Linear Destination Passing as a Modular Semantic Framework

Frank Pfenning

Carnegie Mellon University

ConCert Meeting, May 2003

Warning: Work in progress!

Joint work with Kevin Watkins

Natural Semantics

- Typing judgment $e : \tau$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \mathrm{fn}\, x.e : \tau_1 \longrightarrow \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \longrightarrow \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau_1}$$

- Evaluation judgment $e \hookrightarrow v$

$$\frac{}{\mathrm{fn}\, x.e \hookrightarrow \mathrm{fn}\, x.e}$$

$$\frac{e_1 \hookrightarrow \mathrm{fn}\, x.e_1' \quad e_2 \hookrightarrow v_2 \quad [v_2/x]e_1' \hookrightarrow v}{e_1\ e_2 \hookrightarrow v}$$

Non-Modular Extensions

- Example: mutable store

$$\frac{}{\langle s, \mathrm{fn}\, x.e \rangle \hookrightarrow \langle s, \mathrm{fn}\, x.e \rangle} \qquad \frac{\begin{array}{c}\langle s_1, e_1 \rangle \hookrightarrow \langle s_2, \mathrm{fn}\, x.e'_1 \rangle \\ \langle s_2, e_2 \rangle \hookrightarrow \langle s_3, v_2 \rangle \\ \langle s_3, [v_2/x]e'_1 \rangle \hookrightarrow \langle s_4, v \rangle\end{array}}{\langle s_1, e_1\, e_2 \rangle \hookrightarrow \langle s_4, v \rangle}$$

- Others: exceptions, continuations, futures, etc.
- More abstract, *modular* presentation?
- Exploit substructural (linear) judgments!

Linear Destination-Passing

- New semantic presentation: *Linear Destination-Passing* (LDP)
- Usually: dest-passing as a compiler optimization
- Here: destinations $d{:}\tau$ as names for values
- Frames $f{:}\tau$ for intermediate states
- Basic judgments
  - $e \mapsto d$ evaluate e with destination d
  - $f \rightarrowtail d$ compute f with destination d
  - $d{=}v$ value of destination d is v

Linear Destination-Passing

- Judgment form

$$H ::= \cdot \mid e \mapsto d, H \mid f \rightarrowtail d, H \mid d{=}v, H$$

- Linear, but order is irrelevant
- Overall deduction and value rule

$$\begin{array}{c} \overline{d_0{=}v, \cdot} \\ \vdots \\ e \mapsto d_0, \cdot \end{array} \qquad \frac{d{=}v, H}{v \mapsto d, H}$$

Terminology of Substructural Logic

- J is *linear* in J, H if J must be used *exactly once*
- J is *affine* in J, H if J can used *at most once*
- J is *unrestricted* in J, H if J can be used *arbitrarily many times*
- [J is *strict* in J, H must be used *at least once*]
- Order is never important in this talk (future work)

LDP Examples

- This talk:
  - Sequential evaluation
  - Parallel application
  - Futures
  - Continuations
  - Mutable references
  - Call-by-need
  - Exceptions
  - Heaps
  - π-Calculus

Design Criteria

- Modularity
  - Do not revise earlier specifications
- Orthogonality
  - No cross-references between features
- Substructural properties
  - Which judgments are linear, affine, unrestricted

Sequential Evaluation

- Abstractions handled by value rule
- Applications (new parameters noted $[-]$)

$$\frac{e_1 \mapsto d_1, d_1\, e_2 \rightarrowtail d, H}{e_1\, e_2 \mapsto d, H}\ [d_1]$$

$$\frac{e_2 \mapsto d_2, d_1{=}v_1, d_1\, d_2 \rightarrowtail d, H}{d_1{=}v_1, d_1\, e_2 \rightarrowtail d, H}\ [d_2]$$

$$\frac{[v_2/x]e_1' \mapsto d, H}{d_2{=}v_2, d_1{=}(\mathrm{fn}\, x.e_1'), d_1\, d_2 \rightarrowtail d, H}$$

Parallel Application

- Execute function and argument in parallel
- Replace application rules by:

$$\frac{e_1 \mapsto d_1, e_2 \mapsto d_2, d_1\, d_2 \rightarrowtail d, H}{e_1\, e_2 \mapsto d, H}\,[d_1,d_2]$$

$$\frac{[v_2/x]e_1' \mapsto d, H}{d_1{=}(\mathrm{fn}\, x.e_1'), d_2{=}v_2, d_1\, d_2 \rightarrowtail d, H}$$

Futures

- Futures as in Multilisp [Halstead'85]
- Use destinations p as *promises*
- New expression

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathrm{future}\, e : \tau}$$

- New value form $\mathrm{promise}\, p : \tau$ for $p{:}\tau$

Futures

- Use destinations p as *promises*

$$\frac{e_1 \mapsto p_1, d{=}\mathrm{promise}\, p_1, H}{\mathrm{future}\, e_1 \mapsto d, H}\,[p_1]$$

$$\frac{d'{=}v_1, p_1{=}v_1, H}{d'{=}\mathrm{promise}\, p_1, p_1{=}v_1, H}$$

- Form $p_1{=}v_1$ must be *affine*
- Will remain throughout the computation

Continuations

- New expressions

$$\frac{\Gamma, k{:}\tau\,\mathrm{cont} \vdash e : \tau}{\Gamma \vdash \mathrm{callcc}\, k.e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau\,\mathrm{cont} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathrm{throw}\, e_1\, e_2 : \tau'}$$

- Use destinations d as *continuations*
- New values $\mathrm{cont}\, d : \tau\,\mathrm{cont}$ for $d{:}\tau$

Continuations

- Use destinations d as *continuations*

$$\frac{[\mathrm{cont}\, d/k]e_1 \mapsto d, H}{\mathrm{callcc}\, k.e_1 \mapsto d, H}$$

$$\frac{e_1 \mapsto d_1, \mathrm{throw}\, d_1\, e_2 \rightarrowtail d, H}{\mathrm{throw}\, e_1\, e_2 \mapsto d, H}\,[d_1]$$

$$\frac{e_2 \mapsto d_2, H}{d_1{=}\mathrm{cont}\, d_2, \mathrm{throw}\, d_1\, e_2 \rightarrowtail d, H}$$

- All computations $f \rightarrowtail d$ must be *unrestricted*
- Reflects non-linear use of control stack

Mutable References

- New expressions

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathrm{ref}\, e : \tau\,\mathrm{ref}} \qquad \frac{\Gamma \vdash e : \tau\,\mathrm{ref}}{\Gamma \vdash \,!\,e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau\,\mathrm{ref} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash (e_1 := e_2) : \mathrm{unit}}$$

- New values $\mathrm{cell}\, l : \tau\,\mathrm{ref}$ for $l{:}\tau$

Mutable References

- Use destinations l as mutable locations
- Allocate $(\mathrm{ref}\, e_1)$ and read $(!\, e_1)$

$$\frac{e_1 \mapsto d_1, \mathrm{ref}\ d_1 \rightarrowtail d, H}{\mathrm{ref}\ e_1 \mapsto d, H}\,[d_1] \qquad \frac{d{=}\mathrm{cell}\, l_1, H, l_1{=}v_1}{d_1{=}v_1, \mathrm{ref}\ d_1 \rightarrowtail d, H}\,[l_1]$$

$$\frac{e_1 \mapsto d_1, !\, d_1 \rightarrowtail d, H}{!\, e_1 \mapsto d, H}\,[d_1] \qquad \frac{d{=}v, H, l{=}v}{d_1{=}\mathrm{cell}\, l, !\, d_1 \rightarrowtail d, H, l{=}v}$$

- Form $l{=}v$ must be *affine*

Mutable References

- Write ($e_1 := e_2$)

$$\frac{e_1 \mapsto d_1, (d_1 := e_2) \rightarrowtail d, H}{(e_1 := e_2) \mapsto d, H} [d_1]$$

$$\frac{e_2 \mapsto d_2, d_1{=}v_1, (d_1 := d_2) \rightarrowtail d, H}{d_1{=}v_1, (d_1 := e_2) \rightarrowtail d, H} [d_2]$$

$$\frac{d{=}\langle\,\rangle, H, l_1{=}v_2}{d_2{=}v_2, d_1{=}\mathrm{cell}\, l_1, (d_1 := d_2) \rightarrowtail d, H, l_1{=}v_1}$$

Call-by-Need (Lazy Evaluation)

- Thunks as destinations t

$$\frac{e_1 \mapsto d_1, d_1\, e_2 \rightarrowtail d, H}{e_1\, e_2 \mapsto d, H}\,[d_1]$$

$$\frac{[\mathrm{force}\, t_2/x]e_1' \mapsto d, H, \mathrm{delay}\, e_2 \rightarrowtail t_2}{d_1{=}(\mathrm{fn}\, x.e_1'), d_1\, e_2 \rightarrowtail d, H}\,[t_2]$$

$$\frac{e_2 \mapsto t_2, \mathrm{force}\, t_2 \rightarrowtail d, H}{\mathrm{force}\, t_2 \mapsto d, H, \mathrm{delay}\, e_2 \rightarrowtail t_2} \qquad \frac{d{=}v_2, H, t_2{=}v_2}{t_2{=}v_2, \mathrm{force}\, t_2 \rightarrowtail d, H}$$

- Form $\mathrm{delay}\, e \rightarrowtail t$ is *affine*, $t{=}v$ *unrestricted*

Exceptions

- New expressions

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathrm{try}\, e_1\, e_2 : \tau}$$

$$\frac{}{\Gamma \vdash \mathrm{fail}\ : \tau}$$

- No new values
- For single-threaded computation

Exceptions

- New frame handle for currently active handler

$$\frac{e_1 \mapsto d_1, \mathrm{try}\, d_1\, e_2\, d' \rightarrowtail d, H, \mathrm{handle} \rightarrowtail d_1}{\mathrm{try}\, e_1\, e_2 \mapsto d, H, \mathrm{handle} \rightarrowtail d'}\,[d_1]$$

$$\frac{d{=}v_1, H, \mathrm{handle} \rightarrowtail d'}{d_1{=}v_1, \mathrm{try}\, d_1\, e_2\, d' \rightarrowtail d, H, \mathrm{handle} \rightarrowtail d_1}$$

$$\frac{e_2 \mapsto d, H, \mathrm{handle} \rightarrowtail d'}{\mathrm{fail} \mapsto d'', \mathrm{try}\, d_1\, e_2\, d' \rightarrowtail d, H, \mathrm{handle} \rightarrowtail d_1}$$

- Form $\mathrm{handle} \rightarrowtail d$ must exist initially and fi nally

Heap Semantics

- Use destinations h as heap locations

$$\frac{e_1 \mapsto d_1, d_1\, e_2 \rightarrowtail d, H}{e_1\, e_2 \mapsto d, H}\,[d_1] \qquad \frac{d{=}v, H, h{=}v}{h \mapsto d, H, h{=}v}$$

$$\frac{e_2 \mapsto d_2, d_1{=}v_1, d_1\, d_2 \rightarrowtail d, H}{d_1{=}v_1, d_1\, e_2 \rightarrowtail d, H}\,[d_2]$$

$$\frac{[h_2/x]e_1' \mapsto d, H, h_2{=}v_2}{d_2{=}v_2, d_1{=}(\mathrm{fn}\, x.e_1'), d_1\, d_2 \rightarrowtail d, H}\,[h_2]$$

- Form $h{=}v$ must be *unrestricted*

π-Calculus

- Names a, b, variables x, y
- Syntax

$$
\begin{array}{rrcll}
\text{Procs} & P & ::= & (P_1|P_2) \mid 0 \mid \mathrm{new}\, x.P \mid !P \mid M & \\
\text{Sums} & M & ::= & G \mid M_1 + M_2 & \\
\text{Guards} & G & ::= & \bar{a}\langle b\rangle.P & \text{output} \\
 & & \mid & a(x).P & \text{input} \\
 & & \mid & \tau.P & \text{silent action}
\end{array}
$$

- Structural congruence as properties of H

Judgments

- $\mathrm{proc}(P)$ process P
- Auxiliary judgments
  - $\mathrm{sync}_2(M, N) \to (P, Q)$ synch M and N yields P and Q
  - $\mathrm{sync}_1(M) \to P$ silent action on M yields P
- Auxiliary judgments not part of state H

Process Expressions

- Process evolution

$$\frac{\text{proc}(P), \text{proc}(Q), H}{\text{proc}(P|Q), H} \qquad \frac{H}{\text{proc}(0), H}$$

$$\frac{\text{proc}([a/x]P), H}{\text{proc}(\text{new}\, x.P), H}\,[a] \qquad \frac{\text{proc}^*(P), H}{\text{proc}(!P), H}$$

- Form $\text{proc}^*(P)$ is *unrestricted* from of $\text{proc}(P)$

Communication

- Synchronization

$$\frac{\mathrm{sync}_2(M, N) \longrightarrow (P, Q) \qquad \mathrm{proc}(P), \mathrm{proc}(Q), H}{\mathrm{proc}(M), \mathrm{proc}(N), H}$$

$$\frac{\mathrm{sync}_1(M) \longrightarrow P \qquad \mathrm{proc}(P), H}{\mathrm{proc}(M), H}$$

- Communication action

$$\frac{}{\mathrm{sync}_2(\bar{a}\langle b\rangle.P, a(x).Q) \longrightarrow (P, [b/x]Q)}$$

Synchronization

- Choice in synchronization

$$\frac{\mathrm{sync}_2(M_1, N) \longrightarrow (P, Q)}{\mathrm{sync}_2(M_1 + M_2, N) \longrightarrow (P, Q)} \quad \frac{\mathrm{sync}_2(M_2, N) \longrightarrow (P, Q)}{\mathrm{sync}_2(M_1 + M_2, N) \longrightarrow (P, Q)}$$

$$\frac{\mathrm{sync}_2(M, N_1) \longrightarrow (P, Q)}{\mathrm{sync}_2(M, N_1 + N_2) \longrightarrow (P, Q)} \quad \frac{\mathrm{sync}_2(M, N_2) \longrightarrow (P, Q)}{\mathrm{sync}_2(M, N_1 + N_2) \longrightarrow (P, Q)}$$

- Don't-know vs don't-care nondeterminism:

$$(\bar{a}\langle b\rangle.P_1 + \bar{b}\langle a\rangle.P_2)|(a(x).Q_1 + b(y).Q_2)$$

has exactly two actions, no deadlock

Silent Action

- One-way synchronization

$$\frac{}{\mathrm{sync}_1(\tau.P) \longrightarrow P}$$

$$\frac{\mathrm{sync}_1(M_1) \longrightarrow P}{\mathrm{sync}_1(M_1 + M_2) \longrightarrow P} \qquad \frac{\mathrm{sync}_1(M_2) \longrightarrow P}{\mathrm{sync}_1(M_1 + M_2) \longrightarrow P}$$

Other Examples

- ML and Haskell language families
  - Products, disjoint sums, recursive types, . . .
  - Concurrent ML
- Concurrent calculi
  - Petri nets
  - Asynchronous π-calculus
  - Multiset Rewriting (MSR)

Other Modular Approaches

- Monadic Metalanguage [Moggi'89]
  - Insulate effects *inside* the language
- Contextual semantics [Wright & Felleisen'92]
  - Well-suited for continuations
  - Not as appropriate for concurrency?
- MSOS [Mosses'02]
  - Small-step *structured operational semantics*
  - Add effect annotations
  - Not as flexible or modular in effect notation

Future Work: More Examples

- Spatial computation [Cardelli & Gordon'98] [Moody'03]
  - Index destinations by location
- Other concurrent calculi (action, join)
- Garbage collection
  - Index destinations by to-space or from-space
- Saturation-based procedures [MacAllester,Ganzinger]
- Protocols [Cervesato] [Bozzano'02]

Future Work: Implementation

- Linear Destination Passing reverse engineered from Concurrent Logical Framework!
- With minor changes, all examples here can be readily implemented in CLF . . .
- . . . when an implementation of CLF exists
- Issues
  - Executing LDP using CLF operational semantics
  - Interleaving don't-know (search) and don't-care (concurrency) non-determinism
  - Representation of meta-theoretic proofs

Future Work: Order

- Divide H into ordered and unordered zone
- Adjacent interaction only for ordered zone
  - Representation of stacks
  - Direct mapping of abstract machines
- Simpler meta-theory(?)

Future Work: Slick Proofs

- Best formulation of meta-theoretic properties?
  - Type preservation
  - Progress
  - Termination
  - Infinite computations
- Some modularity of proofs?

CLF Example: Parallel Application

- Application rule in LDP

$$\frac{e_1 \mapsto d_1, e_2 \mapsto d_2, d_1\, d_2 \rightarrowtail d, H}{e_1\, e_2 \mapsto d, H}\,[d_1, d_2]$$

- Representation in CLF (omitting rule name)

$$\begin{array}{l} \mathsf{eval}\ (\mathsf{app}\ \mathrm{E}_1\ \mathrm{E}_2)\ \mathrm{D} \\ \quad \multimap \{\exists \mathsf{d}_1.\ \exists \mathsf{d}_2.\ \mathsf{eval}\ \mathrm{E}_1\ \mathsf{d}_1 \otimes \mathsf{eval}\ \mathrm{E}_2\ \mathsf{d}_2 \otimes \mathsf{comp}\ (\mathsf{app}_2\ \mathsf{d}_1\ \mathsf{d}_2)\ \mathrm{D}\} \end{array}$$

CLF Example: Parallel Application

- Frame rule in LDP

$$\frac{[v_2/x]e'_1 \mapsto d, H}{d_1{=}(\mathrm{fn}\, x.e'_1), d_2{=}v_2, d_1\, d_2 \rightarrowtail d, H}$$

- Representation in CLF (omitting rule name)

$$\begin{array}{l}\mathsf{is}\ \mathrm{D}_1\ (\mathsf{fun}\ (\lambda\mathsf{x}.\ \mathrm{E}'_1\ \mathsf{x})) \multimap \mathsf{is}\ \mathrm{D}_2\ \mathrm{V}_2 \multimap \mathsf{comp}\ (\mathsf{app}_2\ \mathrm{D}_1\ \mathrm{D}_2)\ \mathrm{D} \\ \quad \multimap \{\mathsf{eval}\ (\mathrm{E}'_1\ \mathrm{V}_2)\ \mathrm{D}\}\end{array}$$

CLF Example: Futures

- Recall specifi cation in LDP ($p{=}v$ *affine*)

$$\frac{e_1 \mapsto p_1, d{=}\mathrm{promise}\, p_1, H}{\mathrm{future}\, e_1 \mapsto d, H}\,[p_1] \qquad \frac{d'{=}v_1, p_1{=}v_1, H}{d'{=}\mathrm{promise}\, p_1, p_1{=}v_1, H}$$

- Encoding in CLF

$$\mathsf{evf} : \mathsf{eval}\ (\mathsf{future}\ \mathrm{E}_1)\ \mathrm{D} \multimap \{\exists \mathsf{p}_1.\, \mathsf{eval}\ \mathrm{E}_1\ \mathsf{p}_1 \otimes \mathsf{is}\ \mathrm{D}\ (\mathsf{promise}\ \mathsf{p}_1)\}$$

$$\mathsf{evp} : \mathsf{is}\ \mathrm{D}'\ (\mathsf{promise}\ \mathrm{P}_1) \multimap \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1 \multimap \{\mathsf{is}\ \mathrm{D}'\ \mathrm{V}_1 \otimes \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1\}$$

- Goal $\{(\exists \mathsf{v}.\, \mathsf{is}\ \mathsf{d}_0\ \mathsf{v}) \otimes \top\}$ for affi ne hypotheses

CLF Example: Futures Alternative

- Explicit step from linear to unrestricted dests.

$$\frac{e_1 \mapsto d_1, \text{deliver}\, d_1 \rightarrowtail p_1, d{=}\text{promise}\, p_1, H}{\text{future}\, e_1 \mapsto d, H}\ [d_1,p_1]$$

$$\begin{array}{l} \mathsf{eval}\ (\mathsf{future}\ \mathrm{E}_1)\ \mathrm{D} \multimap \{\exists \mathsf{d}_1.\ \exists \mathsf{p}_1. \\ \qquad \mathsf{eval}\ \mathrm{E}_1\ \mathsf{d}_1 \otimes \mathsf{comp}\ (\mathsf{deliver}\ \mathsf{d}_1)\ \mathsf{p}_1 \otimes \mathsf{is}\ \mathrm{D}\ (\mathsf{promise}\ \mathsf{p}_1)\}. \end{array}$$

CLF Example: Futures Alternative

- Creating unrestricted destinations (note !)

$$\frac{H, p_1 = v_1}{d_1 = v_1, \mathrm{deliver}\, d_1 \rightarrowtail p_1, H}$$

is $\mathrm{D}_1\ \mathrm{V}_1 \multimap$ comp $($deliver $\mathrm{D}_1)\ \mathrm{P}_1 \multimap \{!$ is $\mathrm{P}_1\ \mathrm{V}_1\}$.

- Using unrestricted destinations (note $\rightarrow$)

$$\frac{d' = v_1, H, p_1 = v_1}{d' = \mathrm{promise}\, p_1, H, p_1 = v_1}$$

is D' $($promise $\mathrm{P}_1) \multimap$ is $\mathrm{P}_1\ \mathrm{V}_1 \rightarrow \{$is $\mathrm{D}'\ \mathrm{V}_1\}$.

CLF Example: Continuations

- LDP specifi cation ($f \rightarrowtail d$ *unrestricted*)

$$\frac{[\mathrm{cont}\, d/k]e_1 \mapsto d, H}{\mathrm{callcc}\, k.e_1 \mapsto d, H} \qquad \frac{e_1 \mapsto d_1, \mathrm{throw}\, d_1\, e_2 \rightarrowtail d, H}{\mathrm{throw}\, e_1\, e_2 \mapsto d, H}[d_1]$$

$$\frac{e_2 \mapsto d_2, H}{d_1{=}\mathrm{cont}\, d_2, \mathrm{throw}\, d_1\, e_2 \rightarrowtail d, H}$$

- CLF encoding

$\mathsf{eval}\ (\mathsf{callcc}\ (\lambda \mathsf{k}.\, \mathrm{E}_1\ \mathsf{k}))\ \mathrm{D} \multimap \{\mathsf{eval}\ (\mathrm{E}_1\ (\mathsf{cont}\ \mathrm{D}))\ \mathrm{D}\}.$

$\mathsf{eval}\ (\mathsf{throw}\ \mathrm{E}_1\ \mathrm{E}_2)\ \mathrm{D}$
$\quad \multimap \{\exists \mathsf{d}_1.\, \mathsf{eval}\ \mathrm{E}_1\ \mathsf{d}_1 \otimes\, !\, \mathsf{comp}\ (\mathsf{throw}\ \mathsf{d}_1\ \mathrm{E}_2)\ \mathrm{D}\}.$

$\mathsf{is}\ \mathrm{D}_1\ (\mathsf{cont}\ \mathrm{D}_2) \multimap \mathsf{comp}\ (\mathsf{throw}\ \mathrm{D}_1\ \mathrm{E}_2)\ \mathrm{D} \rightarrow \{\mathsf{eval}\ \mathrm{E}_2\ \mathrm{D}_2\}.$

Summary: LDP

- *Linear Destination Passing* as uniform and modular semantic framework for functional, imperative, and concurrent languages
- Structural properties
  - *Linear* for concurrent computation
  - *Affine* for store
  - *Unrestricted* for memoization, continuations
- Readily specified in CLF