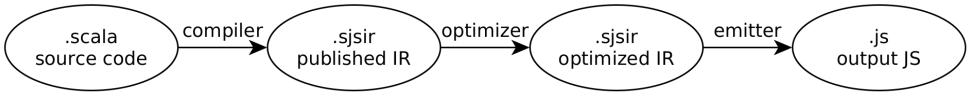
Foundations of Software
Fall 2022

Week 14

Sébastien Doeraene

Elements of the Scala.js IR type system

Scala.js compilation pipeline



Why formally study an IR

Why formally study an IR

- Optimizations may only be applicable if the type tystem is sound
- Prove that certain optimizations are correct
- Prove that the translation from source and to the target language are correct
- etc.

Mixing primitives and objects

Motivation

Featherweight Java only has objects. How do we model primitives, for example, `int` and `bool`?

Motivation

Featherweight Java only has objects. How do we model primitives, for example, `int` and `bool`?

Moreover, in Scala, primitive types are "object-like". We can use them in arbitrary type parameters, and they should behave like objects.
On the JVM, this is implemented with *boxing*. In Scala.js, however, boxing would be detrimental to *interoperability* with JavaScript.
How do we make primitives object-like without boxing?

Motivation

Featherweight Java only has objects. How do we model primitives, for example, `int` and `bool`?

Moreover, in Scala, primitive types are "object-like". We can use them in arbitrary type parameters, and they should behave like objects.
On the JVM, this is implemented with *boxing*. In Scala.js, however, boxing would be detrimental to *interoperability* with JavaScript.
How do we make primitives object-like without boxing?

Idea: make primitive types *subtypes* of their "representative classes".

Types and subtyping

`T` ::=		*types*
	`C`	*class*
	`int`	*primitive int*
	`bool`	*primitive bool*

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{...\}}}{\texttt{C <: D}}$$

$$\texttt{T <: T}$$

$$\frac{\texttt{S <: W} \qquad \texttt{W <: T}}{\texttt{S <: T}}$$

`int <: Integer` `bool <: Boolean`

Representative classes

$$\mathit{tpcls}(\texttt{C}) = \texttt{C}$$

$$\mathit{tpcls}(\texttt{int}) = \texttt{Integer}$$

$$\mathit{tpcls}(\texttt{bool}) = \texttt{Boolean}$$

$$\texttt{T} <: \mathit{tpcls}(\texttt{T})$$

Syntax (terms)

t ::=		*terms*
	x	*variable*
	t.f	*field access*
	t.m($\overline{\texttt{t}}$)	*method invocation*
	new C($\overline{\texttt{t}}$)	*object creation*
	(T) t	*cast*
	false	
	true	
	if t then t else t	
	0	
	succ t	
	pred t	
	iszero t	

Syntax (values)

$\mathtt{v}$ ::=		*values*
	$\mathtt{new\ C}(\overline{\mathtt{v}})$	*object creation*
	$\mathtt{nv}$	*numeric value*
	$\mathtt{bv}$	*boolean value*
$\mathtt{nv}$ ::=		*numeric values*
	$\mathtt{0}$	*zero*
	$\mathtt{succ\ nv}$	*non-zero*
$\mathtt{bv}$ ::=		*boolean values*
	$\mathtt{false}$	*false*
	$\mathtt{true}$	*true*

Typing rules: method calls

Adapting from Featherweight Java:

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \quad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{S}} \to \mathtt{T} \quad \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{S}_1} \quad \overline{\mathtt{S}_1} <: \overline{\mathtt{S}}}{\Gamma \vdash \mathtt{t}_0.\mathtt{m}(\overline{\mathtt{t}}) : \mathtt{T}} \qquad (\text{T-Invk})$$

What if $\mathtt{t}_0$ is a primitive?

Typing rules: method calls

Adapting from Featherweight Java:

$$\frac{\Gamma \vdash t_0 : C_0 \quad mtype(m, C_0) = \overline{S} \to T \quad \Gamma \vdash \overline{t} : \overline{S_1} \quad \overline{S_1} <: \overline{S}}{\Gamma \vdash t_0.m(\overline{t}) : T} \qquad \text{(T-Invk)}$$

What if t_0 is a primitive?

$$\frac{\Gamma \vdash t_0 : T_0 \quad mtype(m, tpcls(T_0)) = \overline{S} \to T \quad \Gamma \vdash \overline{t} : \overline{S_1} \quad \overline{S_1} <: \overline{S}}{\Gamma \vdash t_0.m(\overline{t}) : T} \qquad \text{(T-Invk)}$$

If $\Gamma \vdash x : \texttt{int}$, the call `x.m(...)` is typed by looking up `m` in `Integer`.

Example

```
class Boolean extends Object { Boolean() { super(); } }
class Integer extends Object {
  Integer() { super(); }
  int plus(int that) {
    return if (iszero that) then ((int) this)
           else (succ this.plus(pred that)); }
}
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd; }
  int sum() {
    return ((int) this.fst).plus((int) this.snd); }
}

new Pair(5, 11).sum()
```

Typing rules: fields

Adapting from Featherweight Java:

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{t}_0.\mathtt{f}_i : \mathtt{T}_i} \qquad (\text{T-Field})$$

What if $\mathtt{t}_0$ is a primitive?

Typing rules: fields

Adapting from Featherweight Java:

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{t}_0.\mathtt{f}_i : \mathtt{T}_i} \qquad (\text{T-Field})$$

What if $\mathtt{t}_0$ is a primitive?

We can't have that!

Typing rules: fields

Adapting from Featherweight Java:

$$\frac{\Gamma \vdash t_0 : C_0 \qquad \mathit{fields}(C_0) = \overline{T}\ \overline{f}}{\Gamma \vdash t_0.f_i : T_i} \qquad \text{(T-Field)}$$

What if t_0 is a primitive?

We can't have that!

Add additional well-formedness conditions for representative classes:

$$\frac{\mathit{fields}(\texttt{Integer}) = \emptyset \qquad \mathit{fields}(\texttt{Boolean}) = \emptyset}{\texttt{repr classes OK}}$$

Typing rules: casts

Straightforward generalization to all types.

$$\frac{\Gamma \vdash t_0 : S \qquad S <: T}{\Gamma \vdash (T)t_0 : T} \qquad \text{(T-UCast)}$$

$$\frac{\Gamma \vdash t_0 : S \qquad T <: S \qquad T \neq S}{\Gamma \vdash (T)t_0 : T} \qquad \text{(T-DCast)}$$

$$\frac{\Gamma \vdash t_0 : S \qquad T \not<: S \qquad S \not<: T \qquad \textit{stupid warning}}{\Gamma \vdash (T)t_0 : T} \qquad \text{(T-SCast)}$$

Typing rules: casts

Since it is an Intermediate Representation, warnings are not relevant anymore. Therefore, we keep only one typing rule for casts.

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{S}}{\Gamma \vdash \texttt{(T)t}_0 : \texttt{T}} \qquad \text{(T-Cast)}$$

Typing rules: casts

Since it is an Intermediate Representation, warnings are not relevant anymore. Therefore, we keep only one typing rule for casts.

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{S}}{\Gamma \vdash \texttt{(T)t}_0 : \texttt{T}} \qquad \text{(T-Cast)}$$

Question: can we remove the premise of that rule?

Evaluation rules

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{v}}))\mathtt{.f}_i \longrightarrow \mathtt{v}_i} \quad \text{(E-ProjNew)}$$

$$\frac{\mathit{mbody}(\mathtt{m}, \mathit{tpcls}(\mathit{vtpe}(\mathtt{v}))) = (\overline{\mathtt{x}}, \mathtt{t}_0)}{\mathtt{v.m}(\overline{\mathtt{u}}) \longrightarrow [\overline{\mathtt{x}} \mapsto \overline{\mathtt{u}}, \mathtt{this} \mapsto \mathtt{v}]\mathtt{t}_0} \quad \text{(E-InvkVal)}$$

$$\frac{\mathit{vtpe}(\mathtt{v}) <: \mathtt{T}}{(\mathtt{T})\mathtt{v} \longrightarrow \mathtt{v}} \quad \text{(E-CastVal)}$$

$\mathit{vtpe}(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{v}})) = \mathtt{C}$ $\quad \mathit{vtpe}(\mathtt{nv}) = \mathtt{int}$ $\quad \mathit{vtpe}(\mathtt{bv}) = \mathtt{bool}$

plus congruence rules and rules for `if`, `pred`, `succ` and `iszero` (omitted)