Multi-Methods in Racket

Daniel Cardoso 66964
Francisco Raposo 66986
Miguel Roxo 66259

Structs

```
struct method {
  list<procedure> types;
  procedure function;
}

struct generic-function {
  symbol name;
  list<symbol> parameters;
  list<method> methods = new list<method>();
}
```

- `hash-table<procedure,procedure> (subtype,supertype) to store specificity relations`
- `generic-function-parameters, generic-function-methods, method-types trivially achieved`

defsubtype, defgeneric, defmethod

```
(defsubtype subtype supertype)
  hash-table.Put(subtype, supertype)

(defgeneric name parameters)
  name = new generic-function(name, parameters)

(defmethod name ((arg pred) …) body …)
  types = list(pred …)
  function = lambda (arg …) { body …}
  methods = name.methods
  method = new method(types, function)
  name.methods = update-methods(methods, method)
```

Method (re-)definition

```
update-methods(list<method> methods, method new-method) {
  cond {
    methods == empty-list
      new-method
    eq-predicates?(methods[0].types, method.types)
      new-method ++ methods
    else
      methods[0] ++ update-methods(methods[1..end], new-method)
}
```

- keeps definition order

Method (re-)definition

```
eq-predicates?(list<procedure> list1, list<procedure> list2) {
  cond {
    list1.size() != list2.size()
      #f
    list1.size() == 0
      #t
    list1[0] == list2[0]
      eq-predicates?(list1[1..end], list2[1..end])
    else
      #f
}
```

Call protocol

```
most-specific-method(list<method> methods, list<?> parameters) {
  sorted-methods = sort(methods, lambda (m1, m2) { more-specific?(m1.types, m2.types) })
  find-most-specific(sorted-methods, parameters)
}
```

Call protocol

```
more-specific?(list<procedure> pred1, list<procedure> pred2) {
  cond {
    subtype?(pred2[0], pred1[0])
      #f
    subtype?(pred1[0], pred2[0])
      #t
    pred1[1..end] == empty-list
      #f
    else
      more-specific?(pred1[1..end], pred2[1..end])
  }
}
```

Call protocol

```
subtype?(procedure subtype, procedure supertype) {
  parent = hash-table.Get(subtype)
  if(parent == null)
    #f
  if(parent == supertype)
    #t
  subtype?(parent, supertype)
}
```

Call protocol

```
find-most-specific(list<method> methods, list<?> parameters) {
  if(methods == empty-list)
    error(“Method missing for arguments ” ++ parameters)
  if(applicable?(methods[0].types, parameters))
    methods[0]
  find-most-specific(methods[1..end], parameters)
}
```

Call protocol

```
applicable?(list<procedure> predicates, list<?> parameters) {
  if(predicates == empty-list && parameters == empty-list)
    #t
  if(predicates.size() != parameters.size())
    #f
  if(predicates[0](parameters[0]))
    applicable?(predicates[1..end], parameters[1..end])
  #f
}
```