

Type Inference Rules For Container Types in CCL

*The paper is written under the direction and guidance of Dr. Alan K. Zaring (akzaring@luther.edu).

David Oniani
Luther College
oniada01@luther.edu

Last Updated: November 10, 2019

Abstract

We present the type inference rules introducing the notion of container types in the CCL programming language. This redesign required a number of substantial changes to the major aspects of the language, including the type system, the syntax, and the definitional semantics.

The $<$: Relation

$$\frac{}{\mathcal{Box}(\mathit{Triv}) <: \mathcal{Box}(\mathit{Triv})} \quad (1)$$

$$\frac{}{\mathcal{Box}(\mathit{Int}) <: \mathcal{Box}(\mathit{Int})} \quad (2)$$

$$\frac{\mathcal{T} <: \mathcal{Box}(\mathcal{U})}{\mathcal{T} <: \mathit{Immut}(\mathcal{Box}(\mathcal{U}))} \quad (3)$$

$$\frac{\mathcal{Box}(\mathcal{T}) <: \mathit{Immut}(\mathcal{U})}{\mathit{Immut}(\mathcal{Box}(\mathcal{T})) <: \mathit{Immut}(\mathcal{U})} \quad (4)$$

$$\frac{\mathcal{T} <: \mathcal{U} \quad \mathcal{Box}^?(\mathcal{T})}{\mathit{Ref} \ \mathcal{T} <: \mathit{Ref} \ \mathcal{U}} \quad (5)$$

$\mathcal{Box}^?$ statement checks for both mutable and immutable containers. In other words, $\mathcal{Box}(\mathit{Int})$ and $\mathit{Immut}(\mathcal{Box}(\mathit{Int}))$ would both satisfy the $\mathcal{Box}^?$ condition.

The $:$ and $::$ Relations

$$\frac{\text{triv } x}{x : \text{Triv}} \quad (6)$$

$$\frac{\text{triv } x}{x :: \text{Box}(\text{Triv})} \quad (7)$$

$$\frac{\text{int } x}{x : \text{Int}} \quad (8)$$

$$\frac{\text{int } x}{x :: \text{Box}(\text{Int})} \quad (9)$$

$$\frac{x :: \text{Box}(T)}{\text{immut } x :: \text{Immut}(\text{Box}(T))} \quad (10)$$

$$\frac{x :: T}{\text{ref } x :: \text{Box}(\text{Ref}(T))} \quad (11)$$

$$\frac{x :: T}{\& x : \text{Ref}(T)} \quad (12)$$

$$\frac{x : \text{Ref}(T)}{x @ : T} \quad (13)$$

$$\frac{x : \text{Ref}(T)}{x @ :: T} \quad (14)$$

$$\frac{x :: T \quad y :: \mathcal{U} \quad T <: \mathcal{U} \quad x : \mathcal{V}}{x := y : \mathcal{V}} \quad (15)$$

$x : T$ is read as “expression x is of type T and is in an *r-context*.”

$x :: T$ is read as “variable x is of type T and is in an *l-context*.”

The operators could also be referred to as the “r-type of” and “l-type of” operators.

l-context denotes everything that is *assignable* (indicated as a storable memory). r-context, on the other hand, denotes everything that is *expressible* (can be produced by an expression).

There is no r-value (e.g. expression) of the type $\text{Box}(T)$.

We omit rules for *Con* types as they only operate on r-values.

We omit rules for *Fun* types as they only accept r-values. Any variable and/or primitive type has both r-value and l-value (when it comes to primitive types, only r-value). In all cases, the r-value part of the actual parameter is passed when the function is being called.

Resulting Relationships (A Short List)

<code>int i</code>	$\rightarrow i$	$\rightarrow \mathcal{Box}(Int)$
<code>immut int ii</code>	$\rightarrow ii$	$\rightarrow \mathcal{Immut}(\mathcal{Box}(Int))$
<code>ref int ri</code>	$\rightarrow ri$	$\rightarrow \mathcal{Box}(\mathcal{Ref}(\mathcal{Box}(Int)))$
<code>immut ref int iri</code>	$\rightarrow iri$	$\rightarrow \mathcal{Immut}(\mathcal{Box}(\mathcal{Ref}(\mathcal{Box}(Int))))$
<code>ref immut int rii</code>	$\rightarrow rii$	$\rightarrow \mathcal{Box}(\mathcal{Ref}((\mathcal{Immut}(\mathcal{Box}(Int)))))$
<code>immut ref immut int irii</code>	$\rightarrow irii$	$\rightarrow \mathcal{Immut}(\mathcal{Box}(\mathcal{Ref}((\mathcal{Immut}(\mathcal{Box}(Int)))))$

Type $\mathcal{Immut} \mathcal{Box}(\mathcal{Immut} \mathcal{Box}(Int))$ cannot exist. Nested \mathcal{Box} types are only possible when there is at least one \mathcal{Ref} type.

$$\begin{aligned}
 \mathcal{Box}(Triv) &<: \mathcal{Box}(Triv) \\
 \mathcal{Box}(Int) &<: \mathcal{Box}(Int) \\
 \mathcal{Box}(Triv) &<: \mathcal{Immut}(\mathcal{Box}(Triv)) \\
 \mathcal{Box}(Int) &<: \mathcal{Immut}(\mathcal{Box}(Int)) \\
 \mathcal{Immut}(\mathcal{Box}(Triv)) &<: \mathcal{Immut}(\mathcal{Box}(Triv)) \\
 \mathcal{Immut}(\mathcal{Box}(Int)) &<: \mathcal{Immut}(\mathcal{Box}(Int))
 \end{aligned}$$

All the rules above should work with \mathcal{Ref} types in the similar manner:

$$\begin{aligned}
 \mathcal{Ref}(\mathcal{Box}(Triv)) &<: \mathcal{Ref}(\mathcal{Box}(Triv)) \\
 \mathcal{Ref}(\mathcal{Box}(Int)) &<: \mathcal{Ref}(\mathcal{Box}(Int)) \\
 \mathcal{Ref}(\mathcal{Box}(Triv)) &<: \mathcal{Ref}(\mathcal{Immut}(\mathcal{Box}(Triv))) \\
 \mathcal{Ref}(\mathcal{Box}(Int)) &<: \mathcal{Ref}(\mathcal{Immut}(\mathcal{Box}(Int))) \\
 \mathcal{Ref}(\mathcal{Immut}(\mathcal{Box}(Triv))) &<: \mathcal{Ref}(\mathcal{Immut}(\mathcal{Box}(Triv))) \\
 \mathcal{Ref}(\mathcal{Immut}(\mathcal{Box}(Int))) &<: \mathcal{Ref}(\mathcal{Immut}(\mathcal{Box}(Int)))
 \end{aligned}$$