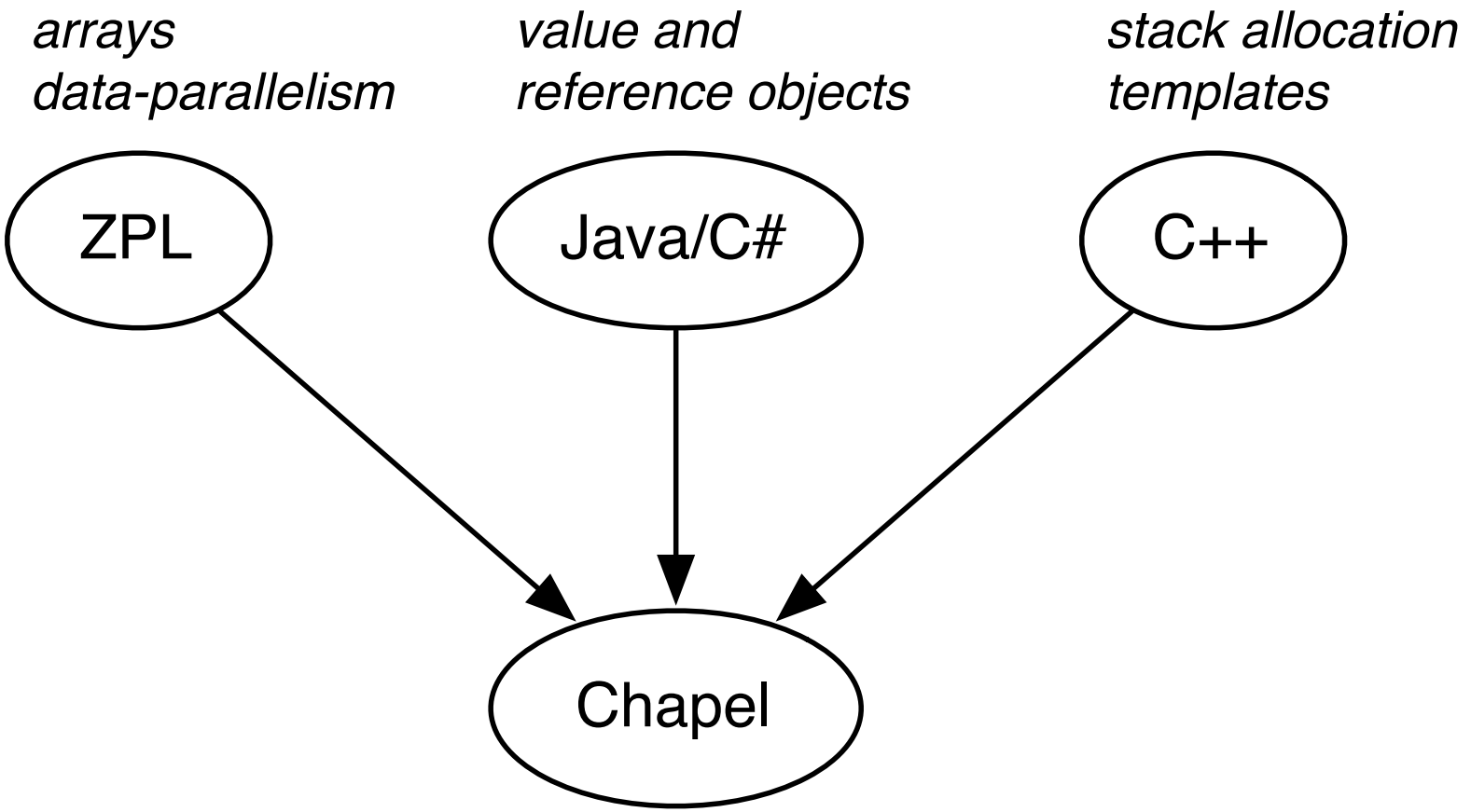
Effects for Funargs

Jeremy G. Siek[1] M. Vitousek[1] J. Turner[1]

[1]University of Colorado at Boulder

HOPE 2012

Chapel's Goals: Productivity and Parallelism



Higher-order Functions and Lexical Scope

```
var my_state = read(string);
var addresses_in_my_state = new Vector(Address);

copy_if(addresses, back_inserter(addresses_in_my_state),
        fun (a:Address) {
            return a.state == my_state;
        });
```

Lexical Scope and Stack Allocation, Danger!

```
proc deriv(f: float → float, d:float) {
  return fun(x:float) {
              return f(x + d) − f(x − d);
          };
}

proc square(x:float) { return x ∗ x; }

var line = deriv(square, 0.01);
// Kaboom!
writeln(line(5.0));
```

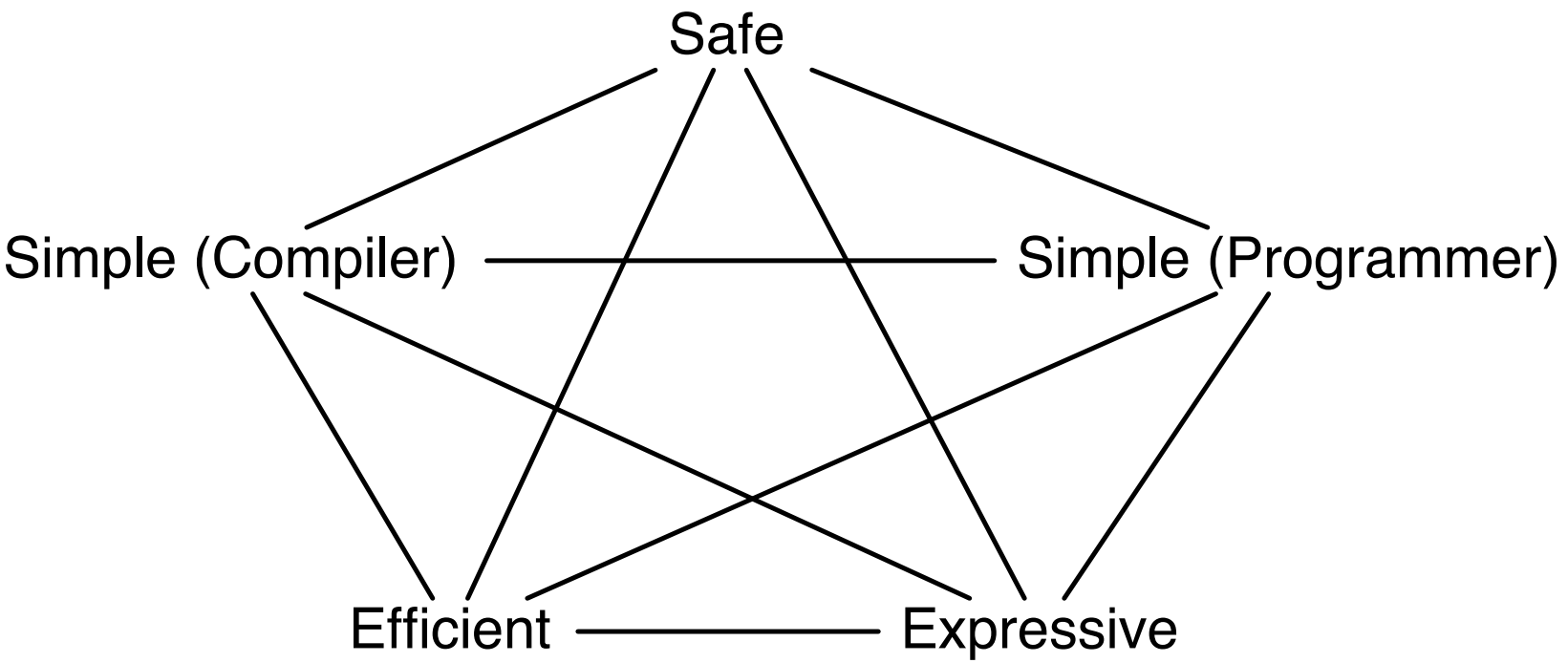
The function of FUNCTION in LISP or why the FUNARG problem should be called the environment problem. J. Moses, SIGSAM Bull., 1970.

Representatives of Prior Work

Research	Industry
Effects: Talpin & Jouvelot	GC'd languages (H)
ML $\Rightarrow$ Regions: Toft & Talpin	Blocks in Objective C (V, H)
Cyclone: Grossman et al.	Lambda in C++ (V, R)
RT Java: Boyapati et al.	Inner Classes in Java (V)

H: heap
V: value
R: reference

Design Considerations



Capture by reference or by value

By Reference	By Value
+Always Fast	−Sometimes slow
+Downard funargs ok	+Downward funargs ok
−No upward funargs	+Upward funargs ok
	− No mutation

Our Approach

1. Programmer chooses between capture by-reference or by-value. (like C++)
   - Here we show by-reference as the default.
   - The default could also be by-value or neither.
2. A type and effect system disallows escaping references. (like effect systems for regions)
3. Effects are sets of term variables. (new)

Example 1 Revisited

- No annotations
- Capture by reference is the default

```
var my_state = read(string);
var addresses_in_my_state = new Vector(Address);

copy_if(addresses, back_inserter(addresses_in_my_state),
        fun (a:Address) {
            return a.state == my_state;
        });
```

Example 2 with capture by-value

- Annotate capture by-value
- Upward funarg is ok

```
proc deriv(f: float → float, d:float) {
  return fun(x:float) f, d {
                return f(x + d) − f(x − d);
            };
}

proc square(x:float) { return x ∗ x; }

var times_two = deriv(square, 0.5);
writeln(times_two(5.0));
> 10.0
```

Example 2 with capture by-reference

- Type system detects escaping references

```
proc deriv(f: float → float, d:float) {
  return fun(x:float) {
              return f(x + d) − f(x − d);
          };
}
```

Error: f and d escape from deriv

```
proc square(x:float) { return x ∗ x; }

var times_two = deriv(square, 0.5);
writeln(times_two(5.0));
```

Inferred Effect Polymorphism

proc deriv(f: **float** $\rightarrow$ **float**, d:**float**) {
  return fun(x:**float**) f,d { **return** f(x + d) − f(x − d); };
}
// deriv : $\forall\epsilon$. (**float** $\xrightarrow{\epsilon}$ **float**, **float**) $\rightarrow$ **float** $\xrightarrow{\epsilon}$ **float**
var pi = 3.14159;

proc times_pi(x:**float**) { **return** pi ∗ x; } // float $\xrightarrow{\text{pi}}$ float

var almost_pi = deriv(times_pi, 0.5); // float $\xrightarrow{\text{pi}}$ float

writeln(almost_pi(1.0));
> 3.14158999999

Region-based memory management in Cyclone. Grossman et al. PLDI 2002

The Core Language

ground effects	γ	$::=$	$\{x\} \mid \epsilon$
effects	φ	$::=$	$\emptyset \mid \gamma \mid \varphi \cup \varphi$
types	T	$::=$	$\texttt{int} \mid T \xrightarrow{\varphi} T \mid \forall\epsilon.\, T$
expressions	e	$::=$	$n \mid x \mid f \mid e\texttt{<}\varphi\texttt{>}$
abstractions	f	$::=$	$\texttt{fun}(x : T)\,{}^{\varphi}_{T}\overline{y = e}\{s\} \mid \Lambda\epsilon.\, f$
statements	s	$::=$	$x\texttt{=}e\texttt{(}e\texttt{)}\texttt{;}\ s \mid \texttt{ret}\ e\texttt{(}e\texttt{)} \mid \texttt{ret}\ e$

- Function parameters (x) are bound to stack locations.
- Function-local variables ($\overline{y}$) reside in the function.

Evaluation of Expressions

$$
\begin{array}{lrcl}
\text{stack locations} & \ell & \in & \mathbb{N} \\
\text{ground effects} & \gamma & ::= & \ldots \mid \{\ell_T\} \\
\text{expressions} & e & ::= & \ldots \mid \ell_T \\
\text{values} & v & ::= & n \mid \mathtt{fun}(x:T)^{\varphi}_{T}\overline{y=v}\{s\} \\
\text{stack o' values} & \sigma & ::= & [] \mid v :: \sigma
\end{array}
$$

$$\boxed{[\![e]\!]_\sigma = v}$$

$$
\begin{aligned}
[\![\ell_T]\!]_\sigma &= \sigma_{n-1-\ell} \qquad \text{where } n = |\sigma| \\
[\![\mathtt{fun}(x:T_1)^{\varphi}_{T_2}\overline{y=e}\{s\}]\!]_\sigma &= \mathtt{fun}(x:T_1)^{\varphi}_{T_2}\overline{y=v}\{s\} \qquad \text{if } [\![\overline{e}]\!]_\sigma = \overline{v} \\
[\![\Lambda\epsilon.\, f]\!]_\sigma &= \Lambda\epsilon.\, f \\
[\![e\langle\varphi\rangle]\!]_\sigma &= [\![[\epsilon{:=}\varphi]e']\!]_\sigma \qquad \text{if } [\![e]\!]_\sigma = \Lambda\epsilon.\, e'
\end{aligned}
$$

Abstract Machine Transitions

$$\begin{array}{lrcl} \text{Stacks} & \kappa & ::= & [] \mid (x{:}T, s, n) :: \kappa \\ \text{States} & \varsigma & ::= & \langle s, \kappa, \sigma, n \rangle \end{array}$$

$\boxed{\varsigma \longrightarrow \varsigma}$

$$\langle (x{=}e_1\,(e_2)\,;\; s_1), \kappa, \sigma, n \rangle \qquad \text{(Call)}$$
$$\longrightarrow \langle [y{:=}\ell_{T_1}, \overline{z}{:=}\overline{v}]s_2, (x{:}T_2, s_1, n) :: \kappa, [\![e_2]\!]_\sigma :: \sigma, 1 \rangle$$
$$\text{where } [\![e_1]\!]_\sigma = \texttt{fun}(y:T_1)^{\varphi}_{T_2}\overline{z{=}v}\{s_2\} \text{ and } \ell = |\sigma|$$

$$\langle \texttt{ret}\, e_1\,(e_2), \kappa, \sigma, n \rangle \qquad \text{(TailCall)}$$
$$\longrightarrow \langle [y{:=}\ell_{T_1}]s, \kappa, [\![e_2]\!]_\sigma :: \mathit{drop}(n, \sigma), 1 \rangle$$
$$\text{where } [\![e_1]\!]_\sigma = \texttt{fun}(y{:}T_1)^{\varphi}_{T_2}\{s\} \text{ and } \ell = |\sigma| - n$$

$$\langle \texttt{ret}\, e, (x{:}T, s, n_2) :: \kappa, \sigma, n_1 \rangle \qquad \text{(Return)}$$
$$\longrightarrow \langle [x{:=}\ell_T]s, \kappa, [\![e]\!]_\sigma :: \mathit{drop}(n_1, \sigma), n_2 + 1 \rangle$$
$$\text{where } \ell = |\sigma| - n_1$$

Type System, Expressions

$$\Gamma ::= \emptyset \mid \Gamma, x{:}T \mid \Gamma, x{:}\mathsf{ref}(T) \mid \Gamma, \epsilon$$

$\boxed{\Gamma; \varphi \vdash e : T}$

$$\frac{x{:}\mathsf{ref}(T) \in \Gamma \quad \{x\} \subseteq \varphi}{\Gamma; \varphi \vdash x : T} \qquad \frac{x{:}T \in \Gamma}{\Gamma; \varphi \vdash x : T} \qquad \frac{\{\ell_T\} \subseteq \varphi}{\Gamma; \varphi \vdash \ell_T : T}$$

$$\frac{\Gamma; \emptyset \vdash T_1 \xrightarrow{\varphi_2} T_2 \qquad \Gamma; \varphi_1 \vdash \overline{e} : \overline{T} \qquad \Gamma, \overline{y : T}, x{:}\mathsf{ref}(T_1); \varphi_2 \cup \{x\}; \{x\} \vdash s : T_2}{\Gamma; \varphi_1 \vdash \mathtt{fun}(x : T_1) \overset{\varphi_2}{T_2} \overline{y = e}\{s\} : T_1 \xrightarrow{\varphi_2} T_2} \qquad \frac{\Gamma, \epsilon; \emptyset \vdash f : T}{\Gamma; \varphi \vdash \Lambda\epsilon.\, f : \forall\epsilon.\, T}$$

$$\frac{\Gamma; \varphi_1 \vdash e : \forall\epsilon.\, T \qquad \Gamma; \emptyset \vdash \varphi_2}{\Gamma; \varphi_1 \vdash e{<}\varphi_2{>} : [\epsilon{:=}\varphi_2]T}$$

Type System, Statements

$$\frac{\Gamma;\varphi_1 \vdash e_1 : T_1 \xrightarrow{\varphi_3} T_2 \qquad \Gamma;\varphi_1 \vdash e_2 : T_1 \qquad \Gamma, x{:}\mathsf{ref}(T_2);\varphi_1 \cup \{x\};\varphi_2 \cup \{x\} \vdash s : T_3 \qquad \varphi_3 \subseteq \varphi_1}{\Gamma;\varphi_1;\varphi_2 \vdash x{=}e_1(e_2)\,;\; s : T_3}$$

$$\frac{\Gamma;\varphi_1 \vdash e_1 : T_1 \xrightarrow{\varphi_3} T_2 \qquad \Gamma;\varphi_1 \vdash e_2 : T_1 \qquad \varphi_3 \subseteq \varphi_1 - \varphi_2 \qquad \mathrm{FV}(T_2) \cap \varphi_2 = \emptyset}{\Gamma;\varphi_1;\varphi_2 \vdash \mathtt{ret}\, e_1(e_2) : T_2}$$

$$\frac{\Gamma;\varphi_1 \vdash e : T \qquad \mathrm{FV}(T) \cap \varphi_2 = \emptyset}{\Gamma;\varphi_1;\varphi_2 \vdash \mathtt{ret}\, e : T}$$

Lemma (Locals not in Return Type)

If $\Gamma;\varphi_1;\varphi_2 \vdash s : T$, *then* $\mathrm{FV}(T) \cap \varphi_2 = \emptyset$.

Type System, Abstract Machine

$$\boxed{\vdash \varsigma : T}$$

$$\frac{\begin{array}{c}\emptyset; \varphi_1; \varphi_2 \vdash s : T_1 \\ \emptyset; \Sigma \vdash \varphi_1 \quad \emptyset; \mathit{take}(n, \Sigma) \vdash \varphi_2 \quad \emptyset; \mathit{drop}(n, \Sigma) \vdash \varphi_1 - \varphi_2 \\ \vdash \sigma : \Sigma \qquad\qquad \mathit{drop}(n, \Sigma) \vdash \kappa : T_1 \Rightarrow T_2\end{array}}{\vdash \langle s, \kappa, \sigma, n \rangle : T_2}$$

Type System, Abstract Machine

$\boxed{\vdash \sigma : \Sigma}$

$$\frac{}{\vdash \epsilon : \epsilon} \qquad \frac{\emptyset; \emptyset \vdash v : T \qquad \vdash \sigma : \Sigma}{\vdash v :: \sigma : T :: \Sigma}$$

$\boxed{\Sigma \vdash \kappa : T \Rightarrow T}$

$$\frac{}{\Sigma \vdash \epsilon : T \Rightarrow T} \qquad \frac{\begin{array}{c}\emptyset; \Sigma \vdash \varphi_1 \qquad \emptyset; \mathit{take}(n, \Sigma) \vdash \varphi_2 \\ \emptyset; \mathit{drop}(n, \Sigma) \vdash \varphi_1 - \varphi_2 \\ \emptyset, x{:}T_1; \varphi_1 \cup \{x\}; \varphi_2 \cup \{x\} \vdash s : T_2 \\ \emptyset \vdash T_1 \qquad \mathit{drop}(n, \Sigma) \vdash \kappa : T_2 \Rightarrow T_3\end{array}}{\Sigma \vdash (x{:}T_1, s, n) :: \kappa : T_1 \Rightarrow T_3}$$

Evaluation is Safe

Lemma (Evaluation Safety)

If $\emptyset; \varphi \vdash e : T$, $\emptyset; \Sigma \vdash \varphi$, *and* $\vdash \sigma : \Sigma$,
then $\emptyset; \emptyset \vdash [\![e]\!]_\sigma : T$.

The Machine is Safe

Theorem

If $\vdash \varsigma : T$, *then either* ς *is a final state or* $\varsigma \longrightarrow \varsigma'$ *and* $\vdash \varsigma' : T$.

Abstract Syntax of Featherweight Functional Chapel ($\mathrm{F}^2\mathrm{C}$)

variables	x, y, z	$\in$	Var
integers	n	$\in$	$\mathbb{Z}$
effect variables	ϵ	$\in$	EVar
effects	φ	$::=$	$\emptyset \mid \{x, \ldots\} \mid \epsilon$
types	T	$::=$	$\texttt{int} \mid T \to T \mid T \xrightarrow{\varphi} T \mid \forall \overline{\epsilon}.\, T$
expressions	e	$::=$	$n \mid x \mid \texttt{fun<}\overline{\epsilon}\texttt{>(}x\texttt{:}T\texttt{)}\overline{y}\{s\}$
statements	s	$::=$	$x\texttt{=}e\texttt{(}e\texttt{);}\ s \mid \texttt{ret}\ e \mid \texttt{ret}\ e\texttt{(}e\texttt{)}$

Effect Variable is the default for Function Types

$$\vdash \texttt{Int} \rightsquigarrow \texttt{Int}, \emptyset$$

$$\frac{\vdash T_1 \rightsquigarrow T_1' ; \bar{\epsilon}_1 \quad \vdash T_2 \rightsquigarrow T_2' ; \bar{\epsilon}_2 \quad \epsilon \text{ fresh}}{\vdash T_1 \to T_2 \rightsquigarrow T_1' \xrightarrow{\epsilon} T_2' ; (\epsilon\, \bar{\epsilon}_1 \bar{\epsilon}_2)}$$

$$\frac{\vdash T_1 \rightsquigarrow T_1' ; \bar{\epsilon}_1 \quad \vdash T_2 \rightsquigarrow T_2' ; \bar{\epsilon}_2}{\vdash T_1 \xrightarrow{\varphi} T_2 \rightsquigarrow T_1' \xrightarrow{\varphi} T_2' ; \bar{\epsilon}_1\, \bar{\epsilon}_2}$$

$$\frac{\vdash T \rightsquigarrow T' ; \bar{\epsilon}'}{\vdash \forall \bar{\epsilon}.\, T \rightsquigarrow \forall \bar{\epsilon}.\, T' ; \bar{\epsilon}'}$$

Type System and Elaboration, Expressions

$\boxed{\Gamma \vdash e \rightsquigarrow e : T \,!\, \varphi}$

$$\frac{}{\Gamma \vdash n \rightsquigarrow n : T \,!\, \emptyset}$$

$$\frac{x{:}\mathsf{ref}(T) \in \Gamma}{\Gamma \vdash x \rightsquigarrow x : T \,!\, \{x\}} \qquad \frac{x{:}T \in \Gamma}{\Gamma \vdash x \rightsquigarrow x : T \,!\, \emptyset}$$

$$\frac{\begin{array}{cc} \vdash T_1 \rightsquigarrow T_1' ; \overline{\epsilon}_2 & \Gamma \vdash \overline{y} \rightsquigarrow \overline{y} : \overline{T} ; \varphi_3 \\ \overline{\epsilon}_1 \overline{\epsilon}_2, x{:}\mathsf{ref}(T_1'), \overline{y}{:}\overline{T}; x \vdash s \rightsquigarrow s' : T_2 \,!\, \varphi_1 & \varphi_2 = \varphi_1 - \{x\} \end{array}}{\begin{array}{ll} \Gamma \vdash & \mathtt{fun}\langle\overline{\epsilon}_1\rangle(x{:}T_1)\overline{y}\{s\} \\ & \rightsquigarrow \Lambda\overline{\epsilon}_1\overline{\epsilon}_2.\ \mathtt{fun}(x : T_1')\,{}^{\varphi_2}_{T_2}\overline{y{=}y}\{s'\} \end{array} \quad : (\forall\overline{\epsilon}_1\overline{\epsilon}_2.\ T_1' \xrightarrow{\varphi_2} T_2) \,!\, \varphi_3}$$

Type System and Elaboration, Statements

$$\boxed{\Gamma; \overline{z} \vdash s \rightsquigarrow s' : T \,!\, \varphi}$$

$$\frac{\Gamma \vdash e_1 \rightsquigarrow e_1' : \forall \overline{\epsilon}.\ T_1 \xrightarrow{\varphi_4} T_2 \,!\, \varphi_1 \qquad \Gamma \vdash e_2 \rightsquigarrow e_2' : T_3 \,!\, \varphi_2 \qquad \mathit{match}(T_3, [\overline{\epsilon}{:=}\overline{u}]\, T_1) = \{\overline{u} \mapsto \overline{\varphi}\} \qquad \Gamma, x{:}\mathrm{ref}(T_2); x\,\overline{z} \vdash s \rightsquigarrow s' : T_3 \,!\, \varphi_3}{\Gamma; \overline{z} \vdash \begin{array}{l} x{=}e_1(e_2)\,;\ s \\ \rightsquigarrow x{=}e_1'{<}\overline{\varphi}{>}(e_2')\,;\ s' \end{array} : T_3 \,!\, [\overline{\epsilon}{:=}\overline{\varphi}]\varphi_4 \cup \bigcup_{i=1}^{3} \varphi_i}$$

$$\frac{\Gamma \vdash e \rightsquigarrow e' : T \,!\, \varphi \qquad \mathrm{FV}(T) \cap \overline{z} = \emptyset}{\Gamma; \overline{z} \vdash \mathtt{ret}\, e \rightsquigarrow \mathtt{ret}\, e' : T \,!\, \varphi}$$

$$\frac{\Gamma \vdash e_1 \rightsquigarrow e_1' : \forall \overline{\epsilon}.\ T_1 \xrightarrow{\varphi_3} T_2 \,!\, \varphi_1 \qquad \Gamma \vdash e_2 \rightsquigarrow e_2' : T_3 \,!\, \varphi_2 \qquad \mathit{match}(T_3, [\overline{\epsilon}{:=}\overline{u}]\, T_1) = \{\overline{u} \mapsto \overline{\varphi}\} \qquad \mathrm{FV}(T_2) \cap \overline{z} = \emptyset}{\Gamma; \overline{z} \vdash \mathtt{ret}\, e_1(e_2) \rightsquigarrow \mathtt{ret}\, e_1'{<}\overline{\varphi}{>}(e_2') : T_2 \,!\, [\overline{\epsilon}{:=}\overline{\varphi}]\varphi_3 \cup \varphi_1 \cup \varphi_2}$$

Effect Argument Inference, Types

$$
\begin{aligned}
\mathit{match}(\mathtt{Int}, \mathtt{Int}, \Sigma) &= \Sigma \\
\mathit{match}(T_1 \xrightarrow{\varphi_1} T_2, T_3 \xrightarrow{\varphi_2} T_4, \Sigma) &= \Sigma_3 \\
\text{where } \Sigma_1 &= \mathit{match}(T_3, T_1, \Sigma) \\
\Sigma_2 &= \mathit{match}(\varphi_1, \varphi_2, \Sigma_1) \\
\Sigma_3 &= \mathit{match}(T_2, T_4, \Sigma_2) \\
\mathit{match}(\forall \overline{\epsilon}.\ T_1, \forall \epsilon.\ T_2, \Sigma) &= \mathit{match}(T_1, T_2, \Sigma)
\end{aligned}
$$

Effect Argument Inference, Effects

$$
\begin{aligned}
\mathit{match}(\varphi, u, \Sigma) &= \Sigma[u{:=}\varphi] \\
&\quad \text{if } u \notin \mathrm{dom}(\Sigma) \\
\mathit{match}(u, \varphi, \Sigma) &= \Sigma[u{:=}\varphi] \\
&\quad \text{if } u \notin \mathrm{dom}(\Sigma) \\
\mathit{match}(\varphi_1, \varphi_2, \Sigma) &= \Sigma \\
&\quad \text{if } \vdash \Sigma(\varphi_1) \subseteq \Sigma(\varphi_2)
\end{aligned}
$$

Conclusion

- Effects are sets of variables.
- Type system prevents escaping references to the stack.
- Programmer controls capture by reference or by value.

Questions?

Effect Subset

$$\boxed{\vdash \varphi \subseteq \varphi}$$

$$\vdash \emptyset \subseteq \varphi \qquad \vdash \gamma \subseteq \gamma$$

$$\frac{\vdash \varphi_1 \subseteq \varphi_3 \quad \vdash \varphi_2 \subseteq \varphi_3}{\vdash \varphi_1 \cup \varphi_2 \subseteq \varphi_3} \qquad \frac{\vdash \gamma \subseteq \varphi_1 \text{ or } \vdash \gamma \subseteq \varphi_2}{\vdash \gamma \subseteq \varphi_1 \cup \varphi_2}$$

Effect Union

$\boxed{\varphi \Cup \varphi}$

$$\emptyset \Cup \varphi = \varphi$$
$$\varphi \Cup \emptyset = \varphi$$
$$(\varphi_1 \cup \varphi_2) \Cup \varphi_3 = (\varphi_1 \Cup \varphi_3) \Cup (\varphi_2 \Cup \varphi_3)$$
$$\gamma \Cup (\varphi_1 \cup \varphi_2) = (\gamma \Cup \varphi_1) \Cup (\gamma \Cup \varphi_2)$$
$$\gamma_1 \Cup \gamma_2 = \begin{cases} \gamma_1 & \text{if } \gamma_1 = \gamma_2 \\ \gamma_1 \cup \gamma_2 & \text{otherwise} \end{cases}$$

$$[\epsilon{:=}\varphi_3](\varphi_1 \Cup \varphi_2) = [\epsilon{:=}\varphi_3]\varphi_1 \Cup [\epsilon{:=}\varphi_3]\varphi_2$$

Effect Intersection

$\boxed{\varphi \cap \varphi}$

$$(\varphi_1 \cup \varphi_2) \cap \varphi_3 = (\varphi_1 \cap \varphi_3) \Cup (\varphi_2 \cap \varphi_3)$$

$$\gamma \cap (\varphi_2 \cup \varphi_3) = (\gamma \cap \varphi_2) \Cup (\gamma \cap \varphi_3)$$

$$\gamma_1 \cap \gamma_2 = \begin{cases} \gamma_1 & \text{if } \gamma_1 = \gamma_2 \\ \emptyset & \text{if } \gamma_1 \neq \gamma_2 \end{cases}$$

$$\emptyset \cap \varphi = \emptyset$$

$$\varphi \cap \emptyset = \emptyset$$

Effect Difference

$\boxed{\varphi - \varphi}$

$$(\varphi_1 \cup \varphi_2) - \varphi_3 = (\varphi_1 - \varphi_3) \Cup (\varphi_2 - \varphi_3)$$

$$\gamma - (\varphi_2 \cup \varphi_3) = (\gamma - \varphi_2) - \varphi_3$$

$$\gamma_1 - \gamma_2 = \begin{cases} \emptyset & \text{if } \gamma_1 = \gamma_2 \\ \gamma_1 & \text{otherwise} \end{cases}$$

$$\varphi - \emptyset = \varphi$$

$$\emptyset - \varphi = \emptyset$$