Dynamic Languages – Übungen Blatt 8

Abgabedatum: `22. Juni 2010`

These exercises continue building the interpreter, working towards making the language more usable. In case you are not happy with your solutions to last weeks exercises (or in case you didn't solve them) you can find example solutions on the web site. As usual, add tests about cases that are not tested by the existing test functions.

Happy hacking!

Aufgabe 1 – Primitives

(5 Punkte)

The integers as they are in the language so far are fairly uninteresting. They exist as constants in the source code but the only operation that can be performed on them is testing whether they are zero or not (using the `if` statement). However, within the language it would not be possible to write integer addition. Therefore the interpreter needs to be extended.

The goal of this exercise is to add a mechanism to the interpreter to define so-called primitives. Primitives are methods that are implemented on the level of the interpreter (i.e. in Python). Primitives are called using a special syntax:

```
receiver $primitivename(arg_1, ..., arg_n)
```

For this, a new AST node called `PrimitiveMethodCall` has been added. The primitive (which starts with a `$`) is then looked up in some global dictionary of all primitives and the Python function that is found there is called. For this exercise you should implement the general primitive mechanism and implement the primitive `$int_add` which adds two integers. There are tests about the intended behaviour in `test_primitive.py`.

(The number of primitives for a "real" interpreter is typically large, e.g. the original Self implementation had more than 1000 primitives.)

Aufgabe 2 – Prototypes

(5 Punkte)

The basic object model built last week does not support any prototypical features, e.g. delegation and parents. The goal of this exercise is to add those. There are tests about the intended behaviour in `test_parent.py`.

- Every object has one or more parent attributes. Every object has at least the attribute `__parent__` as a parent attribute, which is the last parent.

To get an attribute from a lookup, the method-resolution-order of the object is computed using the C3 algorithm and the lookup proceeds in that order. The C3 algorithm is implemented in `c3computation.py`, you can just use that file or use your own implementation from Blatt 6. Setting an attribute will always set the attribute directly in the receiver object, whether the attribute is already defined in a parent or not.

The implicit `__parent__` attribute is used to implement scoping. Whenever a new object is created using the `object` statement (or a new method using the `def` statement) the `__parent__` attribute of the new object (method) is set to the current implicit self. This makes sure that every object has access to its enclosing scope.

- To define other parent attributes the `object` syntax is extended to allow the following sort of code:

```
object a(parent1=b, parent2=c):
    ...
```

This will declare the parent attributes `"parent1"` and `"parent2"` on `a` and set them to `b` and `c` respectively.