giaco.ml

an interpreted EDSL written in OCAML

Mario A. Barbara Andrea Cancellieri

1 Introduction

giaco.ml is an imperative (but also functional) EDSL interpreter written in OCAML.

The interpreter is capable of handling expressions, commands and declarations all within a program.

A program can be parsed into valid code from a file/string thanks to reflection.

Static taint analysis can be performed, guards have been also added to the Reflect command as a form of protection, similar to perl.

The main execution functions are: interpret (or interpret' which returns also the declarations' output), reflect and taint_analysis which all operate on a Prog(ds,cs) construct.

2 Design Choices

The interpreter works with 3 domains:

generic this domain is shared amongst the syntax (external) and semantic (internal) domains

syntax this domain contains all the domains accessible to the user. it features 3 sub-domains

expressions these are entities which may be mapped directly to a value

commands these are entities which allow modifying the global state of the program, though allocating new memory is not allowed

declarations these are entities which allow allocation of new memory in the global state

semantic this domain is used for representing the internal state of the interpreter.

- evaluatable values these are values that may be directly evaluated from an expressions. they represent the base internal type for the interpreter. Basic allowed types are Integer, Boolean, Float, String and Subprograms.
- **environment** this is a mapping representing internal non-mutable state. It maps variables (identifiers) to values.
- store this is a mapping representing internal mutable state. It allows aliasing of 2 identifiers for one value. It maps addresses (pointers or locations) to values.
- denotable values these are values contained in the environment. Basic allowed types are Integer, Boolean, Float, String and Locations.
- mutable memory values these are values contained in the store. Basic allowed types are Integer, Boolean, Float, String.

The interpreter itself is implemented by 3 main functions:

eval this function basically maps expressions to evaluatable values

cval this function maps commands to a changed internal state (store)

dval this function maps declarations to a new internal state (environment + store)

Some important choices:

- memory is viewed as a mapping. it is therefore impossible to modify in ocaml. changing the type signature of the interpreter would allow this
- because memory is a mapping, new locations are simulated by randomly selecting a new location based on given store size
- regular expressions are used in reflection, so the Str module must be available
- fixed points are calculated through ocamls's function and let rec operations
- to ease debugging, the interpreted is not compiled but interpreted in an ocaml shell with the #use "file.ml" construct
- since the program is interpreted, let and between multiple files is not available, this has been fixed by using mutable state pointers to future functions. These pointers will be updated when the function is created.
- many utility functions have been created to ease debugging. most notable are emptyenv emptystore tenv' tstore' tnew' which simplify dealing with memory
- due to the non-recursive nature of Procedure and Call, they have been omitted from reflection and taintanalysis. The syntax restriction is due to the difficulty of implementing a parser for the complete OCAML list syntax, and the static analysis of commands that cannot be recursively unpacked (one lies in the *expressions*, the other in the *commands*)

3 Usage

```
the interpreter can be loaded up with
```

```
>> #use "giaco.ml";;
   and subsequent testing may be performed:
>> (* have a functional sample: *)
>> Plus(Int(3), Int(4)) in eval e emptyenv emptystore;;
 : evalue = EInt 7
>> (* now let's try a complete program *)
>> let e = Lambda("x", Multiply(Val("global"), Var("x"))) in
>> let d = New("global", Str("yes")) in
>> let c = Assign("global", Apply(e, Int(5))) in
>> let program = Prog(d,c) in
>> let result_environment, result_store = interpret' program emptyenv emptystore in
>> eval (Val("global")) result_environment result_store;;
- : evalue = EStr "yesyesyesyes"
   The most useful interpreter functions are:
       expr -> env -> store -> evalue evaluates expressions
eval:
cval: com -> env -> store -> store converts commands into a modified mutable memory
dval:
       dec -> env -> store -> env*store allows extending mutable memory
interpret: prog -> env -> store -> store combines all of the above. cannot return expression values like
```

eval, though, as we do not have print functionality

interpret': prog -> env -> store -> env*store just like interpret but return the last evaluated environment as well, which allows for further analysis of the output

emptyenv and emptystore already initialized empty environment and store

env' and store' allow extending environments and stores outside of syntax. new' combines this and allows doing something like the New(...) command outside of syntax.

ereflect and creflect and dreflect and reflect reflection of expressions, commands, declarations, full blown programs.

etaint and ctaint and dtaint and taint_anlaysis static taint analysis of expressions, commands, declarations and full blown programs.

3.1 EXPRESSIONS

Int(3) Str("hello world") Bool(true) Float(4.5) Lambda("x", <exp containing="" x="">) RecLambda("r", "x", <exp and="" containing="" f="" x="">) RecLambda("r", "x", <exp and="" containing="" f="" x="">) RecLambda("r", "x", <exp and="" containing="" f="" x="">) Rec("f", Lambda()) Proc(["x", "y", "z",], Block()) IfThenElse(Bool(true),,) Var("x") LetIn("x", et e.) Multiply(el, e2) Multiply(el, e2) Apply(el, e2) Final function, applies to: Int, Str, Float multiply function, applies to: Int, Str, Float typical function application, el is of type: Lambda, Rec like C's = like C's = Greater(el, e2) Not(e) Or(el, e2) And(el, e2) Len(Str()) Sub(Str(), i, j) Lower(Str()) Trim(Str()) Trim(Str()) basic integer basic ASCII string basic hocal typical function typical recursive function tontrol flow element typical recursive function typical recursive function this is a way to retrieve an immutable variable's content this is a way to retrieve an mutable variable's content this is a way to retrieve an mutable variable's content this is a way to retrieve an mutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way to retrieve an immutable variable's content this is a way</exp></exp></exp></exp>	EXAMPLE	DESCRIPTION
Bool(true) Float(4.5) Float(4.5) Lambda("x", <exp containing="" x="">) RecLambda("f", "x", <exp and="" containing="" f="" x="">) Rec("f", Lambda()) Froc(["x", 'y", '"z",], Block()) Froc(["x", 'y", 'z",], Block()) Froc(["x", 'y", 'y", 'z",], Block()) Froc([x", 'y", 'y", 'y", 'z",], Block()) Froc([x", 'y", 'y", 'y", 'y", 'y", 'y", 'y", 'y</exp></exp>	Int(3)	basic integer
Bloat (4.5) basic float	Str("hello world")	basic ASCII string
Lambda("x", <exp containing="" x="">) RecLambda("f", "x", <exp and="" containing="" f="" x="">) Rec("f", Lambda()) Proc(["x", "y"; "z";], Block()) If ThenElse(Bool(true),,) Var("x") LetIn("x", e1, e2) Val("x") Plus(e1, e2) Multiply(e1, e2) Multiply(e1, e2) Apply(e1, e2) Fequals(e1, e2) Greater(e1, e2) Greater(e1, e2) Not(e) Or(e1, e2) And(e1, e2) Let() And(e1, e2) Let() Sub(Str()) Sub(Str()) Sub(Str()) Trim(Str()) Trim(Str()) ytypical function typical function typical recursive function then this is a way to retrieve an immutable variable's content this is a way to retrieve a mutable variable's content tent plus function, applies to: Int, Str, Float multiply function, applies to: Int, Str, Float typical function application, e1 is of type: Lambda, RecLambda, Rec like C's == Greater(e1, e2) like C's = like C's like C's like C's like C's like C's like C's like C's && gets the length of a St sub(Str()) reduces a string to lowercase, like Python's lower() Trim(Str()) Trim(Str()) trims whitespace from a string, like Python's</exp></exp>	Bool(true)	basic boolean
Lambda("x", <exp containing="" x="">) RecLambda("f", "x", <exp and="" containing="" f="" x="">) Rec("f", Lambda()) Proc(["x", "y"; "z";], Block()) If ThenElse(Bool(true),,) Var("x") LetIn("x", e1, e2) Val("x") Plus(e1, e2) Multiply(e1, e2) Multiply(e1, e2) Apply(e1, e2) Fequals(e1, e2) Greater(e1, e2) Greater(e1, e2) Not(e) Or(e1, e2) And(e1, e2) Let() And(e1, e2) Let() Sub(Str()) Sub(Str()) Sub(Str()) Trim(Str()) Trim(Str()) ytypical function typical function typical recursive function then this is a way to retrieve an immutable variable's content this is a way to retrieve a mutable variable's content tent plus function, applies to: Int, Str, Float multiply function, applies to: Int, Str, Float typical function application, e1 is of type: Lambda, RecLambda, Rec like C's == Greater(e1, e2) like C's = like C's like C's like C's like C's like C's like C's like C's && gets the length of a St sub(Str()) reduces a string to lowercase, like Python's lower() Trim(Str()) Trim(Str()) trims whitespace from a string, like Python's</exp></exp>	Float(4.5)	basic float
RecLambda("f", "x", <exp and="" containing="" f="" x="">) Rec("f", Lambda()) Proc("x","y","z";], Block()) If thenElse(Bool(true),,) Var("x") LetIn("x", e1, e2) LetIn("x", e1, e2) This is a way to retrieve an immutable variable's content LetIn("x") LetIn("x", e1, e2) This is a way to retrieve an immutable variable's content LetIn("x") LetIn("x") LetIn("x") LetIn("x", e1, e2) This is a way to retrieve an immutable variable's content LetIn("x") LetIn("x") LetIn("x", e1, e2) This is a way to retrieve an immutable variable's content LetIn("x") LetIn("x") LetIn("x") LetIn("x", e1, e2) LetIn("x") LetIn("x") LetIn("x", e1, e2) LetIn("x") LetIn("x", e1, e2) LetIn("x") LetIn("x", e1, e2) LetIn("x") LetIn("x", e1, e2) LetIn(xtin, applies to: Int, Str, Float typical function, applies to: Int, Str, Float like C's == [like C's == [like C's == [like C's 1] And(e1, e2) Len(Str()) gets the length of a St gets a substring. i and j of type Int. Lower(Str()) Lower(Str()) Lower(Str()) Lower(Str()) Trim(Str()) Trim(Str()) Lower(Str()) Trim(Str())</exp>		typical function
Rec("f", Lambda()) Proc(["x";"y";"z";], Block()) IfThenElse(Bool(true),,) Var("x") LetIn("x", e1, e2) Val("x") Plus(e1, e2) Multiply(e1, e2) Apply(e1, e2) Greater(e1, e2) Not(e) Or(e1, e2) Not(e) Or(e1, e2) And(e1, e2) Len(Str()) Sub(Str()) Trim(Str()) Just another way to define recursive lambdas this is a procedure, check the commands section control flow element this is a way to retrieve an immutable variable's content this is a way to nest functional blocks and scopes this is a way to retrieve a mutable variable's content plus function, applies to: Int, Str, Float multiply function, applies to: Int, Str, Float multiply function application, e1 is of type: Lambda, RecLambda, Rec Equals(e1, e2) like C's == Greater(e1, e2) Not(e) Or(e1, e2) And(e1, e2) Len(Str()) gets the length of a St Sub(Str(), i, j) Lower(Str()) reduces a string to lowercase, like Python's lower() reduces a string to uppercase, like Python's upper() Trim(Str()) Trim(Str()) trims whitespace from a string, like Python's	, , ,	v =
Proc(["x";"y";"z";], Block()) IfThenElse(Bool(true),,) Var("x") LetIn("x", e1, e2) Val("x") LetIn("x", e1, e2) This is a way to retrieve an immutable variable's content this is a way to retrieve a mutable variable's content thent Plus(e1, e2) Plus(e1, e2) Multiply(e1, e2) Apply(e1, e2) Lambda, RecLiambda, RecLiambda, Rec Lambda, Recliambda, Rec Like C's = Greater(e1, e2) like C's		
IfThenElse(Bool(true),,) Control flow element		
this is a way to retrieve an immutable variable's content LetIn("x", e1, e2) Val("x") this is a way to nest functional blocks and scopes this is a way to retrieve a mutable variable's content Plus(e1, e2) Multiply(e1, e2) Apply(e1, e2) Apply(e1, e2) Equals(e1, e2) Greater(e1, e2) Not(e) Or(e1, e2) And(e1, e2) Len(Str()) Sub(Str(), i, j) Lower(Str()) Upper(Str()) Trim(Str()) this is a way to retrieve an immutable variable's content this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes the list is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes this is a way to nest functional blocks and scopes functional blocks and scopes this is a way to nest functional blocks and scopes function application, plus function application, and scopes function application, and scopes function application, and scopes function application, and scopes fun		- · · · · · · · · · · · · · · · · · · ·
LetIn("x", e1, e2) this is a way to nest functional blocks and scopes this is a way to retrieve a mutable variable's content Plus(e1, e2) plus function, applies to: Int, Str, Float multiply function, applies to: Int, Str, Float function application, e1 is of type: Lambda, RecLambda, RecLambda, Rec Apply(e1, e2) like C's == Greater(e1, e2) like C's ! Not(e) like C's! Or(e1, e2) like C's!! And(e1, e2) like C's && Len(Str()) gets the length of a St Sub(Str(), i, j) gets a substring. i and j of type Int. Lower(Str()) reduces a string to lowercase, like Python's lower() Upper(Str()) reduces a string to uppercase, like Python's upper() Trim(Str()) trims whitespace from a string, like Python's		this is a way to retrieve an immutable variable's
this is a way to retrieve a mutable variable's content Plus(e1, e2) Multiply(e1, e2) Apply(e1, e2) Apply(e1, e2) Equals(e1, e2) Bike C's == Greater(e1, e2) Not(e) Or(e1, e2) And(e1, e2) Len(Str()) Sub(Str(), i, j) Lower(Str()) Trim(Str()) this is a way to retrieve a mutable variable's content tent plus function, applies to: Int, Str, Float multiply function, applies to: Int, Str, Float function applies to: Int, Str, Float multiply function, applies to: Int, Str, Float function applies to: Int, Str, Float function, applies to: Int, Str, Float function applies to: Int, Str, Float function applies to: Int, Str, Str, Str, Str, Str, Str, Str, St	LetIn("x", e1, e2)	
tent Plus(e1, e2) plus function, applies to: Int, Str, Float Multiply(e1, e2) multiply function, applies to: Int, Str, Float Apply(e1, e2) typical function application, e1 is of type: Lambda, RecLambda, Rec Equals(e1, e2) like C's == Greater(e1, e2) like C's ! Or(e1, e2) like C's !! And(e1, e2) like C's && Len(Str()) gets the length of a St Sub(Str(), i, j) gets a substring. i and j of type Int. Lower(Str()) reduces a string to lowercase, like Python's lower() Upper(Str()) reduces a string to uppercase, like Python's upper() Trim(Str()) trims whitespace from a string, like Python's		-
Multiply(e1, e2)multiply function, appliest to: Int, Str, FloatApply(e1, e2)typical function application, e1 is of type: Lambda, RecLambda, RecEquals(e1, e2)like C's ==Greater(e1, e2)like C's !Not(e)like C's !Or(e1, e2)like C's !!And(e1, e2)like C's &&Len(Str())gets the length of a StSub(Str(), i, j)gets a substring. i and j of type Int.Lower(Str())reduces a string to lowercase, like Python'slower()reduces a string to uppercase, like Python'sUpper(Str())reduces a string to uppercase, like Python'sTrim(Str())trims whitespace from a string, like Python's		· ·
Multiply(e1, e2)multiply function, appliest to: Int, Str, FloatApply(e1, e2)typical function application, e1 is of type: Lambda, RecLambda, RecEquals(e1, e2)like C's ==Greater(e1, e2)like C's !Not(e)like C's !Or(e1, e2)like C's !!And(e1, e2)like C's &&Len(Str())gets the length of a StSub(Str(), i, j)gets a substring. i and j of type Int.Lower(Str())reduces a string to lowercase, like Python'slower()reduces a string to uppercase, like Python'sUpper(Str())reduces a string to uppercase, like Python'sTrim(Str())trims whitespace from a string, like Python's	Plus(e1, e2)	plus function, applies to: Int, Str, Float
Apply(e1, e2)typical function application, e1 is of type: Lambda, RecLambda, RecEquals(e1, e2)like C's ==Greater(e1, e2)like C's !Not(e)like C's !Or(e1, e2)like C's !!And(e1, e2)like C's &&Len(Str())gets the length of a StSub(Str(), i, j)gets a substring. i and j of type Int.Lower(Str())reduces a string to lowercase, like Python'slower()lower()Trim(Str())trims whitespace from a string, like Python's		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	= - • • • •	
Equals(e1, e2) like C's == Greater(e1, e2) like C's! Or(e1, e2) like C's And(e1, e2) like C's && Len(Str()) gets the length of a St Sub(Str(), i, j) gets a substring. i and j of type Int. Lower(Str()) reduces a string to lowercase, like Python's lower() Upper(Str()) reduces a string to uppercase, like Python's upper() Trim(Str()) trims whitespace from a string, like Python's		7
Greater(e1, e2) Not(e) Or(e1, e2) And(e1, e2) Len(Str()) Sub(Str(), i, j) Lower(Str()) Upper(Str()) Trim(Str()) like C's ! like C's ! like C's ! like C's && gets the length of a St gets a substring. i and j of type Int. reduces a string to lowercase, like Python's lower() reduces a string to uppercase, like Python's upper() trims whitespace from a string, like Python's	Equals(e1, e2)	·
Not(e) like C's! Or(e1, e2) like C's And(e1, e2) like C's && Len(Str()) gets the length of a St Sub(Str(), i, j) gets a substring. i and j of type Int. Lower(Str()) reduces a string to lowercase, like Python's lower() Upper(Str()) reduces a string to uppercase, like Python's upper() Trim(Str()) trims whitespace from a string, like Python's		
Or(e1, e2) And(e1, e2) Len(Str()) Sub(Str(), i, j) Lower(Str()) Upper(Str()) Trim(Str()) like C's && like C's && like C's && like C's && gets the length of a St gets a substring. i and j of type Int. reduces a string to lowercase, like Python's lower() reduces a string to uppercase, like Python's upper() Trim(Str())		
And(e1, e2) Len(Str()) Sub(Str(), i, j) Lower(Str()) Upper(Str()) Trim(Str()) like C's && gets the length of a St gets a substring. i and j of type Int. reduces a string to lowercase, like Python's lower() reduces a string to uppercase, like Python's upper() trims whitespace from a string, like Python's		like C's ¦¦
Len(Str()) Sub(Str(), i, j) Lower(Str()) gets the length of a St gets a substring. i and j of type Int. reduces a string to lowercase, like Python's lower() Upper(Str()) reduces a string to uppercase, like Python's upper() Trim(Str()) trims whitespace from a string, like Python's		like C's &&
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		gets the length of a St
Lower(Str()) reduces a string to lowercase, like Python's lower() Upper(Str()) reduces a string to uppercase, like Python's upper() Trim(Str()) trims whitespace from a string, like Python's		gets a substring. i and j of type Int.
Upper(Str()) reduces a string to uppercase, like Python's upper() Trim(Str()) trims whitespace from a string, like Python's		
upper() Trim(Str()) trims whitespace from a string, like Python's		lower()
Trim(Str()) trims whitespace from a string, like Python's	Upper(Str())	reduces a string to uppercase, like Python's
		upper()
	$\operatorname{Trim}(\operatorname{Str}(\dots))$	
S.trim()		s.trim()
Replace(<string be="" replace="" to="">,<replacer string="">,<string>) replaces a string with another string in a string,</string></replacer></string>	Replace(<string be="" replace="" to="">,<replacer string="">,<string>)</string></replacer></string>	replaces a string with another string in a string,
like Python's s.replace()		2 9