Regions and Permissions for Data Invariants

Romain Bardou and Claude Marché

Septembre 2009

Motivation

preservation of data invariants in pointer programs

- ownership system of Spec# [Barnett et al 04]

static typing instead of theorem provers

- Universe Types [Dietl, Müller 05]

how?

- regions [Tofte, Talpin, Jouvelot 91] ... [Banerjee et al 08]
- with permissions [Crary et al 99]

Data Invariant Example

```
class PosInt {
  int value;
  //@ invariant this.value > 0;

  void double() {
    value := value + value;
  }
}
```

Core Language

functional style with references ($e_1 := e_2$, $!e$)

type *PosInt* =
  int
  inv(*this*) = $!this > 0$
end

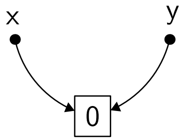
val *double*(*x*: *PosInt*): unit =
  $x := !x + !x$

focus on pointers and aliasing
ignore inheritance and dynamic dispatch

Problem: Pointer Aliasing

```
val f(x: PosInt, y: PosInt): unit =
  x := 0;
  x := 1 / !y
```

what if $x = y$?

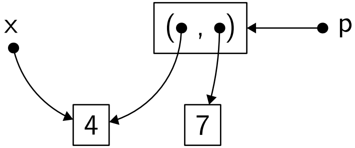


Problem: Components

type *SortedPair* $=$
PosInt $\times$ *PosInt*
inv(*this*) $= !this.1 < !this.2$
end

val *double*(x: *PosInt*): unit $=$
$x := !x + !x$

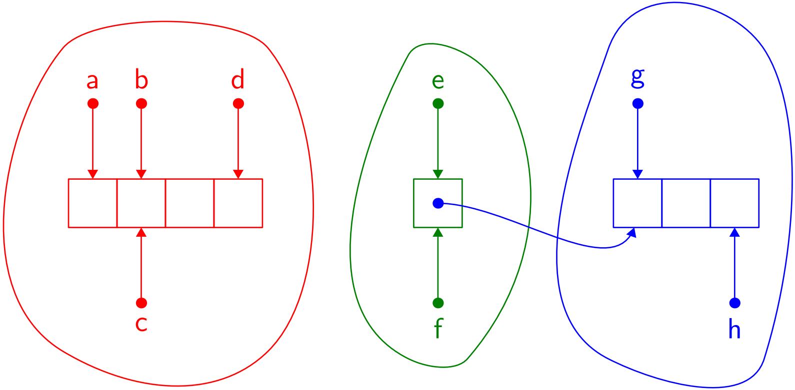
what if x is member of a *SortedPair* p?



Regions

solution: group pointers by regions

pointers of two different regions may not be aliased



Permissions

permission = static linear information about a region

"linear" means:

- permissions cannot be duplicated
- permissions depend on the program point
- operations may consume some permissions
- operations may produce other permissions

Empty Regions

regions are created empty

region ρ **in**

this produces permission $\rho^{\emptyset}$: "ρ is empty"

Allocation and Singleton Regions

pointers are allocated in empty regions

new *PosInt*$[\rho]$

this:

- consumes permission $\rho^{\emptyset}$
- produces permission ρ^{S}: "ρ is singleton"

region ρ is no longer empty: it is singleton

Group Regions

a singleton region ρ may be demoted to a group region

this is implicit

this:

- consumes permission ρ^S
- produces permission ρ^G: "ρ is group"

Adoption

adoption moves a pointer from a singleton region to an already-existing group region

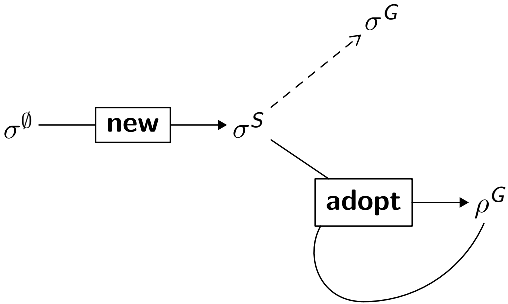
if x is in region σ:

adopt x **in** ρ

this:

- consumes permissions σ^S and ρ^G
- produces permission ρ^G

The Permission Diagram (so far)



Permissions for Invariants

use permissions to denote whether invariants hold

- $\rho^{\emptyset}$: empty region, no invariant
- ρ°: open singleton region, invariant does not hold
- $\rho^{\times}$: closed singleton region, invariant holds
- ρ^{G}: group region, all invariants hold

only pointers in open regions can be assigned

Packing and Unpacking

pack x

packing a pointer of ρ:

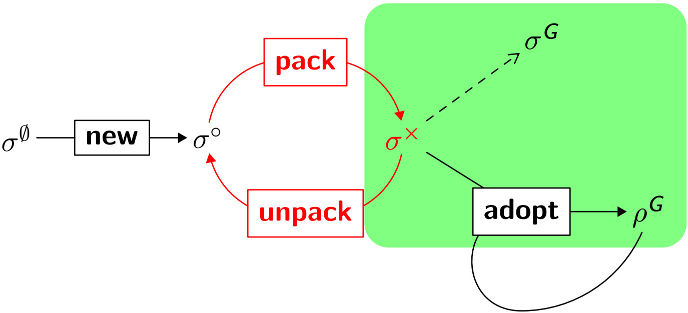
- consumes ρ°
- produces $\rho^{\times}$
- generates a proof obligation (the invariant)

unpack x

unpacking is the opposite operation:

- consumes $\rho^{\times}$
- produces ρ°

The Permission Diagram (with packing)

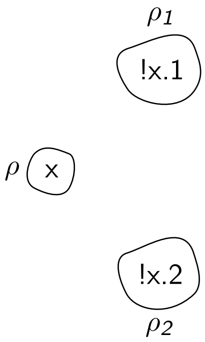


Owned Regions

problem: invariants about other pointers?

$\textbf{type}\ \mathit{SortedPair}\ \langle \rho_1, \rho_2 \rangle =$
$\quad \mathit{PosInt}[\rho_1] \times \mathit{PosInt}[\rho_2]$
$\quad \textbf{inv}(\mathit{this}) = !\mathit{this}.1 < !\mathit{this}.2$
$\textbf{end}$

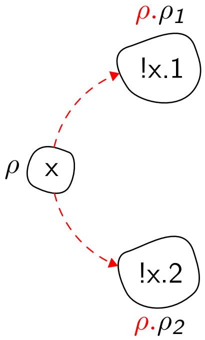
$\textbf{val}\ \mathit{bad}(x\colon \mathit{SortedPair}\langle \rho_1, \rho_2 \rangle[\rho])$
$\quad \textbf{consumes}\ \rho^{\times}, \rho_1{}^{\circ}, \rho_2{}^{\circ}$
$\quad \textbf{produces}\ \rho^{\times}, \rho_1{}^{\circ}, \rho_2{}^{\circ} =$
$\quad\quad !x.1 := 69;$
$\quad\quad !x.2 := 42$



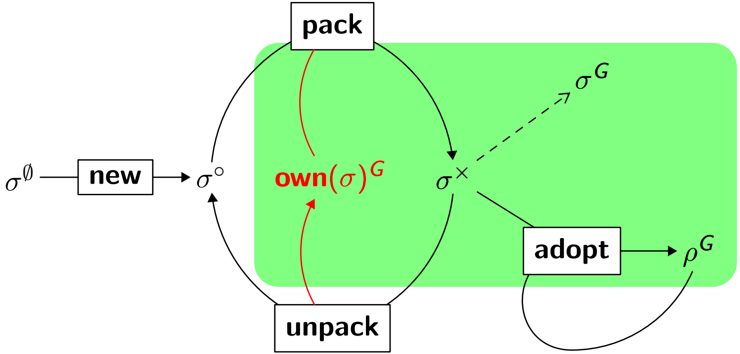
Owned Regions

solution: owned regions

type *SortedPair* =
  own ρ_1, ρ_2
  $PosInt[\rho_1] \times PosInt[\rho_2]$
  inv(*this*) = $!this.1 < !this.2$
end

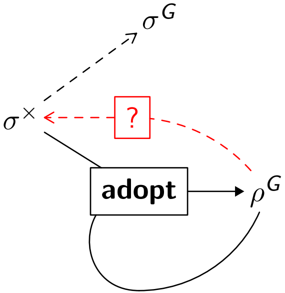


The Permission Diagram (with owned regions)



Group to Singleton?

problem: how to modify a pointer of a group region?



Group to Singleton?

solution: extract the pointer to a singleton region

problem: what happens to the group region?

- what if several pointers are extracted?
- what if a pointer is extracted several times?

solution: group region temporarily disabled

Linear Implication

$$\sigma \multimap \rho$$

ρ is disabled temporarily

$\sigma^{\times}$ must be given to enable ρ

allows temporary extraction from ρ to σ

Focus

if y in region ρ:

focus y **in** σ

this:

- consumes $\sigma^{\emptyset}$ and ρ^{G}
- produces $\sigma^{\times}$ and $\sigma \multimap \rho$

region σ now also contains y

Unfocus

if y in region σ:

unfocus y **in** ρ

this:

- consumes $\sigma^\times$ and $\sigma \multimap \rho$
- produces ρ^G

region σ is disabled definitely

Focus and Unfocus Usage

if x in group region ρ:

region σ **in**	$\{\ \sigma^{\emptyset},\ \rho^{G}\ \}$
let $x_f = ($**focus** x **in** $\sigma)$ **in**	$\{\ \sigma^{\times},\ \sigma \multimap \rho\ \}$
unpack x_f;	$\{\ \sigma^{\circ},\ \sigma \multimap \rho\ \}$
$x_f := \cdots;$	$\{\ \sigma^{\circ},\ \sigma \multimap \rho\ \}$
pack x_f;	$\{\ \sigma^{\times},\ \sigma \multimap \rho\ \}$
unfocus x_f **in** ρ	$\{\ \rho^{G}\ \}$

$x = x_f$, but:

- x is in ρ
- x_f is in σ

Soundness

Definition

heap is coherent w.r.t. $\bar{\Sigma}$:

- invariants of closed pointers hold
- ...

Theorem

If:

- e is well-typed w.r.t. types, regions, permissions
  - when given permissions $\bar{\Sigma}$, e gives back $\bar{\Sigma}'$
- e and heap $\mathcal{H}$ reduce to e' and $\mathcal{H}'$
- $\mathcal{H}$ is coherent w.r.t. $\bar{\Sigma}$

then:

- $\mathcal{H}'$ is coherent w.r.t. $\bar{\Sigma}'$

Conclusion

static type system with regions and permissions

guarantees invariant preservation

- only VCs: invariants, when packing

ownership at the level of regions

can handle examples such as observer pattern

can handle some form of abstraction

- owned regions can be hidden

Need for Inference

inference of region annotations

$$
\begin{array}{l}
\textbf{val } f(): \mathit{PosInt}[\rho] = \\
\quad \textbf{region } \sigma \textbf{ in} \\
\quad \textbf{let } x = \textbf{new } \mathit{PosInt}[\sigma] \textbf{ in} \\
\quad x := 5; \\
\quad \textbf{pack } x; \\
\quad \textbf{let } x = (\textbf{adopt } x \textbf{ in } \rho) \textbf{ in} \\
\quad \textbf{region } \sigma_y \textbf{ in} \\
\quad \textbf{let } y = (\textbf{focus } x \textbf{ in } \sigma_y) \textbf{ in} \\
\quad \textbf{unpack } y; \\
\quad y := 7; \\
\quad \textbf{pack } y; \\
\quad \textbf{unfocus } y \textbf{ in } \rho; \\
\quad y
\end{array}
$$

$$
\begin{array}{l}
\textbf{val } f(): \mathit{PosInt} = \\
\quad \textbf{let } x = \textbf{new } \mathit{PosInt} \textbf{ in} \\
\quad x := 5; \\
\quad x := 7; \\
\quad x
\end{array}
$$

Future Works

more powerful abstraction using refinement approaches

inference

- current direction: given function prototypes and **focus** annotations, infer remaining annotations