

---

# Typed Memory Management via Static Capabilities

David Walker and Karl Crary and Greg Morrisett

ACM TOPLAS, July 2000

Presented by Nikhil Swamy for CMSC838z

# Region-based Memory Management

---

- Regions provide some control over memory management
  - Regions are LIFO – lexical scoping
  - Objects are allocated into regions
  - Efficient
- But there are limitations
  - Object lifetimes can be longer than needed – no prompt deallocation
  - Definitions of regions may be large – no late allocation

# Capability Language

---

- A Continuation Passing Style (CPS) language
- Provides regions with explicit allocation/deallocation
  - Memory is a map from region *names* to regions
  - A region is a map from *locations* to *values*
- Create a new region  $\nu$  using *newrgn*  $\rho, x$ 
  - Binds  $\rho$  to  $\nu$  the name of the region  
Region names are used at compile-time for type checking
  - Binds  $x$  to  $\nu$  *hdl* – handle of the region  
Region handles are used at run-time for allocation etc.
- Free a new region  $\nu$  using *freergn*  $x; (x : \nu \text{ hdl})$

# Types

---

## Three *kinds* of types

- Value Types

$\tau ::= \alpha \mid \text{int} \mid r \text{ hdl} \mid \langle \tau_i \rangle \text{ at } r \mid$

$r \text{ hdl}$  is a singleton type for a region handle

Non-word values are tuples (or function) in some region  $r$

- Memory and Region Types

$\Upsilon ::= \{l_i : \tau_i\}$  – Region Type; location:value type

$\Psi ::= \{\nu_i : \Upsilon_i\}$  – Memory Type; region names:region type

# Capabilities and Function Types

---

- Capability Type

$$C ::= 0 | \{r^\phi\} | C \oplus C | \bar{C}$$

Basically a list of live region names ... (except for that  $\phi$ ).

Only ok to access a  $\tau$  at  $r$  if  $C = C' \oplus \{r\}$

- Function Type

Like tuples function are also resident in regions

$$\tau ::= \dots | \forall[\Delta].(C, \tau_1, \dots, \tau_n) \rightarrow 0 \text{ at } r$$

$\Delta$  is the context of all bound variables (all kinds)

$C$  is capability requirement for this function.

The explicit  $\dots \rightarrow 0$  emphasizes CPS

But then implicitly assume that  $r \in C!$

# Typing Environments

---

Expressions are typed with regard to  $\Psi, \Delta, \Gamma, C$

Simple type judgement for the projection operation  $\pi$

$\pi_i$  projects the  $i$ -th element of the tuple

$$\Pi \frac{\Delta \vdash C = C' \oplus \{r\} \quad \Psi; \Delta; \Gamma \vdash v : \langle \tau_1, \dots, \tau_n \rangle \text{ at } r}{\Psi; \Delta; \Gamma; C \vdash x = \pi_i v : \tau_i}$$

Capability is an unforgeable key to access a region

# Capability – Attempt I

---

Intuitively,  $C$  is a list of all live regions

So, try to type the `newrgn` and `freergn` as :

- Create a new key when a region is allocated.

$$\text{newrgn} \frac{}{\Psi; \Delta; \Gamma; C \vdash \text{newrgn } \rho \ x : \Delta, \rho; \Gamma, x : \rho \ hdl; C \oplus \{\rho\}}$$

- Destroy the key when a region is freed

$$\text{freergn} \frac{\Psi; \Delta; \Gamma \vdash \nu : r \ hdl \quad C = C' \oplus \{r\}}{\Psi; \Delta; \Gamma; C \vdash \text{freergn } \nu : \Delta; \Gamma; C'}$$

- But region aliases are possible – which keys to destroy?

# Typing Function Calls

---

$$\text{fncall} \frac{\begin{array}{c} \Delta \vdash \nu : \forall[\alpha_1 : \kappa_1, \dots, \alpha_n : \kappa_n](C', \tau_1, \dots, \tau_m) \rightarrow 0 \text{ at } r \\ \Psi; \Delta; \Gamma \vdash c_i : \kappa_i, v_i : \tau_i \\ \Delta \vdash (C = C'' \oplus \{r\}) \leq C'[c_i/\alpha_i] \end{array}}{\Psi; \Delta; \Gamma; C \vdash \nu[c_1, \dots, c_n](v_1, \dots, v_m) :}$$

Define a subtyping relation  $\leq$  on capabilities

But in the paper they use a metavariable  $\epsilon$  and an equality relation

So  $C = \{r\} \oplus \epsilon$  is equivalent to any  $C'$  that contains  $\{r\}$

# Region Aliasing - Example

---

Suppose you have a function definition  $f$

```
f[ $\rho_1 : Rgn, \rho_2 : Rgn$ ] ( $\{\rho_1, \rho_2\}$ ,  $x:\rho_1$  hdl,  $y:int$  at  $\rho_2$ )  
let freergn x in  
let  $z = \pi_1 y$  in ...
```

And it is invoked as  $f[\rho, \rho](x, y)$

Then the projection derefs a dangling pointer

Aliasing due to region polymorphism

Solved by aliasing constraints

# Aliasing Constraints

---

Recall  $C ::= \dots | \{r^\phi\} | \dots$   $\phi \in \{+, 1\}$  is the multiplicity

- $C = C' \oplus \{r^1\}$  uniqueness constraint on region  $r$ .
- $C = C' \oplus \{r^+\}$  allows region  $r$  to be freely aliased.
- Refine the definition of  $\oplus$  so that it is *not idempotent*

$$C = C' \oplus \{r^1\} \neq C' \oplus \{r^1\} \oplus \{r^1\}$$

$$C = C' \oplus \{r^+\} = C' \oplus \{r^+\} \oplus \{r^+\}$$

# Typing Rule for Regions

---

- Allocation creates a *unique* region

$$\text{newrgn} \frac{}{\Psi; \Delta; \Gamma; C \vdash \text{newrgn } \rho \ x : \Delta, \rho; \Gamma, x : \rho \ \text{hdl}; C \oplus \{\rho^1\}}$$

- Only unique regions may be deallocated

$$\text{freergn} \frac{\Psi; \Delta; \Gamma \vdash \nu : r \ \text{hdl} \quad C = C' \oplus \{r^1\}}{\Psi; \Delta; \Gamma; C \vdash \text{freergn } \nu : \Delta; \Gamma; C'}$$

# Subtyping Capabilities

---

- With multiplicities ( $\{+, 1\}$ ) really need subtyping relation

$$C = C' \oplus \{r^1\} \leq C' \oplus \{r^+\}$$

- The function  $\mathbb{F}$  below doesn't free rgns – non-linear  $C$

$\mathbb{F}[\rho_1 : Rgn, \rho_2 : Rgn]$

$(\{\rho_1^+, \rho_2^+\}, x:\text{int at } \rho_1, y:\text{int at } \rho_2, g:(\dots) \rightarrow 0 \text{ at } \rho_1)$

let  $z = x + y$  in

$g(x+y)$

- Can be called as  $f[\rho^1, \rho^1](x, x, g)$  because

$$C = \{\rho^1\} \leq \{\rho^+\} = \{\rho^+\} \oplus \{\rho^+\}$$

# Subtyping - Example contd.

---

- This function frees a region – requires a linear capability

```
f[ $\rho_1 : Rgn, \rho_2 : Rgn$ ]  
  ( $\{\rho_1^1, \rho_2^+\}$ ,  $x:\rho_1$  hdl,  $y:int$ ,  $g:(\dots)\rightarrow 0$  at  $\rho_2$ )  
  let freergn x in  
  let z =  $\pi_1$  y in  
  g(y)
```

- Cannot be invoked as  $f[\rho^1, \rho^1](x, y, g)$  because  
 $C = \{\rho_1^1, \rho_2^+\} = \{\rho_1^1\} \oplus \{\rho_2^+\}$  cannot be unified with  
 $C' = \{\rho^1, \rho^1\} \neq \{\rho^1\} \oplus \{\rho^1\}$

# Recovering Linearity

---

- If a function  $f$  has type  $\forall[\Delta](\{\rho_1^+, \rho_2^+\}, \dots)$  how can the regions ever be freed.
  - Maybe declare the continuation as  $\forall[\Delta](\{\rho_1^1\}, \dots)$ ?
  - But then it couldn't be called from  $f$  since  $\{r^+\} \not\leq \{r^1\}$
- Bounded quantification: finally, the  $\epsilon$  context is useful
  - Allow the caller to instantiate  $\epsilon$  to  $C$  preserving multiplicity
  - The callee's capability constraint are expressed as a subtyping relation w.r.t to  $\epsilon$
  - But the continuation has access to the *original context*  $C$

# Recovering Linearity – Example

---

- Function  $f$  has type  
$$\mathbb{F}[\rho_1 : Rgn, \rho_2 : Rgn, \epsilon \leq \{\rho_1^+, \rho_2^+\}]$$
$$(\epsilon, \dots, g : (\epsilon, \dots) \rightarrow 0 \text{ at } \rho_1) \rightarrow 0 \text{ at } r$$
- $f$  can be called by  $f[\{\rho^1\}](\dots, g)$  instantiating  $\epsilon$  to  
$$\{\rho^1\} \leq \{\rho^+, \rho^+\}$$
- But in the continuation  $g$ , the capability is still precisely  
$$\{\rho^1\}$$

# Comparison to Alias Types

---

- Alias Types (Smith, Walker, Morrisett '00) very similar to this work
  - Attempts to provide per-object manual memory management
  - Capabilities there are represented within the memory type
$$\Psi ::= \{\rho \mapsto \langle int \rangle\} | \{\rho \mapsto junk\} | \{\rho \mapsto junk\}^\omega$$
  - Similar linearity constraints –  $\omega$  is non-linear
  - But does not provide a mechanism for recovering linearity – though easily added

# Alias Types contd

---

- Coarser region aliasing constraints are less restrictive on aliasing of objects
- Other mechanisms are isomorphic
  - Region polymorphism :: Location polymorphism
  - Capability subtyping :: Store polymorphism

# Summary

---

- Neat addition of manual memory management to a type-safe IR
- CPS makes things a whole lot easier ... well positioned as an intermediate language
- Powerful technique – same formalism looks like it can represent many other things
  - Generalize to lock sets, encapsulation, security ...