Typed regions

Stefan Monnier

Université de Montréal
Introduction

Formal methods on the rise: language-based security

Maturity: bugs are more than an annoyance

Verify more properties than memory safety:
Java brought memory safety to the masses, now they want more

Type check low-level code: distrust tricky code

Type systems are oblivious to side-effects
Contributions

A new type system:

• Hybrid: regions $\otimes$ alias types $\otimes$ proofs

• Strong update with arbitrary aliasing

• Manipulate and reason about arbitrary state properties

• Modularity of type systems and power of Hoare logic

• Able to type check a realistic GC

• With a better replacement for widen
Regions

A region holds multiple objects, deallocated all at once

Every allocation specifies the region: \(\text{put}[\rho] \, v\)

A new type for pointers: \(\tau \text{ at } \rho\)

\[
\text{(types)} \quad \tau ::= t \mid \text{int} \mid \tau \times \tau \mid \tau \text{ at } \rho \\
| \forall [\vec{t}] \{\vec{\rho}\} (\vec{\tau}) \rightarrow 0
\]

If a region does not appear in an object’s type, it is not needed

\text{free } \rho \text{ can now be checked for safety}

Simple and efficient

Typed regions
**Alias types**

A pointer to address $\ell$ has type: $\text{ptr } \ell$

(types) $\tau ::= t \mid \text{int} \mid \tau \times \tau \mid \text{ptr } \ell$

$$\mid \forall[\bar{t}\{\ell \mapsto \tau}\{\bar{\tau}\}] \rightarrow 0$$

The heap has its own, separately maintained type. For example:

$$\{\ell_1 \mapsto (\text{int, int}), \ell_2 \mapsto (\text{ptr } \ell_1, \text{ptr } \ell_2)\}$$

Dangling pointers like $\text{ptr } \ell_3$ are allowed but unusable

$$p: \text{ptr } \ell = \text{new } 2; \quad \{\ell \mapsto (\top, \top), \ldots\}$$

$$p.0 := 1; \quad \{\ell \mapsto (\text{int, } \top), \ldots\}$$

$$p.1 := p; \quad \{\ell \mapsto (\text{int, ptr } \ell), \ldots\}$$

Low-level, very powerful, but restrictive
A fundamental principle of type soundness

A memory location cannot have 2 types at the same time

Enforced in the following ways:

regions  The type of a location is immutable
         \(\implies\) can be copied freely

alias types  The type of a location is never copied
            \(\implies\) can be changed at any time

Very few exceptions

<table>
<thead>
<tr>
<th></th>
<th>intuitionistic value</th>
<th>linear state</th>
</tr>
</thead>
<tbody>
<tr>
<td>regions</td>
<td>(\tau) at (\rho)</td>
<td>({\bar{\rho}})</td>
</tr>
<tr>
<td>alias types</td>
<td>ptr (\ell)</td>
<td>({\ell \mapsto \tau})</td>
</tr>
</tbody>
</table>
Typed regions

A hybrid between simple regions and alias types:

Part immutable/copiable and part mutable/centralized

\[
\text{types} \; \tau ::= t \mid \text{int} \mid \tau \times \tau \mid \tau \text{ at } \rho.n \\
\mid \forall \vec{t} \{ \rho \mapsto \phi \}(\vec{\tau}) \rightarrow 0
\]

Pointers have type \( \tau \text{ at } \rho.n \): the \( n^{th} \) object in \( \rho \), of intended type \( \tau \)

Every region has its own type \( \phi \), maintained separately

The intended type of a location is not necessarily its actual type

The actual type of a location depends on the region’s type
The type of a region

A region’s type is a type function of 2 parameters: \( n \) and \( \tau \)

\[
\begin{align*}
\text{fun } \text{plain } n t &= t \\
\text{fun } \text{alias } n t &= \text{if } n = 1 \text{ then } (\text{int}, \text{int}) \text{ else } \bot
\end{align*}
\]

A region of type \textit{plain} is like a traditional region:

\[
\exists n.\tau \text{ at } \rho.n \simeq \tau \text{ at } \rho
\]

\textit{alias} corresponds to alias types:

\[
\exists t.t \text{ at } \rho.n \simeq \text{ptr } \rho.n
\]

The type language is the calculus of inductive constructions, a powerful \( \lambda \)-calculus.
Why?

Copy/create arbitrary cycles: no base case, unknown aliasing

Alias types are too restrictive: pointer reversal with unknown aliasing

Expose memory layout: scanning a region

\[
\text{next : } \tau \text{ at } \rho.n \rightarrow \exists t.t \text{ at } \rho.(n+1)
\]

Used by all forms of GC

Clean up \textit{widen}