Type Inference Rules For Container Types in CCL

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

David Oniani
Luther College
oniada01@luther.edu

Last Updated: November 24, 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

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

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

$$\frac{\mathcal{T} <:: \mathcal{Box}(\mathcal{U})}{\mathcal{T} <:: \mathcal{IBox}(\mathcal{U})} \tag{3}$$

$$\frac{\mathcal{Box}(\mathcal{T}) <:: \mathcal{IBox}(\mathcal{U})}{\mathcal{IBox}(\mathcal{T}) <:: \mathcal{IBox}(\mathcal{U})} \tag{4}$$

$$\frac{\mathcal{T} <:: \mathcal{U}}{\mathcal{Ref}\ \ \mathcal{T} <: \mathcal{Ref}\ \ \mathcal{U}} \tag{5}$$

<:: operator specifies the subtype relationship between two container types.

$\mathcal{T} <:: \mathcal{U}$ is read as "container type $\mathcal{T}$ is a subtype of the container type $\mathcal{U}$."

The : and :: Relations

$$\frac{\texttt{triv x}}{\mathsf{x} : \mathcal{Triv}} \tag{6}$$

$$\frac{\texttt{triv x}}{\mathsf{x} :: \mathcal{Box}(\mathcal{Triv})} \tag{7}$$

$$\frac{\texttt{int x}}{\mathsf{x} : \mathcal{Int}} \tag{8}$$

$$\frac{\texttt{int x}}{\mathsf{x} :: \mathcal{Box}(\mathcal{Int})} \tag{9}$$

$$\frac{\mathsf{x} :: \mathcal{Box}(\mathcal{T})}{\texttt{immut}\ \mathsf{x} :: \mathcal{IBox}(\mathcal{T})} \tag{10}$$

$$\frac{\mathsf{x} :: \mathcal{T}}{\texttt{ref}\ \mathsf{x} :: \mathcal{Box}(\mathcal{Ref}(\mathcal{T}))} \tag{11}$$

$$\frac{\mathsf{x} :: \mathcal{T}}{\texttt{\&}\ \mathsf{x} : \mathcal{Ref}(\mathcal{T})} \tag{12}$$

$$\frac{\mathsf{x} : \mathcal{Ref}(\mathcal{Box}(\mathcal{T}))}{\mathsf{x}\ \texttt{@} : \mathcal{T}} \tag{13}$$

$$\frac{\mathsf{x} : \mathcal{Ref}(\mathcal{IBox}(\mathcal{T}))}{\mathsf{x}\ \texttt{@} : \mathcal{T}} \tag{14}$$

$$\frac{\mathsf{x} : \mathcal{Ref}(\mathcal{T})}{\mathsf{x}\ \texttt{@} :: \mathcal{T}} \tag{15}$$

$$\frac{\mathsf{x} :: \mathcal{T} \qquad \mathsf{y} :: \mathcal{U} \qquad \mathcal{T} <:: \mathcal{U} \qquad \mathsf{x} : \mathcal{V}}{\mathsf{x}\ \texttt{:=}\ \mathsf{y} : \mathcal{V}} \tag{16}$$

`x` : $\mathcal{T}$ is read as “expression `x` is of type $\mathcal{T}$ and is in an *r-context*.”
`x` :: $\mathcal{T}$ is read as “variable `x` is of type $\mathcal{T}$ 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 $\mathcal{Box}(\mathcal{T})$.

For now, we omit rules for $\mathcal{Con}$ types as they only operate on r-values.

For now, we omit rules for $\mathcal{Fun}$ 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)

`int i`	$\rightarrow i$	$\rightarrow \mathcal{Box}(\mathit{Int})$
`immut int ii`	$\rightarrow ii$	$\rightarrow \mathit{IBox}(\mathit{Int})$
`ref int ri`	$\rightarrow ri$	$\rightarrow \mathcal{Box}(\mathcal{Ref}(\mathcal{Box}(\mathit{Int})))$
`immut ref int iri`	$\rightarrow iri$	$\rightarrow \mathit{IBox}(\mathcal{Ref}(\mathcal{Box}(\mathit{Int})))$
`ref immut int rii`	$\rightarrow rii$	$\rightarrow \mathcal{Box}(\mathcal{Ref}((\mathit{Immut}(\mathcal{Box}(\mathit{Int})))))$
`immut ref immut int irii`	$\rightarrow irii$	$\rightarrow \mathit{IBox}(\mathcal{Ref}((\mathit{Immut}(\mathcal{Box}(\mathit{Int})))))$

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

$$\mathcal{Box}(\mathcal{Triv}) <: \mathcal{Box}(\mathcal{Triv})$$

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

$$\mathcal{Box}(\mathcal{Triv}) <: \mathit{IBox}(\mathcal{Triv})$$

$$\mathcal{Box}(\mathit{Int}) <: \mathit{IBox}(\mathit{Int})$$

$$\mathit{IBox}(\mathcal{Triv}) <: \mathit{IBox}(\mathcal{Triv})$$

$$\mathit{IBox}(\mathit{Int}) <: \mathit{IBox}(\mathit{Int})$$

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

$$\mathcal{Ref}(\mathcal{Box}(\mathcal{Triv})) <: \mathcal{Ref}(\mathcal{Box}(\mathcal{Triv}))$$

$$\mathcal{Ref}(\mathcal{Box}(\mathit{Int})) <: \mathcal{Ref}(\mathcal{Box}(\mathit{Int}))$$

$$\mathcal{Ref}(\mathcal{Box}(\mathcal{Triv})) <: \mathcal{Ref}(\mathit{IBox}(\mathcal{Triv}))$$

$$\mathcal{Ref}(\mathcal{Box}(\mathit{Int})) <: \mathcal{Ref}(\mathit{IBox}(\mathit{Int}))$$

$$\mathcal{Ref}(\mathit{IBox}(\mathcal{Triv})) <: \mathcal{Ref}(\mathit{IBox}(\mathcal{Triv}))$$

$$\mathcal{Ref}(\mathit{IBox}(\mathit{Int})) <: \mathcal{Ref}(\mathit{IBox}(\mathit{Int}))$$