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

December 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 type system and the syntax of the language. Two new types $\mathcal{Box}$ and IBox were introduced. In addition, two new operators <:: and :: were designed for dealing with the new types.

The $\mathcal{Box}$ and IBox Types

CCL is, by and large, value-based language. This makes it rather difficult to have types more complex than what already comprises CCL. Besides, this is a major obstacle for the implementation of heterogenous types such as heterogenous ordered pair, heterogenous vector types, etc. One possible solution to this problem is introducing the notion of container types.

Container types are boxes which are either mutable or immutable. They wrap the values allowing the composition of compound types by bringing the higher levels of abstraction. Mutable and immutable types are represented by types $\mathcal{Box}$ and the IBox respectively. If these boxes hold some type $\mathcal{T}$, then the result of wrapping the type will be $\mathcal{Box}(\mathcal{T})$ or $\mathit{IBox}(\mathcal{T})$.

The <:: Relation

For expressing relationships between container types, we introduce a new <:: operator. The sole purpose of this operator is to specify the relation between two container types. As an example, $\mathcal{T} <:: \mathcal{U}$ is read as "container type $\mathcal{T}$ is a subtype of the container type $\mathcal{U}$." We proceed by introducing rules based on the new operator.

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

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

Rules 1 and 2 specify the reflexive property of the operator on Triv and Int.

$$\frac{\mathcal{T} <:: \mathcal{B}ox(\mathcal{U})}{\mathcal{T} <:: \mathcal{IB}ox(\mathcal{U})} \tag{3}$$

$$\frac{\mathcal{B}ox(\mathcal{T}) <:: \mathcal{IB}ox(\mathcal{U})}{\mathcal{IB}ox(\mathcal{T}) <:: \mathcal{IB}ox(\mathcal{U})} \tag{4}$$

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

The :: Relation

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

$$\frac{\texttt{triv x}}{\texttt{x} :: \mathcal{B}ox(\mathcal{T}riv)} \tag{7}$$

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

$$\frac{\texttt{int x}}{\texttt{x} :: \mathcal{B}ox(\mathcal{I}nt)} \tag{9}$$

$$\frac{\texttt{x} :: \mathcal{B}ox(\mathcal{T})}{\texttt{immut x} : \mathcal{T}} \tag{10}$$

$$\frac{\texttt{x} :: \mathcal{T}}{\texttt{ref x} : \mathcal{R}ef(\mathcal{T})} \tag{11}$$

$$\frac{\texttt{x} :: \mathcal{T}}{\texttt{ref x} :: \mathcal{B}ox(\mathcal{R}ef(\mathcal{T}))} \tag{12}$$

$$\frac{\texttt{x} :: \mathcal{T}}{\texttt{\&\ x} : \mathcal{R}ef(\mathcal{T})} \tag{13}$$

$$\frac{\texttt{x} : \mathcal{R}ef(\mathcal{B}ox(\mathcal{T}))}{\texttt{x @} : \mathcal{T}} \tag{14}$$

$$\frac{\texttt{x} : \mathcal{R}ef(\mathcal{IB}ox(\mathcal{T}))}{\texttt{x @} : \mathcal{T}} \tag{15}$$

$$\frac{\texttt{x} : \mathcal{R}ef(\mathcal{T})}{\texttt{x @} :: \mathcal{T}} \tag{16}$$

$$\frac{\texttt{x} :: \mathcal{T} \qquad \texttt{y} :: \mathcal{U} \qquad \mathcal{T} <:: \mathcal{U} \qquad \texttt{x} : \mathcal{V}}{\texttt{x := y} : \mathcal{V}} \tag{17}$$

$\texttt{x} : \mathcal{T}$ 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:

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

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

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

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

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

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