A Meta-Circular Evaluator for Julia

Advanced Programming - Group 15

Vasco Correia - 94188
Gonçalo Guerreiro - 95581
Carlos Vaz - 99188
Javier María - 99240

Topics

1. Internal representation and implementation details for variables.
2. Implementation details for `functions`, `macros`, `fexprs` and `primitives`.
3. The `global` keyword and its implementation details.
4. Where can eval be used and why is it designed this way.
5. Extra features not required by the assignment

Environment

Environment

1. How do we represent the bindings within a scope?

Environment

1. How do we represent the bindings within a scope?
2. How to support multiple scopes? Vector vs. “Linked List”.

Environment

1. How do we represent the bindings within a scope?
2. How to support multiple scopes? Vector vs. “Linked List”.
3. Implementing the operations to query and manipulate the environment.

Functions, Macros, Fexprs and Primitives

Functions, Macros, Fexprs and Primitives

1. First approach: vectors

```
[:function, parameters, body, environment]
[:macro, parameters, body, environment]
[:fexpr, parameters, body, environment]
[:primitive, func]
```

Functions, Macros, Fexprs and Primitives

1. First approach: vectors
2. Final solution: structures

```
struct Function
  parameters
  body
  environment
end
```

```
struct Macro
  parameters
  body
  environment
end
```

```
struct Fexpr
  parameters
  body
  environment
end
```

```
struct Primitive
  func
end
```

Functions, Macros, Fexprs and Primitives

1. First approach: vectors
2. Final solution: structures
3. Override show() methods

```
Base.show(io::IO, f::Function) = print(io, "<function>")
Base.show(io::IO, m::Macro) = print(io, "<macro>")
Base.show(io::IO, f::Fexpr) = print(io, "<fexpr>")
Base.show(io::IO, p::Primitive) = print(io, "<function>")
```

Local environments and global

1. Global scope saved as a global variable, accessible from anywhere in the code (definition of metajulia_eval(expr))
2. Global assignments, function, fexpr and macro definitions.
3. Global declaration (:__uninitialized__) -> Saving information about the variable.

Global declaration (without initialization)

1. Added as support for possible further expansion (for example typing conversion).

```
julia> global y::Int

julia> y = 1.0
1.0

julia> y
1
```

Extension: Overshadow bypassing

1. Julia only allows for definition of same-name local variables with the value of a global counterpart in let initialization. Extend support to same name initialization using global value in the middle of blocks.

```
julia> x = println
println (generic function with 3 methods)

julia> let ;
           x = x(1)
       end
ERROR: UndefVarError: `x` not defined
Stacktrace:
 [1] top-level scope
   @ REPL[20]:2
```

Default julia behaviour

```
sql = Database.execute(provider, login_credentials, query)

function init_postgres_database(login_credentials)
  sql = (query) -> sql(Database.providers.postgres, login_credentials, query)
  ...
end
```

Example where the modified behaviour could be useful

Eval

- One of the main differences between Julia and MetaJulia
  - In MetaJulia, eval can only be accessed from inside a fexpr
- First approach: eval as an initial binding
  - Have a flag inside each environment
  - Throw an error if called from outside a fexpr
- Final approach: eval is only lexically bound inside of a fexpr
  - Programmer can use eval in other contexts

Extension: While loops

1. Our metacircular evaluator also supports while loops. Implemented using julia's own while loop.