A typed calculus for unique access and immutability

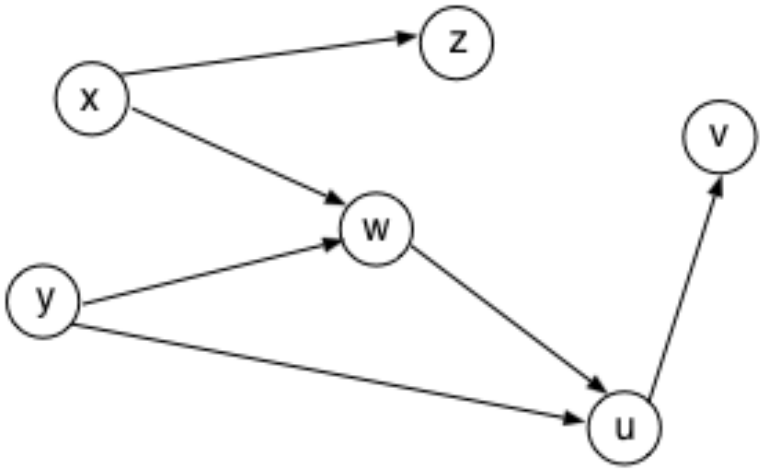
Paola Giannini[(1)], Marco Servetto[(2)], Elena Zucca[(3)]

(1) University of Piemonte Orientale
(2) Victoria University of Wellington
(3) University of Genova

TYPES 2016

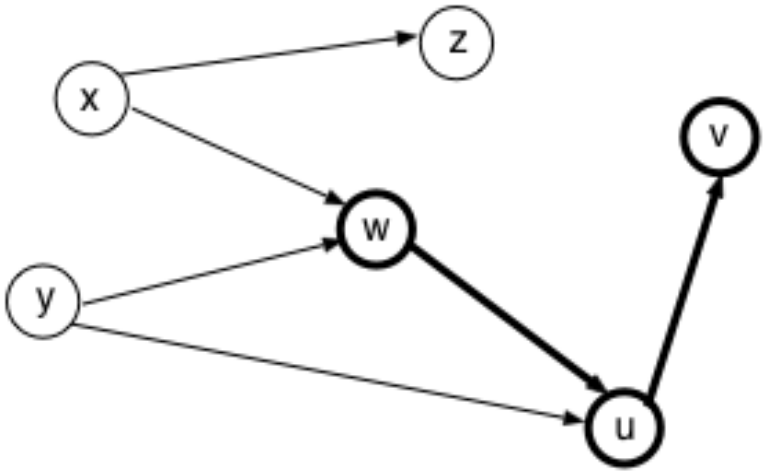
Aim

- types for expressing immutability and aliasing properties in imperative languages (e.g., object-based)
- store can be seen as a graph of references



each node contains a record of fields which are either primitive values or references to other nodes

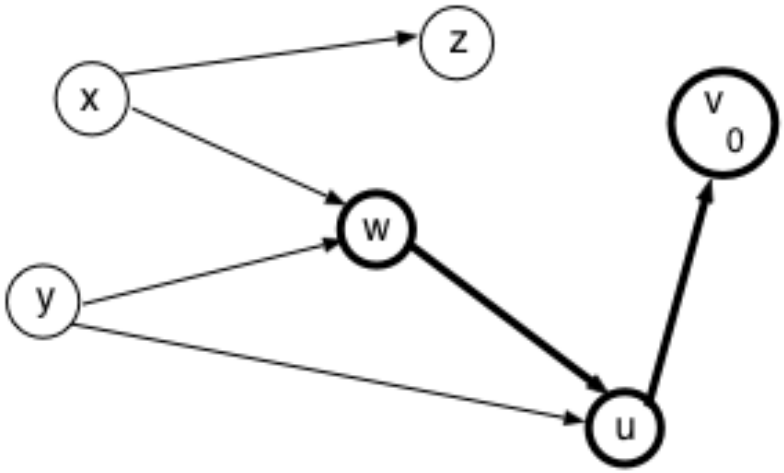
- each (expression denoting a) reference has a reachable graph

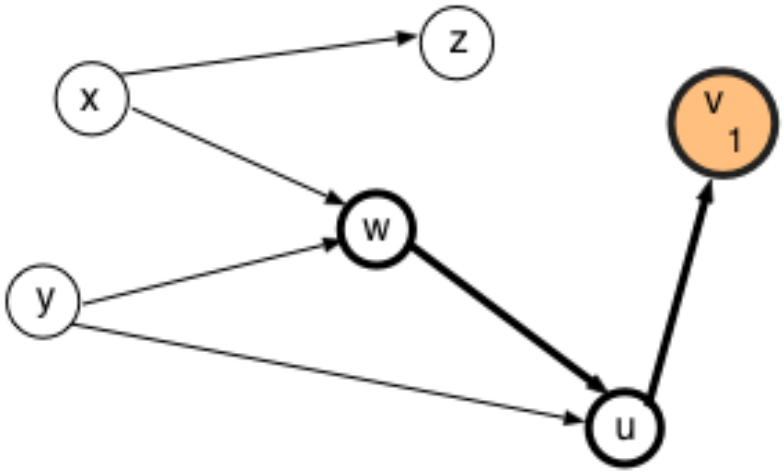


- we focus on two properties:
  - no mutation: the reachable graph cannot be modified
  - no aliasing: we cannot introduce arcs from/to the reachable graph

- four type modifiers expressing the possible combinations:
  - `mut`
    mutation, aliasing
  - `imm`
    no mutation, aliasing
  - `lent`
    mutation, no aliasing
  - `read`
    no mutation, no aliasing
- moreover: `capsule`
  isolated portion of store
  unique entry point is the reference itself

Example: no mutation





ok: w `mut`, `lent`

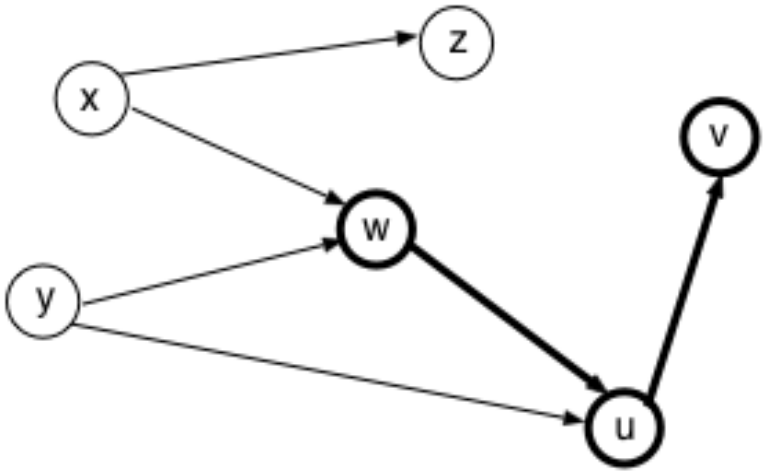
no: w `imm`, `read`

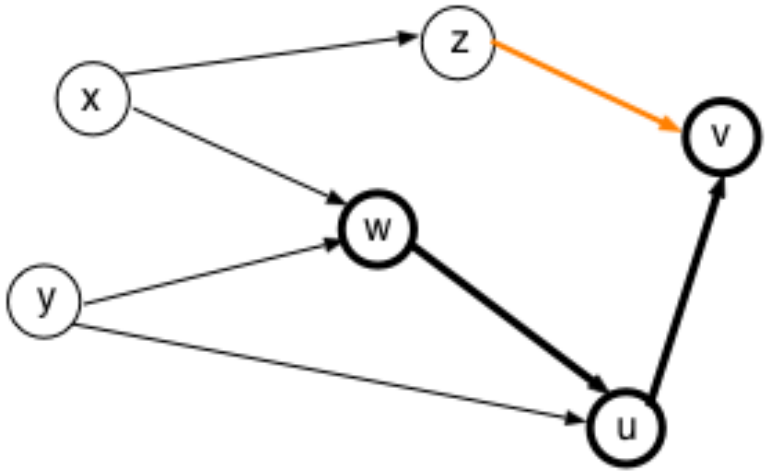
no mutation is

a constraint: we cannot mutate v through w

a guarantee: we can assume that nobody else can mutate v

Example: no aliasing





ok: w `mut`, `imm`

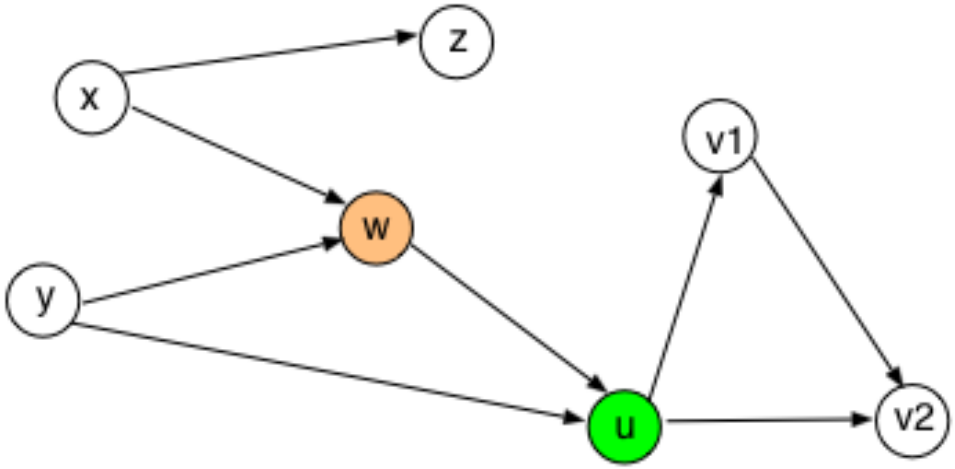
no: w `lent`, `read`

no aliasing is only a constraint:

we cannot introduce an alias to v through w

no guarantee on somebody else

Example: capsule



isolated portion of store

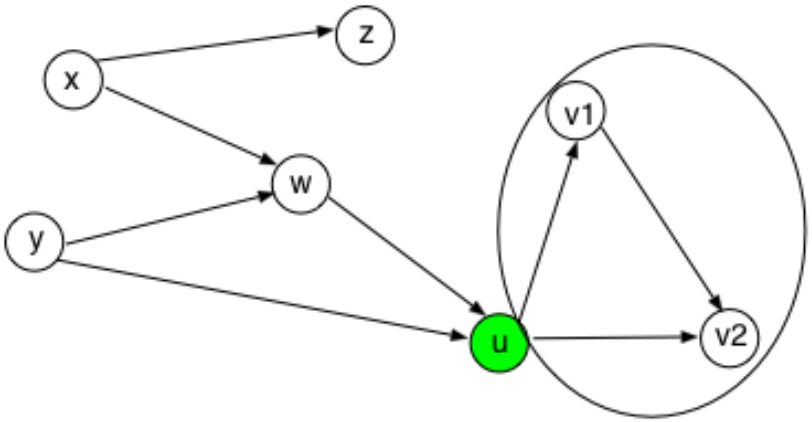
unique entry point is the reference itself

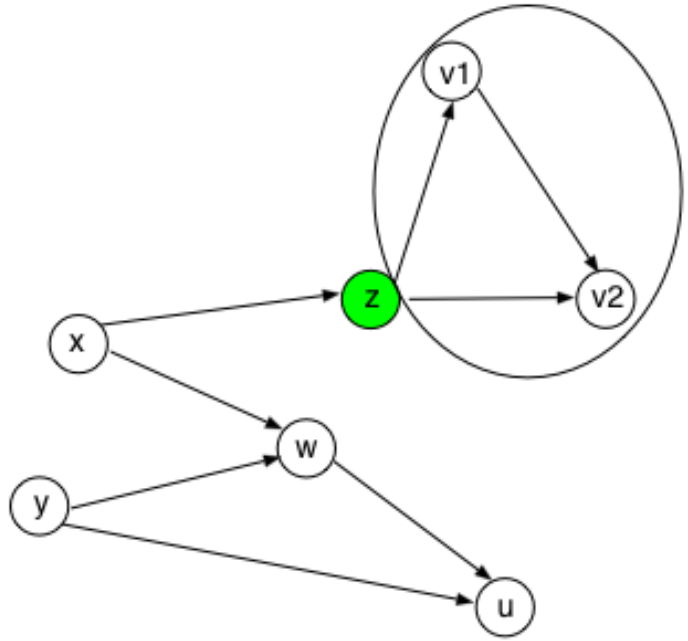
w is not a capsule

u is a capsule

Example: capsule

capsules can be safely "moved", that is, assigned to both mutable and immutable references





Concepts already proposed in literature

P. Almeida. Balloon types: Controlling sharing of state in data types. ECOOP'97

J. Boyland. Semantics of fractional permissions with nesting. TOPLAS 32(6), 2010.

D. Clarke, T. Wrigstad. External uniqueness is unique enough. ECOOP'03

C.S. Gordon, M.J. Parkinson, J. Parsons, A. Bromeld, J. Duy. Uniqueness and reference immutability for safe parallelism. OOPSLA'12

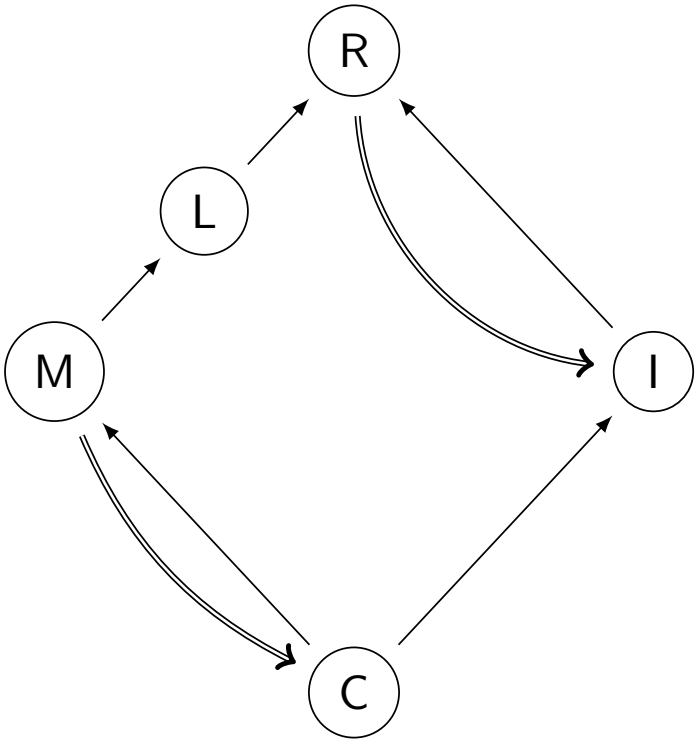
J. Hogg. Islands: Aliasing protection in object-oriented languages. OOPSLA'91

K. Naden, R. Bocchino, J. Aldrich, K. Bierho. A type system for borrowing permissions. POPL'12.

Novelties

1. integration of concepts
2. expressivity enhanced by promotion rules
   - an expression can be promoted to a more specific type provided that external references are used in a restricted way
3. execution model as pure calculus [only shown by examples]
   - no memory, just rewriting source code
   - object graphs are represented at the syntactic level
   - allows simpler statement and proof of propoerties

Subtyping hierarchy and promotions



M Mutable: alias, write

I Immutable: alias, no write

C Capsule: unique access
Reference used only once

L Lent: no alias, write

R Readable: no alias, no write

Subtype

Promotion

Syntax

convention: ds is a sequence of d

Java-like flavour is matter of taste

cd	$::=$	$\texttt{class}\ C\ \{fds\ mds\}$	class declaration
fd	$::=$	$C\ f$	field declaration
md	$::=$	$T\ m\ \mu\ (T_1\ x_1, \ldots, T_n\ x_n)\ \{\texttt{return}\ e\}$	method declaration
e	$::=$	$x \mid e.f \mid e.m(es) \mid e.f{=}e' \mid \texttt{new}\ C(es) \mid \{ds\ e\}$	expression
d	$::=$	$T\,x{=}e$	variable declaration
T	$::=$	$\mu\ C \mid \texttt{int}$	type
μ	$::=$	$\texttt{imm} \mid \texttt{mut} \mid \texttt{capsule} \mid \texttt{lent} \mid \texttt{read}$	type modifier

Type system

simplified version: only capsule promotion

Typing judgment

T	$::=$	$\mu\, C \mid \texttt{int}$	type
μ	$::=$	$\texttt{imm} \mid \texttt{mut} \mid \texttt{capsule} \mid \texttt{lent} \mid \texttt{read}$	type modifier
Δ	$::=$	$\Gamma; xss$	type context
Γ	$::=$	$x_1 : T_1 \ldots x_n : T_n$	type assignment
xss	$::=$	$xs_1 \ldots xs_n$	lent-restricted variables

$$\Gamma; xs_1 \ldots xs_n \vdash e : T$$

Typing judgment

$$\Gamma; xss \vdash e : T$$

variables which are mutable in Γ are partitioned in $n+1$ groups:

$xss = xs_1 \ldots xs_n$ = lent-restricted variables = can only be used as `lent`

$xs_0 = \mathrm{dom}^{\mathtt{mut}}(\Gamma) \backslash xss$ = unrestricted mutable variables

no aliasing is introduced among (portions of store reachable from) $xs_0, xs_1, \ldots, xs_n$

Typing rules (1)

a group of lent-restricted variables is introduced by *promotion* rule

$$(\text{T-PROM})\ \frac{\Gamma; xss\ xs \vdash e : C}{\Gamma; xss \vdash e : \texttt{capsule}\ C}\ xs = \text{dom}^{\texttt{mut}}(\Gamma) \setminus xss$$

an expression can be promoted to `capsule` if all external references are only used as `lent`

xs = currently unrestricted mutable variables which become lent-restricted

Typing rules (2)

a group can become unrestricted by swapping

$$(\text{T-SWAP})\ \frac{\Gamma; xss\ xs' \vdash e : \mu\ C}{\Gamma; xss\ xs \vdash e : \mu'\ C} \qquad xs' = \mathsf{dom}^{\mathtt{mut}}(\Gamma) \backslash (xss\ xs)$$

$$\mu' = \begin{cases} \mathtt{lent} & \text{if } \mu = \epsilon \\ \mu & \text{otherwise} \end{cases}$$

xs = lent-restricted variables which become available

xs' = currently unrestricted mutable variables which become lent-restricted

Example: capsule promotion

a `capsule` uses external references only as `lent`

```
D z= new D(0)
capsule C x= {
  D y= new D(z.f+1)
  new C(y,y) }
x
```

$\longrightarrow^{\star}$

```
D z= new D(0)
capsule C x= {
  D y= new D(1)
  new C(y,y) }
x
```

Counterexample

```
D z= new D(0)
capsule C x= {   //ill-typed
  D y= z
  new C(y,y) }
x
```

$\longrightarrow$

```
D z= new D(0)
C x= new C(z,z)
x
```

Example: swapping

How to modify (the object denoted by) a `lent` reference?

```
lent D z= new D(0)
z.f=z.f+1
```

the singleton group `z` is swapped with the empty set

Example: swapping to achieve promotion

```
D z= new D(0)
capsule C x= (
  D y= new D(z.f=z.f+1)
  new C(y,y) )
x
```

$\longrightarrow^{\star}$

```
D z= new D(1)
capsule C x= (
  D y= new D(1)
  new C(y,y) )
x
```

Typing rules (3)

$$(\text{T-SUB})\ \frac{\Delta \vdash e : T}{\Delta \vdash e : T'}\ T \leq T'$$

$$(\text{T-VAR})\ \frac{\Gamma(x) = \mu\, C}{\Gamma; xss \vdash x : \mu'\, C}\quad \mu' = \begin{cases} \texttt{lent} & \text{if } x \in xss \\ \mu & \text{otherwise} \end{cases}$$

Typing rules (4)

$$(\text{T-FIELD-ACCESS})\ \frac{\Delta \vdash e : \mu\, C}{\Delta \vdash e.f : \mu\, C_i} \quad \begin{array}{l}\mathsf{fields}(C) = C_1\ f_1 \ldots C_n\ f_n \\ f = f_i\end{array}$$

$$(\text{T-METH-CALL})\ \frac{\Delta \vdash e_i : T_i \quad \forall i \in 0..n}{\Delta \vdash e_0.m(e_1, \ldots, e_n) : T} \quad \begin{array}{l}T_0 = \mu\, C \\ \mathsf{mtype}(C, m) = \langle T, \mu, T_1 \ldots T_n \rangle\end{array}$$

$$(\text{T-FIELD-ASSIGN})\ \frac{\Delta \vdash e : C \quad \Delta \vdash e' : C_i}{\Delta \vdash e.f{=}e' : C_i} \quad \begin{array}{l}\mathsf{fields}(C) = C_1\ f_1 \ldots C_n\ f_n \\ f = f_i\end{array}$$

$$(\text{T-NEW})\ \frac{\Delta \vdash e_i : C_i \quad \forall i \in 1..n}{\Delta \vdash \texttt{new}\ C(e_1, \ldots, e_n) : C} \quad \mathsf{fields}(C) = C_1\ f_1 \ldots C_n\ f_n$$

$$(\text{T-BLOCK})\ \frac{\Gamma[\Gamma'];\, xss \vdash e_i : T_i\ \forall i \in 1..n \quad \Gamma[\Gamma'];\, xss \vdash e : T}{\Gamma;\, xss \vdash \{\, T_1\, x_1 = e_1 \ldots T_n\, x_n = e_n\ e\} : T} \quad \Gamma' = x_1{:}T_1 \ldots x_n{:}T_n$$

Results

- Soundness
  If $\vdash e$, and $e \longrightarrow^\star e'$, then either e' is a value, or $e' \longrightarrow$

- Modifiers have the expected behaviour, e.g.
  a `capsule` expression reduces to a closed value

  If $\vdash \mathcal{E}[e]$, $\Gamma = \mathsf{typectx}(\mathcal{E})$,
  $\Gamma; \emptyset \vdash e : \texttt{capsule}\ C$, and $\mathcal{E}[e] \longrightarrow^\star \mathcal{E}'[v]$,
  then v is closed

Conclusion

Key contributions:

- powerful type system for tracing mutation and aliasing
- non standard operational model of imperative features as a pure calculus: properties of modifiers are expressed on terms
- part of the design of the novel language L42, aimed at massive use of libraries
  `L42.is`
- long term goal: Hoare-like logic for the model

Thanks