Set[Any]) => Set[Any]

The SetDifference expression takes two parameters that are of Set[Any]. It returns the set of values in set1 that are not in set 2. Other expressions can be nested into SetDifference if they return a set of type Set[Any]..

SymmDifference(set1: Set[Any], set2: Set[Any]) => Set[Any]

The SymmDifference(Symmetrical Difference) expression takes two parameters of Set[Any]. It returns the set of values that are not in both set1 and in set2. Other expressions can be nested into SetDifference if they return a set of type Set[Any].

CartProduct(set1: Set[Any], set2: Set[Any])) => Set[Tuple(Any, Any)]

The CartPreduct(Cartesian Product) expression takes two parameters of Set[Any]. It returns the set of all possible order pairs between set1 and set2.

Macro(name: “String”, obj: Set[Any]) => Set[Any]

The Macro expression takes a parameter name and a parameter obj. The name parameter is a string that binds that name to the result of the expression that is passed through the obj parameter. Macro returns the set that is a result of the expression passed in the obj parameter. To assign the macro, use Var(name) in the second parameter of Assign. This will Assign the result of that Macro.
Scope(name : “String”, obj: Set[Any]) => Set[Any]

The Scope expression takes a parameter name and obj. The name is a string that is the name of the created scope. The obj is the expressions that bound to that scope. These expressions return a Set[Any], then the expression binds the name of the scope to the set. The expression also returns a type of Set[Any]

In order to evaluate the following expression, use . checkItem after the expression.

Check(name: “String”, value: Any) => Boolean

This expression takes a parameter name of type string and a value of type any. This expression checks if value exists in the set that is bounded to the name that is passed in. This returns true if it exists and false if it does not.