Compiling from F_i^+ to JavaScript

Yaozhu Sun

June 2, 2024

Syntax of F_i^+

Types	$A, B, C ::= \top \mid \bot \mid \mathbb{Z} \mid X \mid A \to B \mid \forall X{*}A.\ B \mid \{\ell : A\} \mid A \,\&\, B$
Expressions	$e ::= \{\} \mid n \mid x \mid \mathbf{fix}\, x{:}A.\ e \mid \lambda x{:}A.\ e{:}B \mid e_1\ e_2 \mid \Lambda X{*}A.\ e{:}B \mid e\ A$ $\mid \{\ell = e\} \mid e.\ell \mid e_1\, ,\, e_2 \mid e : A$
Type indices	$T ::= \mathbb{Z} \mid \overrightarrow{T} \mid T^{\forall} \mid \{\ell : T\} \mid T_1 \,\&\, T_2$
JavaScript code	$J ::= \varnothing \mid J_1; J_2 \mid \texttt{code}$
Destinations	$dst ::= \mathbf{nil} \mid y? \mid z$

$\boxed{\Gamma; dst \vdash e \Leftrightarrow A \rightsquigarrow J \mid z}$ *(Type-directed compilation)*

J-Nil
$$\frac{\Gamma; z \vdash e \Leftrightarrow A \rightsquigarrow J \mid z}{\Gamma; \mathbf{nil} \vdash e \Leftrightarrow A \rightsquigarrow \texttt{code} \mid z}$$

J-Opt
$$\frac{\Gamma; z \vdash e \Leftrightarrow A \rightsquigarrow J \mid z}{\Gamma; y? \vdash e \Leftrightarrow A \rightsquigarrow \texttt{code} \mid z}$$

J-Top
$$\frac{}{\Gamma; z \vdash \{\} \Rightarrow \top \rightsquigarrow \varnothing \mid z}$$

J-Int
$$\frac{T = |\mathbb{Z}|}{\Gamma; z \vdash n \Rightarrow \mathbb{Z} \rightsquigarrow \texttt{code} \mid z}$$

J-IntOpt
$$\frac{T = |\mathbb{Z}|}{\Gamma; y? \vdash n \Rightarrow \mathbb{Z} \rightsquigarrow \texttt{code} \mid z}$$

J-IntNil
$$\frac{}{\Gamma; \mathbf{nil} \vdash n \Rightarrow \mathbb{Z} \rightsquigarrow \texttt{code} \mid z}$$

J-Var
$$\frac{x : A \in \Gamma}{\Gamma; z \vdash x \Rightarrow A \rightsquigarrow \texttt{code} \mid z}$$

J-VarOpt
$$\frac{x : A \in \Gamma}{\Gamma; y? \vdash x \Rightarrow A \rightsquigarrow \texttt{code} \mid x}$$

J-VarNil
$$\frac{x : A \in \Gamma}{\Gamma; \mathbf{nil} \vdash x \Rightarrow A \rightsquigarrow \varnothing \mid x}$$

J-Fix
$$\frac{\Gamma, x : A; z \vdash e \Leftarrow A \rightsquigarrow J \mid z}{\Gamma; z \vdash \mathbf{fix}\, x{:}A.\ e \Rightarrow A \rightsquigarrow \texttt{code} \mid z}$$

J-TopAbs
$$\frac{\rceil B \lceil}{\Gamma; z \vdash \lambda x{:}A.\ e{:}B \Rightarrow A \to B \rightsquigarrow \varnothing \mid z}$$

J-Abs
$$\frac{T = |\overrightarrow{B}| \qquad \Gamma, x : A; y? \vdash e \Leftarrow B \rightsquigarrow J \mid y_0}{\Gamma; z \vdash \lambda x{:}A.\ e{:}B \Rightarrow A \to B \rightsquigarrow \texttt{code} \mid z}$$

J-App
$$\frac{\Gamma; \mathbf{nil} \vdash e_1 \Rightarrow A \rightsquigarrow J_1 \mid x \qquad \Gamma; \mathbf{nil} \vdash e_2 \Rightarrow B \rightsquigarrow J_2 \mid y \qquad \Gamma; dst \vdash x : A \bullet y : B \rightsquigarrow J_3 \mid z : C}{\Gamma; dst \vdash e_1\ e_2 \Rightarrow C \rightsquigarrow J_1; J_2; J_3 \mid z}$$

J-TopTAbs
$$\frac{\rceil B \lceil}{\Gamma; z \vdash \Lambda X{*}A.\ e{:}B \Rightarrow \forall X{*}A.\ B \rightsquigarrow \varnothing \mid z}$$

J-TAbs
$$\frac{T = |B|^{\forall} \qquad \Gamma, X * A; y? \vdash e \Leftarrow B \rightsquigarrow J_2 \mid y_0}{\Gamma; z \vdash \Lambda X{*}A.\ e{:}B \Rightarrow \forall X{*}A.\ B \rightsquigarrow \texttt{code} \mid z}$$

J-TApp

$$\frac{\Gamma; \mathbf{nil} \vdash e \Rightarrow B \rightsquigarrow J_1 \mid y \qquad \Gamma; dst \vdash y : B \bullet A \rightsquigarrow J_2 \mid z : C}{\Gamma; dst \vdash e\, A \Rightarrow C \rightsquigarrow J_1; J_2 \mid z}$$

J-TopRcd

$$\frac{\Gamma \vdash e \Rightarrow A \qquad \rceil A \lceil}{\Gamma; z \vdash \{\ell = e\} \Rightarrow \{\ell : A\} \rightsquigarrow \varnothing \mid z}$$

J-Rcd

$$\frac{T = \{\ell : |A|\} \qquad \Gamma; \mathbf{nil} \vdash e \Rightarrow A \rightsquigarrow J \mid y}{\Gamma; z \vdash \{\ell = e\} \Rightarrow \{\ell : A\} \rightsquigarrow \texttt{code} \mid z}$$

J-Proj

$$\frac{\Gamma; \mathbf{nil} \vdash e \Rightarrow A \rightsquigarrow J_1 \mid y \qquad y : A \bullet \{\ell\} \rightsquigarrow J_2 \mid z : B}{\Gamma; z \vdash e.\ell \Rightarrow B \rightsquigarrow J_1; J_2 \mid z}$$

J-Merge

$$\frac{\Gamma; z \vdash e_1 \Rightarrow A \rightsquigarrow J_1 \mid z \qquad \Gamma; z \vdash e_2 \Rightarrow B \rightsquigarrow J_2 \mid z \qquad \Gamma \vdash A * B}{\Gamma; z \vdash e_1\, , e_2 \Rightarrow A \,\&\, B \rightsquigarrow J_1; J_2 \mid z}$$

J-Anno

$$\frac{\Gamma; dst \vdash e \Leftarrow A \rightsquigarrow J \mid z}{\Gamma; dst \vdash e : A \Rightarrow A \rightsquigarrow J \mid z}$$

J-Def

$$\frac{\Gamma; x \vdash e_1 \Rightarrow A \rightsquigarrow J_1 \mid x \qquad \Gamma, x : A; z \vdash e_2 \Rightarrow B \rightsquigarrow J_2 \mid z}{\Gamma; z \vdash x = e1; e_2 \Rightarrow B \rightsquigarrow \texttt{code} \mid z}$$

J-Sub

$$\frac{\Gamma; \mathbf{nil} \vdash e \Rightarrow A \rightsquigarrow J_1 \mid x \qquad x : A <: y : B \rightsquigarrow J_2}{\Gamma; y \vdash e \Leftarrow B \rightsquigarrow J_1; J_2 \mid y}$$

J-SubEquiv

$$\frac{A \eqsim B \qquad \Gamma; dst \vdash e \Rightarrow A \rightsquigarrow J \mid z}{\Gamma; dst \vdash e \Leftarrow B \rightsquigarrow J \mid z}$$

```
/* J-Nil */
var z = {};
J;

/* J-Opt */
var z = y || {};
J;

/* J-Int */
z[T] = n;

/* J-IntOpt */
var z = n;
if (y) y[T] = n;

/* J-IntNil */
var z = n;
```

```
/* J-Var */
copy(z, x);

/* J-VarOpt */
if (y) copy(y, x);

/* J-Fix */
var x = z;
J;

/* J-Abs */
z[T] = (x, y) => {
  J;  return y0;
};
```

```
/* J-TAbs */
z[T] = (X, y) => {
  J;  return y0;
};

/* J-Rcd */
z.__defineGetter__(T, () => {
  J;
  delete this[T];
  return this[T] = y;
});

/* J-Def */
export var x = {};
J1;  J2;
```

Copying properties n.b. there seems to be some alternatives:

- `Object.assign(dst, src)` does not properly copy getters;
- `Object.defineProperties(dst, Object.getOwnPropertyDescriptors(src))` is proper but slow;
- `Object.setPrototypeOf(dst, src)` does the prototype trick but is even slower.

```
function copy(dst, src) {
  for (const prop in src) {
    var getter = src.__lookupGetter__(prop);
    if (getter) dst.__defineGetter__(prop, getter);
    else dst[prop] = src[prop];
  }
}
```

$\boxed{\Gamma; dst \vdash x : A \bullet p \rightsquigarrow J \mid z : B}$ *(Distributive application)*

JA-Nil
$$\frac{\Gamma; z \vdash x : A \bullet p \rightsquigarrow J \mid z : B}{\Gamma; \mathbf{nil} \vdash x : A \bullet p \rightsquigarrow \texttt{code} \mid z : B}$$

JA-Opt
$$\frac{\Gamma; z \vdash x : A \bullet p \rightsquigarrow J \mid z : B}{\Gamma; y? \vdash x : A \bullet p \rightsquigarrow \texttt{code} \mid z : B}$$

JA-Top
$$\frac{\rceil A \lceil}{\Gamma; z \vdash x : A \bullet p \rightsquigarrow \varnothing \mid z : \top}$$

JA-Arrow
$$\frac{\begin{array}{c} T = |\overrightarrow{B}| \\ y : C <: y_0 : A \rightsquigarrow J_1 \\ \Gamma; dst \vdash x : A \to B \bullet y_0 : A \rightsquigarrow J_2 \mid z : B \end{array}}{\Gamma; dst \vdash x : A \to B \bullet y : C \rightsquigarrow \texttt{code} \mid z : B}$$

JA-ArrowEquiv
$$\frac{A \coloneqq C \qquad T = |\overrightarrow{B}|}{\Gamma; z \vdash x : A \to B \bullet y : C \rightsquigarrow \texttt{code} \mid z : B}$$

JA-ArrowOpt
$$\frac{A \coloneqq C \qquad T = |\overrightarrow{B}|}{\Gamma; z_0? \vdash x : A \to B \bullet y : C \rightsquigarrow \texttt{code} \mid z : B}$$

JA-ArrowNil
$$\frac{A \coloneqq C \qquad T = |\overrightarrow{B}|}{\Gamma; \mathbf{nil} \vdash x : A \to B \bullet y : C \rightsquigarrow \texttt{code} \mid z : B}$$

JA-All
$$\frac{\begin{array}{c} \Gamma \vdash A * C \\ T = |B|^{\forall} \qquad Ts = \mathbf{itoa} \mid C \mid \end{array}}{\Gamma; z \vdash x : \forall X{*}A.\ B \bullet C \rightsquigarrow \texttt{code} \mid z : B[X \mapsto C]}$$

JA-AllOpt
$$\frac{\begin{array}{c} \Gamma \vdash A * C \\ T = |B|^{\forall} \qquad Ts = \mathbf{itoa} \mid C \mid \end{array}}{\Gamma; y? \vdash x : \forall X{*}A.\ B \bullet C \rightsquigarrow \texttt{code} \mid z : B[X \mapsto C]}$$

JA-AllNil
$$\frac{\begin{array}{c} \Gamma \vdash A * C \\ T = |B|^{\forall} \qquad Ts = \mathbf{itoa} \mid C \mid \end{array}}{\Gamma; \mathbf{nil} \vdash x : \forall X{*}A.\ B \bullet C \rightsquigarrow \texttt{code} \mid z : B[X \mapsto C]}$$

JA-And
$$\frac{\begin{array}{c} \Gamma; z \vdash x : A \bullet p \rightsquigarrow J_1 \mid z : A' \\ \Gamma; z \vdash x : B \bullet p \rightsquigarrow J_2 \mid z : B' \end{array}}{\Gamma; z \vdash x : A \,\&\, B \bullet p \rightsquigarrow J_1; J_2 \mid z : A' \,\&\, B'}$$

```
/* JA-Nil */
var z = {};
J;

/* JA-Opt */
var z = y || {};
J;

/* JA-Arrow */
var y0 = {};
J1;  J2;
```

```
/* JA-ArrowEquiv */
x[T](y, z);

/* JA-ArrowOpt */
var z = x[T](y, z0);

/* JA-ArrowNil */
var z = x[T](y);
```

```
/* JA-All */
x[T](Ts, z);

/* JA-AllOpt */
var z = x[T](Ts, y);

/* JA-All */
var z = x[T](Ts);
```

$\boxed{x : A \bullet \{\ell\} \rightsquigarrow J \mid z : B}$ *(Distributive projection)*

JP-Top
$$\frac{\rceil A \lceil}{x : A \bullet \{\ell\} \rightsquigarrow \varnothing \mid z : \top}$$

JP-RcdEq
$$\frac{T = \{\ell : |A|\}}{x : \{\ell : A\} \bullet \{\ell\} \rightsquigarrow \texttt{code} \mid z : A}$$

JP-RcdNeq
$$\frac{\ell_1 \neq \ell_2 \qquad T = \{\ell : |A|\}}{x : \{\ell_1 : A\} \bullet \{\ell_2\} \rightsquigarrow \varnothing \mid z : \top}$$

JP-And
$$\frac{\begin{array}{c} x : A \bullet \{\ell\} \rightsquigarrow J_1 \mid z : A' \\ x : B \bullet \{\ell\} \rightsquigarrow J_2 \mid z : B' \end{array}}{x : A \,\&\, B \bullet \{\ell\} \rightsquigarrow J_1; J_2 \mid z : A' \,\&\, B'}$$

```
/* JP-RcdEq */
var z = x[T];
```

$\boxed{x : A <: y : B \rightsquigarrow J}$ *(Coercive subtyping)*

$$\textsc{JS0-Sub}\ \frac{x : A <:^{+} y : B \rightsquigarrow J}{x : A <: y : B \rightsquigarrow J} \qquad \textsc{JS0-Int}\ \frac{T = |\mathbb{Z}| \quad x : A <:^{+} y : \mathbb{Z} \rightsquigarrow J}{x : A <: y : \mathbb{Z} \rightsquigarrow \texttt{code}} \qquad \textsc{JS0-Var}\ \frac{T = |X| \quad x : A <:^{+} y : X \rightsquigarrow J}{x : A <: y : X \rightsquigarrow \texttt{code}}$$

```
/* JS0-Int */
J;  y = y[T];
```

```
/* JS0-Var */
J;  if (primitive(X)) y = y[T];
```

$\boxed{x : A <:^{\pm} y : B \rightsquigarrow J}$ *(Coercive subtyping)*

$$\textsc{JS-Equiv}\ \frac{A \eqsim B}{x : A <:^{+} y : B \rightsquigarrow \texttt{code}} \qquad \textsc{JS-Top}\ \frac{\rceil B \lceil}{x : A <:^{\pm} y : B \rightsquigarrow \varnothing} \qquad \textsc{JS-Bot}\ \frac{T = |A|}{x : \bot <:^{\pm} y : A \rightsquigarrow \texttt{code}}$$

$$\textsc{JS-Int}\ \frac{}{x : \mathbb{Z} <:^{+} y : \mathbb{Z} \rightsquigarrow \texttt{code}} \qquad \textsc{JS-IntAnd}\ \frac{T = |\mathbb{Z}|}{x : \mathbb{Z} <:^{-} y : \mathbb{Z} \rightsquigarrow \texttt{code}} \qquad \textsc{JS-Var}\ \frac{}{x : X <:^{\pm} y : X \rightsquigarrow \texttt{code}}$$

$$\textsc{JS-Arrow}\ \frac{\begin{array}{c} T_1 = |\overrightarrow{A_2}| \qquad T_2 = |\overrightarrow{B_2}| \\ Ts = \mathbf{itoa} \mid A_1 \mid \\ x_1 : B_1 <: y_1 : A_1 \rightsquigarrow J_1 \\ x_2 : A_2 <: y_2 : B_2 \rightsquigarrow J_2 \end{array}}{x : A_1 \to A_2 <:^{\pm} y : B_1 \to B_2 \rightsquigarrow \texttt{code}}$$

$$\textsc{JS-All}\ \frac{\begin{array}{c} T_1 = |A_2|^{\forall} \\ T_2 = |B_2|^{\forall} \qquad B_1 <: A_1 \\ x_0 : A_2 <: y_0 : B_2 \rightsquigarrow J \end{array}}{x : \forall X * A_1.\ A_2 <:^{\pm} y : \forall X * B_1.\ B_2 \rightsquigarrow \texttt{code}}$$

$$\textsc{JS-Rcd}\ \frac{\begin{array}{c} T_1 = \{\ell : |A|\} \\ T_2 = \{\ell : |B|\} \\ x_0 : A <: y_0 : B \rightsquigarrow J \end{array}}{x : \{\ell : A\} <:^{\pm} y : \{\ell : B\} \rightsquigarrow \texttt{code}} \qquad \textsc{JS-Split}\ \frac{\begin{array}{c} B_1 \lhd B \rhd B_2 \\ y_1 : B_1 \rhd z : B \lhd y_2 : B_2 \rightsquigarrow J_3 \\ x : A <:^{\pm} y_1 : B_1 \rightsquigarrow J_1 \\ x : A <:^{\pm} y_2 : B_2 \rightsquigarrow J_2 \end{array}}{x : A <:^{\pm} z : B \rightsquigarrow \texttt{code}} \qquad \textsc{JS-AndL}\ \frac{x : A <:^{-} y : C \rightsquigarrow J}{x : A \,\&\, B <:^{\pm} y : C \rightsquigarrow J}$$

$$\textsc{JS-AndR}\ \frac{x : B <:^{-} y : C \rightsquigarrow J}{x : A \,\&\, B <:^{\pm} y : C \rightsquigarrow J}$$

```
/* JS-Equiv */
copy(y, x);

/* JS-Bot */
y[T] = null;

/* JS-Int */
y[T] = x;

/* JS-IntAnd */
y[T] = x[T];

/* JS-Var */
copy(y, x);
```

```
/* JS-Arrow */
y[T2] = (x1, y2) => {
  var y1 = {};  J1;
  var x2 = x[T1](y1);
  y2 = y2 || {};
  J2;  return y2;
};

/* JS-All */
y[T2] = (X, y0) => {
  var x0 = x[T1](X);
  y0 = y0 || {};
  J;  return y0;
};
```

```
/* JS-Rcd */
y.__defineGetter__(T2, () => {
  var x0 = x[T1];
  var y0 = {};  J;
  delete this[T];
  return this[T] = y0;
});

/* JS-Split */
var y1 = {}; // if y1 != z
var y2 = {}; // if y2 != z
J1;  J2;  J3;
```

$\boxed{x : A \ \triangleright \ z : C \ \triangleleft \ y : B \rightsquigarrow J}$ *(Coercive merging)*

JM-And

$$\frac{}{z : A \ \triangleright \ z : A \,\&\, B \ \triangleleft \ z : B \rightsquigarrow \varnothing}$$

JM-Arrow

$$\frac{\begin{array}{c} T = |\overrightarrow{B}| \\ T_1 = |\overrightarrow{B_1}| \qquad T_2 = |\overrightarrow{B_2}| \\ y_1 : B_1 \ \triangleright \ y : B \ \triangleleft \ y_2 : B_2 \rightsquigarrow J \end{array}}{x_1 : A \to B_1 \ \triangleright \ z : A \to B \ \triangleleft \ x_2 : A \to B_2 \rightsquigarrow \texttt{code}}$$

JM-All

$$\frac{\begin{array}{c} T = |B|^{\forall} \\ T_1 = |B_1|^{\forall} \qquad T_2 = |B_2|^{\forall} \\ y_1 : B_1 \ \triangleright \ y : B \ \triangleleft \ y_2 : B_2 \rightsquigarrow J \end{array}}{x_1 : \forall X * A.\ B_1 \ \triangleright \ z : \forall X * A.\ B \ \triangleleft \ x_2 : \forall X * A.\ B_2 \rightsquigarrow \texttt{code}}$$

JM-Rcd

$$\frac{\begin{array}{c} T = \{\ell : |A|\} \\ T_1 = \{\ell : |A_1|\} \\ T_2 = \{\ell : |A_2|\} \\ y_1 : A_1 \ \triangleright \ y : A \ \triangleleft \ y_2 : A_2 \rightsquigarrow J \end{array}}{x_1 : \{\ell : A_1\} \ \triangleright \ z : \{\ell : A\} \ \triangleleft \ x_2 : \{\ell : A_2\} \rightsquigarrow \texttt{code}}$$

```
/* JM-Arrow */
z[T] = (p, y) => {
  y = y || {};
  var y1 = {}; // if y1 != y
  var y2 = {}; // if y2 != y
  x1[T1](p, y1);
  x2[T2](p, y2);
  J;  return y;
};
```

```
/* JM-All */
z[T] = (X, y) => {
  y = y || {};
  var y1 = {}; // if y1 != y
  var y2 = {}; // if y2 != y
  x1[T1](X, y1);
  x2[T2](X, y2);
  J;  return y;
};
```

```
/* JM-Rcd */
z.__defineGetter__(T, () => {
  var y = {};
  var y1 = {}; // if y1 != y
  var y2 = {}; // if y2 != y
  copy(y1, x1[T1]);
  copy(y2, x2[T2]);
  J;
  delete this[T];
  return this[T] = y;
});
```