LinearLang

A Memory Safety Language Without Garbage Collection.

王璐璐, 朱思文, 白宗磊

June 7, 2017

Peking University

Table of contents

Motivation

Memory Error

Different types of memory errors

- dangling pointer / wild pointer
- memory leak
- double free
- ...

background

Two main approaches for memory management.

Manual memory management

- **Pros**: high performance.
- **Cons**: not safe, memory leak

Garbage collection

- **Pros**: (sumi)automatic, safe.
- **Cons**: low performance, memory leak

example1

dangling pointer

```
function writeTwoObjs(obj1, obj2){
    obj1.f()
    obj2.f()
    delete obj1
    delete obj2   //error!
}

let obj = new Object()
writeTwoObjs(obj,obj)
```

example2

memory leak

```
function sendMsg(str, addr){
    let s = new socket(addr)
    write(s,str)
    //close(s)
}

sendMsg("hello",addr1)
sendMsg("world",addr2)
```

Linear type system

linear type system

background

- based on linear logic a substructural logic proposed by Jean-Yves Girard

linear type system

- each linear variable must be used(transfer ownership/delete) exactly once
- no aliasing is possible for linear resource

related

- **C++1x**: smart pointer: unique_ptr, shared_ptr
- **rust**: ownership, move, borrow

Our Work

Based on linear type system we designed a programming language that do not have garbage collection and satisfies:

A well typed program will never have:

- dangling pointer
- memory leak

Language and typing rules

Memory Model

Two kinds of value:

- **basic value:** call by value, stored in stack, released automatically
- **resource value:** call by reference, stored in heap, released manually

stack and heap

- **stack:** list of (variable, basic value | address)
- **heap:** list of (address, resource)

Overview

judgment

$$\Gamma \vdash term : Ty$$

$$\Downarrow$$

$$\Gamma_{in} \vdash term : Ty \; ; \Gamma_{out}$$

in practice:

$$type\_of\,(term, \Gamma_{in}) \to (\,T, \Gamma_{out})$$

basic terms

```
type term =
    | Num_term       of int
    | Add_term       of term * term
    | Bool_term      of bool
    | Le_term        of term * term
    | Eq_term        of term * term
    | And_term       of term * term
    | Or_term        of term * term
    | Not_term       of term
    | Var_term       of string
    | If_term        of term * term * term
    | Lambda_term    of string * ty * term
    | App_term       of term * term
    | Fix_term       of term
    | Begin_term     of term list
    ...
```

types

```
type ty =
    | NumTy
    | BoolTy

    | LinResTy
    | LinListTy

    | UnitTy
    | ArrowTy   of ty * ty
```

basic typing rules

UnVar:

$$\Gamma_1, x : NumTy, \Gamma_2 \vdash x : NumTy; \Gamma_1, x : NumTy, \Gamma_2$$

$$\Gamma_1, x : BoolTy, \Gamma_2 \vdash x : BoolTy; \Gamma_1, x : BoolTy, \Gamma_2$$

LinVar

$$\Gamma_1, x : LinResTy, \Gamma_2 \vdash x : LinResTy; \Gamma_1, \Gamma_2$$

$$\Gamma_1, x : LinListTy, \Gamma_2 \vdash x : LinListTy; \Gamma_1, \Gamma_2$$

more typing rules

If

$$\frac{\begin{array}{c}\Gamma_1 \vdash tm1 : BoolTy \,; \Gamma_2 \\ \Gamma_2 \vdash tm2 : T \,; \Gamma_3 \\ \Gamma_2 \vdash tm3 : T \,; \Gamma_3\end{array}}{\Gamma_1 \vdash \mathit{if}\ tm1\ \mathit{then}\ tm2\ \mathit{else}\ tm3\ : T \,; \Gamma_3}$$

Abs:

$$\frac{\begin{array}{c}\Gamma_1, x : T_1 \vdash tm : T_2; \Gamma_2 \\ \Gamma_1 = \Gamma_2 \div x\end{array}}{\Gamma_1 \vdash \mathit{fun}(x : T_1)\{tm\} : T_1 \to T_2 \,; \Gamma_2 \div x}$$

two kinds of let binding

```
type term =
    ...
    | LetUn_term        of string * term * term
    | LetLin_term       of string * term * term

    | NewLinRes_term    of string
    ...
```

example

```
letUn x = 123
letLin y = newRes("aaa")
```

type error!

```
letLin x = 123
letUn y = newRes("aaa")
```

typing rules of let binding

letLin:

$$\frac{\Gamma_1 \vdash tm_1 : T_1; \Gamma_2 \quad T_1 = LinResTy \ \vee T_1 = LinListTy \quad \Gamma_2, x : T_1 \vdash tm_2 : T_2; \Gamma_3}{\Gamma_1 \vdash letUn\ x = tm_1\ \ in\ tm_2 : T_2; \Gamma_3 \div x}$$

letUn:

$$\frac{\Gamma_1 \vdash tm_1 : T_1; \Gamma_2 \quad \neg(T_1 = LinResTy \ \vee T_1 = LinListTy) \quad \Gamma_2, x : T_1 \vdash tm_2 : T_2; \Gamma_3}{\Gamma_1 \vdash letLin\ x = tm_1\ \ in\ tm_2 : T_2; \Gamma_3 \div x}$$

linear list

```
type term =
    ...
    | LinCons_term      of term * term
    | Null_term
    | Split_term        of term * string * string * term
    | FreeAtom_term     of term
    | FreeList_term     of term
    ...
```

example

```
letLin x = newRes("aaa")
letLin y = newRes("bbb")
letLin l = linCons(x,linCons(y,null))
split l as carx cdrx
freeAtom(carx)
freeList(cdrx)
```

typing rules of linear list

linCons:

$$\frac{\begin{array}{c}\Gamma_1 \vdash tm1 : T_1; \Gamma_2 \\ \Gamma_2 \vdash tm2 : T_2; \Gamma_3 \\ T_1 = LinResTy \\ bigwedge T_2 = LinListTy\end{array}}{\Gamma_1 \vdash linCons(tm1, tm2) : LinListTy\ ; \Gamma_3}$$

split:

$$\frac{\begin{array}{c}\Gamma_1 \vdash tm1 : T_1; \Gamma_2 \\ T_1 = LinListTy \\ \Gamma_2, tcar : LinResTy, tcdr : LinListTy \vdash tm2 : T_2; \Gamma_3\end{array}}{\Gamma_1 \vdash split\ tm1\ as\ tcar\ tcdr\ in\ tm2 : T_2; (\Gamma_3 \div tcar) \div tcdr}$$

freeAtom

$$\frac{\Gamma_1 \vdash tm : LinResTy\ ; \Gamma_2}{\Gamma_1 \vdash freeAtom(tm) : UnitTy\ ; \Gamma_2}$$

helpful primitive operations

```
type term =
    ...
    | CopyAtom_term     of term * string * string * term
    | CopyList_term     of term * string * string * term
    | If_null_term      of string * term * term
    | AppendList_term   of term * term
    | Print_term        of term
```

example

type error

```
letLin x = newRes("aaa")
letLin y = f(x)
letLin l = linCons(x,linCons(y,null))   //type error!
```

well typed

```
letLin x = newRes("aaa")
copyAtom x as x1 x2
letLin y = f(x1)
letLin l = linCons(x2,linCons(y,null))   //well typed!
```

typing rules

copyAtom

$$\frac{\Gamma_1 \vdash tm1 : LinResTy \,; \Gamma_2 \qquad \Gamma_2, v1 : LinResTy, v2 : LinResTy \vdash tm2 : T \,; \Gamma_3}{\Gamma_1 \vdash copyAtom\ tm1\ as\ v1\ v2\ in\ tm2 : T \,; (\Gamma_3 \div v1) \div v2}$$

IfNull:

$$\frac{\Gamma_1 \vdash var : LinListTy \,; \Gamma_2 \qquad \Gamma_2, var : LinListTy \vdash tm2 : T \,; \Gamma_3 \qquad \Gamma_2, var : LinListTy \vdash tm3 : T \,; \Gamma_3}{\Gamma_1 \vdash if\_null\ var\ then\ tm2\ else\ tm3\ : T \,; \Gamma_3}$$

Demo

reference

- Pierce B C. Types and Programming Languages.
- Pierce B C. Advanced Topics in Types and Programming Languages
- F Pottier. Wandering through linear types, capabilities, and regions
- Harvard University. CS 152: Programming Languages
- Baker H G. A "linear logic" Quicksort[M]. ACM, 1994
- Baker, Henry G. "Lively linear Lisp: "look ma, no garbage!" ." Acm Sigplan Notices 27.8(1992):89-98.

小组分工

- 查找资料: 朱思文, 白宗磊
- 语言定义: 朱思文, 白宗磊, 王璐璐
- typing rules: 白宗磊, 王璐璐
- operational semantics: 朱思文, 白宗磊
- 编程实现: 王璐璐
- 课堂报告: 王璐璐
- 书面报告: 朱思文, 白宗磊

Thank you!