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，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，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。

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 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

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

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

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

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

$$\begin{array}{rcllr} (\lambda x{:}\sigma.e)\, v & = & \mathbf{let}\, x{:}\sigma = v\, \mathbf{in}\, e & & (\beta) \\ (\Lambda a.e)\, \varphi & = & e\{\varphi/a\} & & (\beta_\tau) \\ \mathbf{let}\, vb\, \mathbf{in}\, C[x] & = & \mathbf{let}\, vb\, \mathbf{in}\, C[v] & \text{if } (x{:}\sigma = v) \in vb & (inline) \end{array}$$

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

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

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

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

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

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

将外层 case 移入内层再优化就成为

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

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

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

Value bindings and join-point bindings

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

Answers

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

Types

$$
\begin{array}{lcll}
a,b & \in & \text{Type variables} & \\
\tau,\sigma,\varphi & ::= & a & \text{Variable}\\
 & \mid & T & \text{Datatype}\\
 & \mid & \sigma \to \tau & \text{Function type}\\
 & \mid & \tau\,\varphi & \text{Application}\\
 & \mid & \forall a.\,\tau & \text{Polymorphic type}
\end{array}
$$

Frames, evaluation contexts, and stacks

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

Tail contexts

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

Miscellaneous

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

Statics:

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

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

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

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

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

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

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

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

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

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

对 F_j 的优化规则

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

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

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

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

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

参考文献

Compiling without continuations

The essence of compiling with continuations

Stream Fusion:From Lists to Streams to Nothing at all