引入 Join Point 的中间语言模型

设计背景

对于 **if (if e1 then e2 else e3) then e4 else e5** 样式的高级语言语句，编译器通常会引入称为 commuting conversion 的结构转换，成为

if e1 then (if e2 then e4 else e5)

else (if e3 then e4 else e5)

如此一来如果 e4 和 e5 是很长的代码串则会在程序中引入大量冗余，而如果用 let 绑定的形式避免冗余，则成为

let { j4 () = e4; j5 () = e5 }

in if e1 then (if e2 then j4 () else j5 ())

else (if e3 then j4 () else j5 ()

由于 e4 和 e5 是语句序列所以需要引入新定义的函数 j4 和 j5，这样在调用 j4()，j5()时会引起函数转移时保存上下文和局部存储空间分配的开销，这样的内存开销不是原有代码所需要的。

而如果使用 Continuation Passing Style（CPS）形式的中间语言还可能对于 e2 和 e3 引入不同名的 Continuation 变量来替代 e4 或 e5，让冗余代码变得难以识别，并加深代码层次，不利于对多层 join points 进行优化。

定义新的语法元素 Join Point 来指示 e4、e5 这样的语句结构，将充分体现其控制流合并的特性，并能进行多种语句优化。

定义 Join Point

在已有的 Glasgow Haskell Compiler（GHC）中的中间语言的基础上添加 join point 的静态语义

$$\frac{\Gamma,\vec{a},\overrightarrow{x{:}\sigma};\Delta \vdash u : \tau \qquad \Gamma;\Delta,(\mathrm{j}:\forall\vec{a}.\vec{\sigma}\to\forall \mathrm{r.r}) \vdash \mathrm{e}:\tau}{\Gamma;\Delta \vdash \mathrm{join}\ \ \mathrm{j}\,\vec{a}\ \overrightarrow{x{:}\sigma} = u \ \mathrm{in}\ \mathrm{e}:\tau}$$

$$\frac{\overrightarrow{\Gamma,\vec{a},\overrightarrow{x{:}\sigma};\Delta,\overrightarrow{j{:}\forall\vec{a}.\,\vec{\sigma}\to\forall r.r} \vdash u:\tau} \quad \Gamma;\Delta,\overrightarrow{j{:}\forall\vec{a}.\,\vec{\sigma}\to\forall r.r}\vdash e:\tau}{\Gamma;\Delta\vdash \mathbf{join\,rec}\,\overrightarrow{j\ \vec{a}\ \overrightarrow{x{:}\sigma} = u}\ \mathbf{in}\, e:\tau}\ \textsc{RJBind}$$

$$\frac{(j{:}\forall\vec{a}.\,\vec{\sigma}\to\forall r.r)\in\Delta \quad \overrightarrow{\Gamma;\varepsilon\vdash u:\sigma\{\overrightarrow{\varphi/a}\}}}{\Gamma;\Delta\vdash\mathbf{jump}\,j\,\vec{\varphi}\,\vec{u}\,\tau:\tau}\ \textsc{Jump}$$

其中 join point 具有（多个）类型变量 $\vec{a}$，（多个）普通变量 $\overrightarrow{x:\sigma}$，和一个返回类型 τ 。其动态语义

$$\left\langle \begin{matrix} \mathbf{jump}\,j\,\vec{\varphi}\,\vec{v}\,\tau; \\ s' +\!\!+ (\mathbf{join}\,jb\,\mathbf{in}\,\square : s); \\ \Sigma \end{matrix} \right\rangle \mapsto \left\langle \begin{matrix} u\{\overrightarrow{\varphi/a}\}; \\ \mathbf{join}\,jb\,\mathbf{in}\,\square : s; \\ \Sigma, \overrightarrow{x = v} \end{matrix} \right\rangle \quad (jump)$$

$$\text{if } (j\ \vec{a}\ \vec{x} = u) \in jb$$

$$\left\langle \begin{matrix} A; \\ \mathbf{join}\,jb\,\mathbf{in}\,\square : s; \\ \Sigma \end{matrix} \right\rangle \mapsto \langle A;\, s;\, \Sigma\rangle \qquad (ans)$$

后就构成了以 λ 演算为主体的带 join point 的语言 $\mathrm{F_j}$（完整语法见附录 A），定义其优化规则为

$$\boxed{e = e'}$$

$$\begin{array}{rcllr}
(\lambda x{:}\sigma.e)\,v & = & \mathbf{let}\ x{:}\sigma = v\ \mathbf{in}\ e & & (\beta) \\
(\Lambda a.e)\,\varphi & = & e\{\varphi/a\} & & (\beta_\tau) \\
\mathbf{let}\ vb\ \mathbf{in}\ C[x] & = & \mathbf{let}\ vb\ \mathbf{in}\ C[v] & \text{if } (x{:}\sigma = v) \in vb & (\mathit{inline}) \\
\mathbf{let}\ vb\ \mathbf{in}\ e & = & e & \text{if } \mathrm{bv}(vb) \cap \mathrm{fv}(e) = \emptyset & (\mathit{drop}) \\
\mathbf{join}\ jb\ \mathbf{in}\ L[\vec{e}, \mathbf{jump}\ j\ \vec{\varphi}\ \vec{v}\ \tau, \vec{e'}] & = & \mathbf{join}\ jb\ \mathbf{in}\ L[\vec{e}, \mathbf{let}\ \overrightarrow{x{:}\sigma = v}\ \mathbf{in}\ u\{\overrightarrow{\varphi/a}\}, \vec{e'}] & \text{if } (j\ \vec{a}\ \overrightarrow{x{:}\sigma} = u) \in jb & (\mathit{jinline}) \\
\mathbf{join}\ jb\ \mathbf{in}\ e & = & e & \text{if } \mathrm{bv}(jb) \cap \mathrm{fv}(e) = \emptyset & (\mathit{jdrop}) \\
\mathbf{case}\ K\ \vec{\varphi}\ \vec{v}\ \mathbf{of}\ \overrightarrow{alt} & = & \mathbf{let}\ \overrightarrow{x{:}\sigma = v}\ \mathbf{in}\ e & \text{if } (K\ \overrightarrow{x{:}\sigma} \to e) \in \overrightarrow{alt} & (\mathit{case}) \\
E[\mathbf{case}\ e\ \mathbf{of}\ \overrightarrow{K\ \vec{x} \to u}] & = & \mathbf{case}\ e\ \mathbf{of}\ \overrightarrow{K\ \vec{x} \to E[u]} & & (\mathit{casefloat}) \\
E[\mathbf{let}\ vb\ \mathbf{in}\ e] & = & \mathbf{let}\ vb\ \mathbf{in}\ E[e] & & (\mathit{float}) \\
E[\mathbf{join}\ j\ \vec{a}\ \vec{x} = u\ \mathbf{in}\ e] & = & \mathbf{join}\ j\ \vec{a}\ \vec{x} = E[u]\ \mathbf{in}\ E[e] & & (\mathit{jfloat}) \\
E[\mathbf{join\ rec}\ \overrightarrow{j\ \vec{a}\ \vec{x} = u}\ \mathbf{in}\ e] & = & \mathbf{join\ rec}\ \overrightarrow{j\ \vec{a}\ \vec{x} = E[u]}\ \mathbf{in}\ E[e] & & (\mathit{jfloat}_{rec}) \\
E[\mathbf{jump}\ j\ \vec{\varphi}\ \vec{e}\ \tau] : \tau' & = & \mathbf{jump}\ j\ \vec{\varphi}\ \vec{e}\ \tau' & & (\mathit{abort})
\end{array}$$

$$\boxed{e = e'}$$

$$\begin{array}{rcll}
\mathbf{let}\ f = \Lambda\vec{a}.\lambda\vec{x}.u\ \mathbf{in}\ L[\vec{e}] : \tau & = & \mathbf{join}\ j\ \vec{a}\ \vec{x} = u\ \mathbf{in}\ L[\overrightarrow{\mathsf{tail}_\rho(e)}] & (\mathit{contify}) \\
& & \quad \text{if } \rho(f\ \vec{a}\ \vec{x}) = \mathbf{jump}\ j\ \vec{a}\ \vec{x}\ \tau & \\
& & \quad \text{and } f \notin \mathrm{fv}(L), u : \tau & \\
\mathbf{let\ rec}\ \overrightarrow{f = \Lambda\vec{a}.\lambda\vec{x}.L[\vec{u}]}\ \mathbf{in}\ L'[\vec{e}] : \tau & = & \mathbf{join\ rec}\ \overrightarrow{j\ \vec{a}\ \vec{x} = L[\overrightarrow{\mathsf{tail}_\rho(u)}]}\ \mathbf{in}\ L'[\overrightarrow{\mathsf{tail}_\rho(e)}] & (\mathit{contify}_{rec}) \\
& & \quad \text{if } \overrightarrow{\rho(f\ \vec{a}\ \vec{x}) = \mathbf{jump}\ j\ \vec{a}\ \vec{x}\ \tau} & \\
& & \quad \text{and } \overrightarrow{f \notin \mathrm{fv}(\vec{L})}, f \notin \mathrm{fv}(L'), \overrightarrow{L[\vec{u}] : \tau} &
\end{array}$$

$$\begin{array}{rcll}
\mathsf{tail}_\rho(f\ \vec{\sigma}\ \vec{u}) & \triangleq & e\{\overrightarrow{\sigma/a}\}\{\overrightarrow{u/x}\} & \text{if } \rho(f\ \vec{a}\ \vec{x}) = e \text{ and } \mathrm{dom}(\rho) \cap \mathrm{fv}(\vec{u}) = \emptyset \\
\mathsf{tail}_\rho(e) & \triangleq & e & \text{if } \mathrm{dom}(\rho) \cap \mathrm{fv}(e) = \emptyset \\
\mathsf{tail}_\rho(e) & \triangleq & \text{undefined} & \text{otherwise}
\end{array}$$

其中 E[$\vec{e}$] 代表对 $\vec{e}$ 中语句进行枚举，例如 E[$\vec{e}$] =

```
Case v of A → e1
          B → e2
          C → e3
```

利用 Join Point 进行代码优化

对于

```
Let f x = rhs in
Case a of A → ... f y
          B → ... f z
```

这样形如 let f x = rhs in E[...f y]的代码，应用 *float* 规则可将 let 放入枚举中变成

```
Case a of A → Let f x = rhs in ... f y
          B → Let f x = rhs in ... f z
```

因为是尾调用，可用 *contify* 规则将普通函数转化成 Join Point 以节省函数调用开销

```
Case a of A → join f x = rhs in ... jump f y τ
          B → join f x = rhs in ... jump f z τ
```

然后使用 *jfloat* 规则将 Join Point 提出 case 外

```
Join f x = case a of A → rhs[x/y]
                     B → rhs[x/z]
in case ... of ... → ... jump f y τ
```

用 *abort* 规则可以省略 jump 处的无效 case

```
Join f x = Case a of A → rhs[x/y]
```

```
                B → rhs[x/z]
in ... jump f y τ
```

这样实现了 case（if-else）在内外层代码之间的（双向）移动，当存在嵌套 case，即 rhs 中也是 case 语句时，可以将内外层 case 联合起来进行下一步优化。

例如

$$
\begin{array}{l}
any = \Lambda a.\lambda(p : a \to Bool)(xs : [a]). \\
\quad \textbf{case} \left(\begin{array}{l} \textbf{join}\ go\ xs = \textbf{case}\ xs\ \textbf{of} \\ \quad x : xs' \to \textbf{if}\ p\ x\ \textbf{then}\ Just\ x \\ \qquad\qquad\qquad\quad \textbf{else}\ \ \textbf{jump}\ go\ xs'\ (Maybe\ a) \\ \quad [] \qquad\ \to Nothing \\ \textbf{in jump}\ go\ xs\ (Maybe\ a) \end{array} \right) \textbf{of} \\
\qquad \{Just\ \_ \to True; Nothing \to False\}
\end{array}
$$

将外层 case 移入内层再优化就成为

$$
\begin{array}{l}
any = \Lambda a.\lambda(p : a \to Bool)(xs : [a]). \\
\qquad \textbf{join}\ go\ xs = \textbf{case}\ xs\ \textbf{of} \\
\qquad\quad x : xs' \to \textbf{if}\ p\ x\ \textbf{then}\ True \\
\qquad\qquad\qquad\qquad\quad \textbf{else}\ \ \textbf{jump}\ go\ xs'\ Bool \\
\qquad\quad [] \qquad\ \to False \\
\qquad \textbf{in jump}\ go\ xs\ Bool
\end{array}
$$

相比于 CPS 的优越性

- F_j 是基于 A-Normal Form 的形式，与函数式语言比较接近，语句形式比 CPS 简洁；
- CPS 对代码求值顺序有强制规定，而 F_j 没有，并且 GHC 原有的 let floating 和新引入的 *float*、*jfloat* 等规则能方便地交换代码顺序，利于优化；
- CPS 所需的一些将函数转化为 Continuation 的操作可能因为重命名而引入难以优化的代码；
- F_j 能利用 GHC 已有的允许用户自定义优化规则的系统。

附录 A

完整的 F_j 语法，syntax:

Terms

x	$\in$	Term variables	
j	$\in$	Label variables	
e, u, v	$::=$	$x \mid l \mid \lambda x{:}\sigma.e \mid e\,u$	
	$\mid$	$\Lambda a.e \mid e\,\varphi$	Type polymorphism
	$\mid$	$K\,\vec{\varphi}\,\vec{e}$	Data construction
	$\mid$	$\mathbf{case}\ e\ \mathbf{of}\ \overrightarrow{alt}$	Case analysis
	$\mid$	$\mathbf{let}\ vb\ \mathbf{in}\ v$	Let binding
	$\mid$	$\mathbf{join}\ jb\ \mathbf{in}\ u$	Join-point binding
	$\mid$	$\mathbf{jump}\ j\ \vec{\varphi}\ \vec{e}\ \tau$	Jump
alt	$::=$	$K\ \overrightarrow{x{:}\sigma} \to u$	Case alternative

Value bindings and join-point bindings

vb	$::=$	$x{:}\tau = e$	Non-recursive value
	$\mid$	$\mathbf{rec}\ \overrightarrow{x{:}\tau = e}$	Recursive values
jb	$::=$	$j\ \vec{a}\ \overrightarrow{x{:}\sigma} = e$	Non-recursive join point
	$\mid$	$\mathbf{rec}\ \overrightarrow{j\ \vec{a}\ \overrightarrow{x{:}\sigma} = e}$	Recursive join points

Answers

$A \ ::= \ \lambda x{:}\sigma.e \mid \Lambda a.e \mid K\,\vec{\varphi}\,\vec{v}$

Types

a, b	$\in$	Type variables	
τ, σ, φ	$::=$	a	Variable
	$\mid$	T	Datatype
	$\mid$	$\sigma \to \tau$	Function type
	$\mid$	$\tau\,\varphi$	Application
	$\mid$	$\forall a.\,\tau$	Polymorphic type

Frames, evaluation contexts, and stacks

F	$::=$	$\Box\, v$	Applied function
	$\mid$	$\Box\, \tau$	Instantiated polymorphism
	$\mid$	$\mathbf{case}\ \Box\ \mathbf{of}\ \overrightarrow{p \to u}$	Case scrutinee
	$\mid$	$\mathbf{join}\ jb\ \mathbf{in}\ \Box$	Join point
E	$::=$	$\Box \mid F[E]$	Evaluation contexts
s	$::=$	$\varepsilon \mid F : s$	Stacks

Tail contexts

L	$::=$	$\Box$	Empty unary context
	$\mid$	$\mathbf{case}\ e\ \mathbf{of}\ \overrightarrow{p \to L}$	Case branches
	$\mid$	$\mathbf{let}\ vb\ \mathbf{in}\ L$	Body of let
	$\mid$	$\mathbf{join}\ j\ \vec{a}\ \overrightarrow{x{:}\sigma} = L\ \mathbf{in}\ L'$	Join point, body
	$\mid$	$\mathbf{join\ rec}\ \overrightarrow{j\ \vec{a}\ \overrightarrow{x{:}\sigma} = L}\ \mathbf{in}\ L'$	Rec join points, body

Miscellaneous

C	$\in$	General single-hole term contexts	
Σ	$::=$	$\cdot \mid \Sigma, x{:}\sigma = v$	Heap
c	$::=$	$\langle e;\ s;\ \Sigma \rangle$	Configuration

Statics:

$$\boxed{\Gamma; \Delta \vdash e : \tau}$$

$$\frac{(x{:}\tau) \in \Gamma}{\Gamma; \Delta \vdash x : \tau}\ \textsc{Var} \qquad \frac{\mathsf{typeof}(K) = \forall \vec{a}.\, \vec{\sigma} \to T\,\vec{a} \quad \overrightarrow{\Gamma; \varepsilon \vdash u : \sigma\{\varphi/a\}}}{\Gamma; \Delta \vdash K\ \vec{\varphi}\ \vec{u} : T\,\vec{\varphi}}\ \textsc{Con}$$

$$\frac{\Gamma, (x{:}\sigma); \varepsilon \vdash e : \tau}{\Gamma; \Delta \vdash \lambda(x{:}\sigma).e : \sigma \to \tau}\ \textsc{Abs} \qquad \frac{\Gamma, a; \varepsilon \vdash e : \tau}{\Gamma; \Delta \vdash \Lambda a.e : \forall a.\tau}\ \textsc{TAbs}$$

$$\frac{\Gamma; \Delta \vdash e : \sigma \to \tau \quad \Gamma; \varepsilon \vdash u : \sigma}{\Gamma; \Delta \vdash e\,u : \tau}\ \textsc{App} \qquad \frac{\Gamma; \Delta \vdash e : \forall a.\tau}{\Gamma; \Delta \vdash e\,\varphi : \tau\{\varphi/a\}}\ \textsc{TApp}$$

$$\frac{(j{:}\forall \vec{a}.\, \vec{\sigma} \to \forall r.r) \in \Delta \quad \overrightarrow{\Gamma; \varepsilon \vdash u : \sigma\{\overrightarrow{\varphi/a}\}}}{\Gamma; \Delta \vdash \mathbf{jump}\ j\ \vec{\varphi}\ \vec{u}\ \tau : \tau}\ \textsc{Jump}$$

$$\frac{\Gamma; \varepsilon \vdash u : \sigma \quad \Gamma, x{:}\sigma; \Delta \vdash e : \tau}{\Gamma; \Delta \vdash \mathbf{let}\ x{:}\sigma = u\ \mathbf{in}\ e : \tau}\ \textsc{VBind} \qquad \frac{\overrightarrow{\Gamma, \overrightarrow{x{:}\sigma}; \varepsilon \vdash u : \sigma} \quad \Gamma, \overrightarrow{x{:}\sigma}; \Delta \vdash e : \tau}{\Gamma; \Delta \vdash \mathbf{let\ rec}\ \overrightarrow{x{:}\sigma = u}\ \mathbf{in}\ e : \tau}\ \textsc{RVBind}$$

$$\frac{\Gamma, \vec{a}, \overrightarrow{x{:}\sigma}; \Delta \vdash u : \tau \quad \Gamma; \Delta, (j{:}\forall \vec{a}.\, \vec{\sigma} \to \forall r.r) \vdash e : \tau}{\Gamma; \Delta \vdash \mathbf{join}\ j\ \vec{a}\ \overrightarrow{x{:}\sigma} = u\ \mathbf{in}\ e : \tau}\ \textsc{JBind}$$

$$\frac{\overrightarrow{\Gamma, \vec{a}, \overrightarrow{x{:}\sigma}; \Delta, \overrightarrow{j{:}\forall \vec{a}.\, \vec{\sigma} \to \forall r.r} \vdash u : \tau} \quad \Gamma; \Delta, \overrightarrow{j{:}\forall \vec{a}.\, \vec{\sigma} \to \forall r.r} \vdash e : \tau}{\Gamma; \Delta \vdash \mathbf{join\ rec}\ \overrightarrow{j\ \vec{a}\ \overrightarrow{x{:}\sigma} = u}\ \mathbf{in}\ e : \tau}\ \textsc{RJBind}$$

$$\frac{\Gamma; \Delta \vdash e : T\,\vec{\varphi} \quad \overrightarrow{\mathsf{typeof}(K) = \forall \vec{a}.\, \vec{\sigma} \to T\,\vec{a}} \quad \overrightarrow{\vec{v} = \vec{\sigma}\{\overrightarrow{\varphi/a}\}} \quad \overrightarrow{\Gamma, \overrightarrow{x{:}v}; \Delta \vdash u : \tau} \quad \mathsf{ctors}(T) = \{\vec{K}\}}{\Gamma; \Delta \vdash \mathbf{case}\ e\ \mathbf{of}\ \overrightarrow{K\ \overrightarrow{x{:}v} \to u} : \tau}\ \textsc{Case}$$

Dynamics:

程序状态为三元组<e;s;Σ>，e 为当前求值的表达式，s 为 stack 状态，Σ为普通变量的绑定集合

$$\boxed{\langle e;\ s;\ \Sigma\rangle \mapsto \langle e';\ s';\ \Sigma'\rangle}$$

$$\begin{array}{rcll}
\langle F[e];\ s;\ \Sigma\rangle & \mapsto & \langle e;\ F : s;\ \Sigma\rangle & (push) \\
\langle \lambda x.e;\ \Box\, v : s;\ \Sigma\rangle & \mapsto & \langle e;\ s;\ \Sigma, x = v\rangle & (\beta) \\
\langle \Lambda a.e;\ \Box\, \varphi : s;\ \Sigma\rangle & \mapsto & \langle e\{\varphi/a\};\ s;\ \Sigma\rangle & (\beta_\tau) \\
\langle \mathbf{let}\ vb\ \mathbf{in}\ e;\ s;\ \Sigma\rangle & \mapsto & \langle e;\ s;\ \Sigma, vb\rangle & (bind) \\
\langle x;\ s;\ \Sigma[x = v]\rangle & \mapsto & \langle v;\ s;\ \Sigma[x = v]\rangle & (look) \\
\left\langle \begin{matrix} K\ \vec{\varphi}\ \vec{v}; \\ \mathbf{case}\ \Box\ \mathbf{of}\ \overrightarrow{alt} : s; \\ \Sigma \end{matrix} \right\rangle & \mapsto & \langle u;\ s;\ \Sigma, \overrightarrow{x = v}\rangle & (case) \\
 & & \text{if } (K\ \vec{x} \to u) \in \overrightarrow{alt} & \\
\left\langle \begin{matrix} \mathbf{jump}\ j\ \vec{\varphi}\ \vec{v}\ \tau; \\ s' \mathbin{+\!\!+} (\mathbf{join}\ jb\ \mathbf{in}\ \Box : s); \\ \Sigma \end{matrix} \right\rangle & \mapsto & \left\langle \begin{matrix} u\{\overrightarrow{\varphi/a}\}; \\ \mathbf{join}\ jb\ \mathbf{in}\ \Box : s; \\ \Sigma, \overrightarrow{x = v} \end{matrix} \right\rangle & (jump) \\
 & & \text{if } (j\ \vec{a}\ \vec{x} = u) \in jb & \\
\left\langle \begin{matrix} A; \\ \mathbf{join}\ jb\ \mathbf{in}\ \Box : s; \\ \Sigma \end{matrix} \right\rangle & \mapsto & \langle A;\ s;\ \Sigma\rangle & (ans)
\end{array}$$

对 F_j 的优化规则

$$\boxed{e = e'}$$

$$\begin{array}{rcl@{\quad}l@{\quad}r}
(\lambda x{:}\sigma.e)\, v & = & \mathbf{let}\ x{:}\sigma = v\ \mathbf{in}\ e & & (\beta) \\
(\Lambda a.e)\, \varphi & = & e\{\varphi/a\} & & (\beta_\tau) \\
\mathbf{let}\ vb\ \mathbf{in}\ C[x] & = & \mathbf{let}\ vb\ \mathbf{in}\ C[v] & \text{if } (x{:}\sigma = v) \in vb & (inline) \\
\mathbf{let}\ vb\ \mathbf{in}\ e & = & e & \text{if } \mathrm{bv}(vb) \cap \mathrm{fv}(e) = \emptyset & (drop) \\
\mathbf{join}\ jb\ \mathbf{in}\ L[\vec{e}, \mathbf{jump}\ j\ \vec{\varphi}\ \vec{v}\ \tau, \vec{e'}] & = & \mathbf{join}\ jb\ \mathbf{in}\ L[\vec{e}, \mathbf{let}\ \overrightarrow{x{:}\sigma = v}\ \mathbf{in}\ u\{\overrightarrow{\varphi/a}\}, \vec{e'}] & \text{if } (j\ \vec{a}\ \overrightarrow{x{:}\sigma} = u) \in jb & (jinline) \\
\mathbf{join}\ jb\ \mathbf{in}\ e & = & e & \text{if } \mathrm{bv}(jb) \cap \mathrm{fv}(e) = \emptyset & (jdrop) \\
\mathbf{case}\ K\ \vec{\varphi}\ \vec{v}\ \mathbf{of}\ \overrightarrow{alt} & = & \mathbf{let}\ \overrightarrow{x{:}\sigma = v}\ \mathbf{in}\ e & \text{if } (K\ \overrightarrow{x{:}\sigma} \to e) \in \overrightarrow{alt} & (case) \\
E[\mathbf{case}\ e\ \mathbf{of}\ \overrightarrow{K\ \vec{x} \to u}] & = & \mathbf{case}\ e\ \mathbf{of}\ \overrightarrow{K\ \vec{x} \to E[u]} & & (casefloat) \\
E[\mathbf{let}\ vb\ \mathbf{in}\ e] & = & \mathbf{let}\ vb\ \mathbf{in}\ E[e] & & (float) \\
E[\mathbf{join}\ j\ \vec{a}\ \vec{x} = u\ \mathbf{in}\ e] & = & \mathbf{join}\ j\ \vec{a}\ \vec{x} = E[u]\ \mathbf{in}\ E[e] & & (jfloat) \\
E[\mathbf{join\ rec}\ \overrightarrow{j\ \vec{a}\ \vec{x} = u}\ \mathbf{in}\ e] & = & \mathbf{join\ rec}\ \overrightarrow{j\ \vec{a}\ \vec{x} = E[u]}\ \mathbf{in}\ E[e] & & (jfloat_{rec}) \\
E[\mathbf{jump}\ j\ \vec{\varphi}\ \vec{e}\ \tau] : \tau' & = & \mathbf{jump}\ j\ \vec{\varphi}\ \vec{e}\ \tau' & & (abort)
\end{array}$$

$$\boxed{e = e'}$$

$$\begin{array}{rcll}
\mathbf{let}\ f = \Lambda\vec{a}.\lambda\vec{x}.u\ \mathbf{in}\ L[\vec{e}] : \tau & = & \mathbf{join}\ j\ \vec{a}\ \vec{x} = u\ \mathbf{in}\ L[\overrightarrow{\mathsf{tail}_\rho(e)}] & (contify) \\
 & & \quad \text{if } \rho(f\ \vec{a}\ \vec{x}) = \mathbf{jump}\ j\ \vec{a}\ \vec{x}\ \tau & \\
 & & \quad \text{and } f \notin \mathrm{fv}(L), u : \tau & \\
\mathbf{let\ rec}\ \overrightarrow{f = \Lambda\vec{a}.\lambda\vec{x}.L[\vec{u}]}\ \mathbf{in}\ L'[\vec{e}] : \tau & = & \mathbf{join\ rec}\ \overrightarrow{j\ \vec{a}\ \vec{x} = L[\overrightarrow{\mathsf{tail}_\rho(u)}]}\ \mathbf{in}\ L'[\overrightarrow{\mathsf{tail}_\rho(e)}] & (contify_{rec}) \\
 & & \quad \text{if } \overrightarrow{\rho(f\ \vec{a}\ \vec{x}) = \mathbf{jump}\ j\ \vec{a}\ \vec{x}\ \tau} & \\
 & & \quad \text{and } \overrightarrow{f \notin \mathrm{fv}(\vec{L})}, f \notin \mathrm{fv}(L'), \overrightarrow{L[\vec{u}] : \tau} &
\end{array}$$

$$\begin{array}{rcll}
\mathsf{tail}_\rho(f\ \vec{\sigma}\ \vec{u}) & \triangleq & e\{\overrightarrow{\sigma/a}\}\{\overrightarrow{u/x}\} & \text{if } \rho(f\ \vec{a}\ \vec{x}) = e \text{ and } \mathrm{dom}(\rho) \cap \mathrm{fv}(\vec{u}) = \emptyset \\
\mathsf{tail}_\rho(e) & \triangleq & e & \text{if } \mathrm{dom}(\rho) \cap \mathrm{fv}(e) = \emptyset \\
\mathsf{tail}_\rho(e) & \triangleq & \text{undefined} & \text{otherwise}
\end{array}$$

参考文献

Compiling without continuations

The essence of compiling with continuations

Stream Fusion:From Lists to Streams to Nothing at all