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 5, 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

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

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

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

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

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

$\mathcal{Box}?$ 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

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

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

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

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

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

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

$$\frac{\texttt{x} :: \mathcal{T} \qquad \mathcal{Box}?(\mathcal{T})}{\texttt{(x)} :: \mathcal{T}} \tag{12}$$

$$\frac{\texttt{x} :: \mathcal{T} \qquad \mathcal{Box}?(\mathcal{T})}{\texttt{\& x} : \mathcal{Ref}\ \mathcal{T}} \tag{13}$$

$$\frac{\texttt{x} :: \mathcal{Ref}\ \mathcal{T}}{\texttt{x @} : \mathcal{T}} \tag{14}$$

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

$$\frac{\texttt{x} :: \mathcal{Box}(\mathcal{T}) \qquad \texttt{y} :: \mathcal{U} \qquad \mathcal{T} <: \mathcal{U}}{\texttt{x := y} : \mathcal{Box}(\mathcal{T})} \tag{16}$$

$$\frac{\texttt{x} :: \mathcal{Box}(\mathcal{T}) \qquad \texttt{y} :: \mathcal{U} \qquad \mathcal{T} <: \mathcal{U} \qquad \mathcal{Box}?(\mathcal{U})}{\texttt{x := y} : \mathcal{Box}(\mathcal{T})} \tag{17}$$

`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 $\mathcal{Box}(\mathcal{T})$.

We omit rules for $\mathcal{Con}$ types as they only operate on r-values.

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

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

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

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

$$\mathit{Box}(\mathit{Triv}) <: \mathit{Immut}\ \mathit{Box}(\mathit{Triv})$$

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

$$\mathit{Immut}\ \mathit{Box}(\mathit{Triv}) <: \mathit{Immut}\ \mathit{Box}(\mathit{Triv})$$

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

All the rules above should work with Ref types in the similar manner:

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

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

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

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

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

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