Survey Workshop on Typeful Compilation

Garbage Collection Based on a Linear Type System

Atsushi Igarashi and Naoki Kobayashi,
Workshop on Types in Compilation, 2000.

Ahn, Joonseon

ROPAS, KAIST

2000. 12. 9 (Saturday)

1. Introduction

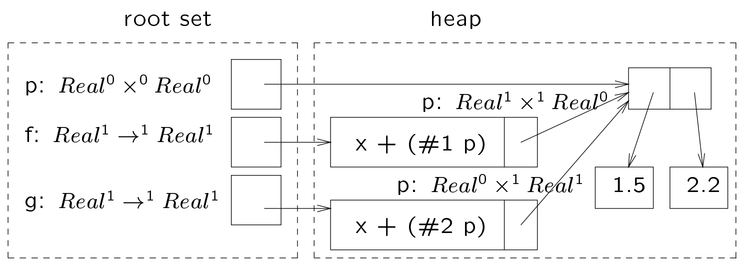
□ Linear Type System

- type + usage

$$\frac{\begin{array}{c} x : Real^1 \times^1 Real^0 \vdash \sharp 1\ x : Real^1 \\ x : Real^0 \times^1 Real^1 \vdash \sharp 2\ x : Real^1 \end{array}}{x : Real^{1+0} \times^{1+1} Real^{0+1} \vdash \sharp 1\ x + \sharp 2\ x : Real^{\omega}}$$

- problem with GC
  - dangling pointers
  - multiple traverses on one heap value
    (ex)

```
let val p = (1.5,2.2)
    fun f x = x + (#1 p)
fun g x = x + (#2 p)
in f 3.1 + g 4.3 end
```



□ Contribution

- GC algorithm for a language with a monomorphic linear type system
- Correctness proof of GC

□ Contents

- Introduction
- Target language and its operational semantics
- Type system of the target language
- GC algorithm and a correctness proof
- Conclusion

2. Language λ^{κ}_{gc}

□ Syntax

$$
\begin{array}{lcll}
\kappa & ::= & 0 \mid 1 \mid \omega & (\mathit{uses}) \\
\tau & ::= & \mathit{Int} \mid \tau_1 \to^{\kappa} \tau_2 \mid \tau_1 \times^{\kappa} \tau_2 & (\mathit{types}) \\
\\
v & ::= & x \mid n & \\
d & ::= & v \mid (v_1, v_2)^{\kappa} \mid {}^{\langle\Gamma\rangle}(\texttt{fun } x(y) = e)^{\kappa} & \\
e & ::= & {}^{\langle\Gamma\rangle}\texttt{let } x = d \texttt{ in } e \mid {}^{\langle\Gamma\rangle}\texttt{let } x = y\ z \texttt{ in } e & \\
 & \mid & {}^{\langle\Gamma\rangle}\texttt{let } x = y + z \texttt{ in } e \mid {}^{\langle\Gamma\rangle}\texttt{let } (x, y) = z \texttt{ in } e & \\
 & \mid & {}^{\langle\Gamma\rangle}\texttt{if0 } x \texttt{ then } e_1 \texttt{ else } e_2 \mid {}^{\langle\Gamma\rangle}v &
\end{array}
$$

- All temporal results are bound to variables and the evaluation order is explicit.
- Each heap-allocated value is associated with a use.
- Expressions and functions are annotated with type information.
- Type and use notations are automatically written by compiler [Iga97,IK00b,...].

□ Operational Semantics

- Definitions
  - type environment (Γ) : $Var \to type$
  - environment (V) : $Var \to$ (non-heap values)
  - heap values

$$h ::= (v_1, v_2) \mid (V, {}^{\langle\Gamma\rangle}\texttt{fun}\ x(y) = e)$$

  - heap (H) : $Var \to$ (pairs of a heap value and a use(h^κ))
  - A stack(S) is a sequence whose elements have the form $[V, \Gamma, \lambda x.e]$, called a stack frame.
  - programs (P) : (H, S, V, e).

- Auxiliary functions

$$(V_1 \uplus V_2)(x) = V_i(x), \text{for } x \in dom(V_i),\ dom(V_1) \cap dom(V_2) = \phi$$

$$H \oplus \{x = h^\kappa\} = \begin{cases} H & (\kappa = 0,\ x \notin dom(H)) \\ H \uplus \{x = h^k\} & (\kappa \neq 0,\ x \notin dom(H)) \\ undefined & (\text{otherwise}) \end{cases}$$

$$\begin{array}{lcl} (H \uplus \{x = h^1\})^{-x} & = & H \\ (H \uplus \{x = h^\omega\})^{-x} & = & (H \uplus \{x = h^\omega\}) \end{array}$$

$$\begin{array}{lcl} \widehat{V}(n) & = & n \\ \widehat{V}(x) & = & V(x) \end{array}$$

- Reduction Rules : $P \longmapsto P'$

$$(H, S, V, {}^{\langle\Gamma\rangle}\mathbf{let}\ x = v\ \mathbf{in}\ e) \longmapsto (H, S, V \uplus \{x = \widehat{V}(v)\}, e) \qquad (\text{R-VAL})$$

$$\frac{z \text{ fresh}}{\begin{array}{c}(H, S, V, {}^{\langle\Gamma\rangle}\mathbf{let}\ x = (v_1, v_2)^\kappa\ \mathbf{in}\ e') \\ \longmapsto (H \oplus \{z = (\widehat{V}(v_1), \widehat{V}(v_2))^\kappa\}, S, V \uplus \{x = z\}, e')\end{array}} \qquad (\text{R-PAIR})$$

$$\frac{z \text{ fresh}}{\begin{array}{c}(H, S, V, {}^{\langle\Gamma\rangle}\mathbf{let}\ x = ({}^{\langle\Gamma_0\rangle}\mathbf{fun}\, y(w) = e_0)^\kappa\ \mathbf{in}\ e') \\ \longmapsto (H \oplus \{z = (V, {}^{\langle\Gamma_0\rangle}\mathbf{fun}\, y(w) = e_0)^\kappa\}, S, V \uplus \{x = z\}, e')\end{array}} \qquad (\text{R-FUN})$$

$$\frac{H(V(y)) = (V_0, {}^{\langle\Gamma_0\rangle}\mathbf{fun}\, z(w) = e_0)^\kappa}{\begin{array}{c}(H, S, V, {}^{\langle\Gamma\rangle}\mathbf{let}\ x = y\, v\ \mathbf{in}\ e) \\ \longmapsto (H^{-V(y)}, S[V, TE(e) \setminus \{x\}, \lambda x.e], V_0 \uplus \{z = V(y), w = \widehat{V}(v)\}, e_0)\end{array}} \qquad (\text{R-APP})$$

$$\frac{n_1 = \widehat{V}(v_1) \qquad n_2 = \widehat{V}(v_2)}{(H, S, V, {}^{\langle\Gamma\rangle}\mathbf{let}\ x = v_1 + v_2\ \mathbf{in}\ e) \longmapsto (H, S, V \uplus \{x = \underline{n_1 + n_2}\}, e)} \qquad (\text{R-PLUS})$$

$$\frac{H(V(z)) = (v_1, v_2)^\kappa}{(H, S, V, {}^{\langle\Gamma\rangle}\mathbf{let}\ (x, y) = z\ \mathbf{in}\ e) \longmapsto (H^{-V(z)}, S, V \uplus \{x = v_1, y = v_2\}, e)} \qquad (\text{R-EXT})$$

$$\frac{\widehat{V}(v) = 0}{(H, S, V, {}^{\langle\Gamma\rangle}\mathbf{if0}\ v\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2) \longmapsto (H, S, V, e_1)} \qquad (\text{R-IFT})$$

$$\frac{\widehat{V}(v) \neq 0}{(H, S, V, {}^{\langle\Gamma\rangle}\mathbf{if0}\ v\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2) \longmapsto (H, S, V, e_2)} \qquad (\text{R-IFF})$$

$$(H, S[V_0, \Gamma_0, \lambda x.e_0], {}^{\langle\Gamma\rangle}v) \longmapsto (H, S, V_0 \uplus \{x = \widehat{V}(v)\}, e_0) \qquad (\text{R-RET})$$

Fig. 2. $\lambda^\kappa_{\text{gc}}$ reduction rules

3. Type System

□ Notational Preliminaries

Definition 1 $\geq$
$\omega \geq 1 \geq 0$

$$\begin{array}{rcll} Int & \geq & Int & \\ \tau_1 \rightarrow^{\kappa_1} \tau_2 & \geq & \tau_1 \rightarrow^{\kappa_2} \tau_2 & \textit{if } \kappa_1 \geq \kappa 2 \\ \tau_{11} \times^{\kappa_1} \tau_{12} & \geq & \tau_{21} \times^{\kappa_2} \tau_{22} & \textit{if } \kappa_1 \geq \kappa_2 \textit{ and } \tau_{1i} \geq \tau_{2i} \textit{ for } i = 1,2 \end{array}$$

Definition 2 *Summation of uses, types and type environments*

$$0+0=0, \quad 1+0=0+1=1,$$
$$1+1=1+\omega=\omega+1=\omega+\omega=\omega$$

$$\begin{array}{lcl} Int + Int & = & Int \\ (\tau_1 \rightarrow^{\kappa_1} \tau_2) + (\tau_1 \rightarrow^{\kappa_2} \tau_2) & = & \tau_1 \rightarrow^{\kappa_1+\kappa_2} \tau_2 \\ (\tau_{11} \times^{\kappa_1} \tau_{12}) + (\tau_{21} \times^{\kappa_2} \tau_{22}) & = & (\tau_{11}+\tau_{21}) \times^{\kappa_1+\kappa_2} (\tau_{12}+\tau_{22}) \end{array}$$

$$(\Gamma_1 + \Gamma_2)(x) = \begin{cases} \Gamma_1(x) + \Gamma_2(x) & \textit{if } x \in dom(\Gamma_1) \cap dom(\Gamma_2) \\ \Gamma_i(x) & \textit{else if } x \in dom(\Gamma_i) \end{cases}$$

Definition 3 *Product of uses, types and type environments*

$$0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0, \ 1 \cdot 1 = 1,$$
$$1 \cdot \omega = \omega \cdot 1 = \omega \cdot \omega = \omega$$

$$\begin{array}{lcl} \kappa \cdot Int & = & Int \\ \kappa \cdot (\tau_1 \rightarrow^{\kappa'} \tau_2) & = & \tau_1 \rightarrow^{\kappa \cdot \kappa'} \tau_2 \\ \kappa \cdot (\tau_1 \times^{\kappa'} \tau_2) & = & (\kappa \cdot \tau_1) \times^{\kappa \cdot \kappa'} (\kappa \cdot \tau_2) \end{array}$$

$$(\kappa \cdot \Gamma)(x) = \kappa \cdot (\Gamma(x))$$

Definition 4 *Truncation*
$$\lfloor \Gamma \rfloor \ = \ \Gamma \setminus \{x | \Gamma(x) = \tau_1 \times^0 \tau_2 \textit{ or } \tau_1 \rightarrow^0 \tau_2\}$$

□ Typing Rules

- Typing Rules for Expressions : $\Gamma \vdash e : \tau$
- Typing Rules for Programs

$$\begin{array}{l} \Gamma \vdash V : \Gamma' \\ \Gamma \vdash h^\kappa : \tau \\ \Gamma \vdash S : \tau_1 \rightarrow^1 \tau_2 \\ \Gamma \vdash H : \Gamma_1; \Gamma_2 \\ \Gamma \vdash P : \tau \end{array}$$

□ Soundness of Type System

Theorem 1 [*Subject Reduction*]
If $\phi \vdash P : \tau$ and $P \longmapsto P'$, then $\phi \vdash P' : \tau$.

Theorem 2 [*Type Soundness*]
If $\vdash P : \tau$ and $P \longmapsto^* P'$ with P' being a normal form, then P' is an answer and $\vdash P' : \tau$.

Non-heap values, pairs, functions:

$$x{:}\,\tau \vdash x : \tau \qquad \text{(T-Var)} \qquad\qquad \Gamma \vdash n : \mathit{Int} \qquad \text{(T-Int)}$$

$$v_1{:}\,\tau_1 + v_2{:}\,\tau_2 \vdash (v_1, v_2)^\kappa : \tau_1 \times^\kappa \tau_2 \qquad \text{(T-Pair)}$$

$$\frac{\Gamma_1, y{:}\,\tau_1 \to^{\kappa_1} \tau_2, z{:}\,\tau_1 \vdash e_0 : \tau_2 \qquad \kappa = \kappa_2 \cdot (\kappa_1 + 1)}{\kappa \cdot \Gamma_1 \vdash ({}^{\langle \Gamma_1 \rangle}\mathbf{fun}\, y(z) = e_0)^\kappa : \tau_1 \to^{\kappa_2} \tau_2} \qquad \text{(T-Fun)}$$

Expressions:

$$\frac{\Gamma_1 \vdash d : \tau' \qquad \Gamma_2, x{:}\,\tau' \vdash e : \tau}{\Gamma_1 + \Gamma_2 \vdash {}^{\langle \Gamma_1 + \Gamma_2 \rangle}\mathbf{let}\ x = d\ \mathbf{in}\ e : \tau} \qquad \text{(T-Dec)}$$

$$\frac{\Gamma', x{:}\,\tau_2 \vdash e : \tau \qquad \Gamma = \Gamma' + (y{:}\,\tau_1 \to^1 \tau_2) + (v{:}\,\tau_2)}{\Gamma \vdash {}^{\langle \Gamma \rangle}\mathbf{let}\ x = y\, v\ \mathbf{in}\ e : \tau} \qquad \text{(T-App)}$$

$$\frac{\Gamma', x{:}\,\mathit{Int} \vdash e : \tau \qquad \Gamma = \Gamma' + v_1{:}\,\mathit{Int} + v_2{:}\,\mathit{Int}}{\Gamma \vdash {}^{\langle \Gamma \rangle}\mathbf{let}\ x = v_1 + v_2\ \mathbf{in}\ e : \tau} \qquad \text{(T-Plus)}$$

$$\frac{\Gamma', x{:}\,\tau_1, y{:}\,\tau_2 \vdash e : \tau \qquad \Gamma = \Gamma' + z{:}\,\tau_1 \times^1 \tau_2}{\Gamma \vdash {}^{\langle \Gamma \rangle}\mathbf{let}\ (x, y) = z\ \mathbf{in}\ e : \tau} \qquad \text{(T-Ext)}$$

$$\frac{\Gamma' \vdash e_1 : \tau \qquad \Gamma' \vdash e_2 : \tau \qquad \Gamma = \Gamma' + v{:}\,\mathit{Int}}{\Gamma \vdash {}^{\langle \Gamma \rangle}\mathbf{if0}\ v\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : \tau} \qquad \text{(T-If)}$$

$$\Gamma + v{:}\,\tau \vdash {}^{\langle \Gamma + v:\tau \rangle} v : \tau \qquad \text{(T-Ret)}$$

Fig. 3. $\lambda^\kappa_{\mathrm{gc}}$ typing rules for expressions

Environments, heap values:

$$v_1{:}\,\tau_1 + \cdots + v_n{:}\,\tau_n \vdash \{x_1 = v_1, \ldots, x_n = v_n, \ldots\} : x_1{:}\,\tau_1, \ldots, x_n{:}\,\tau_n \qquad \text{(T-Env)}$$

$$v_1{:}\,\tau_1 + v_2{:}\,\tau_2 \vdash (v_1, v_2)^\kappa : \tau_1 \times^\kappa \tau_2 \qquad \text{(T-HPair)}$$

$$\frac{\Gamma, y{:}\,\tau_1, x{:}\,\tau_1 \to^{\kappa_1} \tau_2 \vdash e : \tau_2 \qquad \Gamma' \vdash V : \Gamma \qquad \kappa = \kappa_2 \cdot (\kappa_1 + 1)}{\kappa \cdot \Gamma' \vdash (V, {}^{\langle \Gamma \rangle}\mathbf{fun}\, x(y) = e)^\kappa : \tau_1 \to^{\kappa_2} \tau_2} \qquad \text{(T-HFun)}$$

Stacks, heaps, programs:

$$\Gamma \vdash [\,] : \tau \to^1 \tau \qquad \text{(T-Empty)}$$

$$\frac{\Gamma_1 \vdash S : \tau_2 \to^1 \tau_3 \qquad \Gamma_2 \vdash V : \Gamma_3 \qquad \Gamma_3, x{:}\,\tau_1 \vdash e : \tau_2}{\Gamma_1 + \Gamma_2 \vdash S[V, \Gamma_3, \lambda x.e] : \tau_1 \to^1 \tau_3} \qquad \text{(T-Push)}$$

$$\frac{\Gamma_1 \vdash h_1^{\kappa_1} : \tau_1 \quad \cdots \quad \Gamma_n \vdash h_n^{\kappa_n} : \tau_n \qquad \Gamma' + (\tilde{x}{:}\,\tilde{\tau}) \geq \lfloor \Gamma + \Gamma_1 + \cdots + \Gamma_n \rfloor}{\Gamma' \vdash \{x_1 = h_1^{\kappa_1}, \ldots, x_n = h_n^{\kappa_n}\} : \Gamma; \tilde{x}{:}\,\tilde{\tau}} \qquad \text{(T-Heap)}$$

$$\frac{\emptyset \vdash H : \Gamma_1 + \Gamma_2; \Gamma \qquad \Gamma_1 \vdash S : \tau' \to^1 \tau \qquad \Gamma_2 \vdash V : \Gamma_3 \qquad \Gamma_3 \vdash e : \tau'}{\vdash (H, S, V, e) : \tau} \qquad \text{(T-Prog)}$$

Fig. 4. $\lambda^\kappa_{\mathrm{gc}}$ typing rules for environments, stacks, heaps, and programs

here, we often abbreviate $\Gamma \vdash H : \Gamma ; \Gamma$ to $\Gamma \vdash H : \Gamma$. The rule T-Prog is

4. Garbage Collection

□ GC Algorithm

- Collecting type information on heap reference

$$TL_{env}[V,\Gamma] \;=\; \omega\cdot\textstyle\sum_{x\in dom(\Gamma),\Gamma(x)\neq Int} V(x):\Gamma(x)$$

$$\begin{array}{lcl} TL_{stack}[[\;]] & = & \phi \\ TL_{stack}[[S[V,\Gamma,\lambda x.e]] & = & TL_{stack}[S] + TL_{env}[V,\Gamma] \end{array}$$

$$\begin{array}{lcl} TL_{hval}[(V,^{\langle\Gamma\rangle}\mathtt{fun}\ x(y)=e)^{\kappa},\tau_1\to^{\kappa'}\tau_2] & = & \omega\cdot TL_{env}[V,\Gamma] \\ TL_{hval}[(v_1,v_2)^{\kappa},\tau_1\times^{\kappa'}\tau_2] & = & \omega\cdot\sum_{\tau_i\neq Int} v_i:\tau_i \end{array}$$

- GC algorithm : $(H_f,\Gamma_s,H_t,\Gamma_t)\Rightarrow(H'_f,\Gamma'_s,H'_t,\Gamma'_t)$

$$\begin{array}{l}(H_f\uplus\{x=h^k\},(\Gamma_s,x:\tau),H_t,\Gamma_t)\Rightarrow\\ \quad(H_f,\Gamma_s+TL_{hval}[h^{\kappa},\tau],H_t\uplus\{x=h^{\kappa}\},(\Gamma_t,x:\tau))\end{array} \qquad \text{(GC-MARK)}$$

$$\frac{\Gamma_t(x)\not\geq\tau}{\begin{array}{l}(H_f,(\Gamma_s,x:\tau),H_t,\Gamma_t)\Rightarrow\\ \quad(H_f,\Gamma_s+TL_{hval}[H_t(x),\tau],H_t,\Gamma_t+x:\tau)\end{array}} \qquad \text{(GC-Remark)}$$

$$\frac{\tau=\tau_1\times^0\tau_2 \text{ or } \tau=\tau_1\to^0\tau_2 \text{ or } \Gamma_t(x)\geq\tau}{(H_f,(\Gamma_s,x:\tau),H_t,\Gamma_t)\Rightarrow(H_f,\Gamma_s,H_t,\Gamma_t)} \qquad \text{(GC-Skip)}$$

Definition 5 [*collection*]
A heap H' is a collection of a program (H,S,V,e) iff

$$(H,TL_{stack}[S]+TL_{env}[V,TE(e),\phi,\phi)\Rightarrow^*(H'',\phi,H',\Gamma_t)$$

Example: A collection of a program P

$$\begin{array}{l}P=(H,[],\{p=x_1,f=x_3,g=x_4\},\\ \qquad{}^{\langle p:Int\times^0(Int\times^0 Int),f:Int\to^1 Int,g:Int\to^1 Int\rangle}f\ 1+g\ 2)\quad\text{where}\\ H=\{x_1=(2,x_2)^{\omega},x_2=(3,4)^1,\\ \qquad x_3=(\{p=x_1\},^{\langle p:Int\times^1(Int\times^0 Int)\rangle}\mathtt{fun}\ f'(y)=y+(\sharp 1\ p))\\ \qquad x_4=(\{p=x_1\},^{\langle p:Int\times^1(Int\times^1 Int)\rangle}\mathtt{fun}\ g'(y)=y+(\sharp 1(\sharp 2\ p)))\}\end{array}$$

$$\begin{array}{ll} & (H,(x_3:Int\to^{\omega}Int,x_4:Int\to^{\omega}Int),\phi,\phi)\\ \Rightarrow & (H\backslash\{x_3\},(x_1:Int\times^{\omega}(Int\times^0 Int),x_4:Int\to^{\omega}Int),\\ & \;\{x_3=H(x_3)\},x_3:Int\to^{\omega}Int)\\ \Rightarrow & (H\backslash\{x_1,x_3\},(x_2:Int\times^0 Int,x_4:Int\to^{\omega}Int),\\ & \;\{x_1=(2,x_2)^{\omega},x_3=H(x_3)\},\\ & \;(x_1:Int\times^{\omega}(Int\times^0 Int),x:Int\to^{\omega}Int))\\ \Rightarrow & (H\backslash\{x_1,x_3\},(x_4:Int\to^{\omega}Int),\{x_1=(2,x_2)^{\omega},x_3=H(x_3)\},\\ & \;(x_1:Int\times^{\omega}(Int\times^0 Int),x:Int\to^{\omega}Int))\\ \Rightarrow & (\{x_2=(3,4)^1\},x_1:Int\times^{\omega}(Int\times^{\omega}Int),H\backslash\{x_2\},\\ & \;(x_1:Int\times^{\omega}(Int\times^0 Int),x_3:Int\to^{\omega}Int,x_4:Int\to^{\omega}Int))\\ \Rightarrow & (\{x_2=(3,4)^1\},x_2:Int\times^{\omega}Int,H\backslash\{x_2\},\\ & \;(x_1:Int\times^{\omega}(Int\times^{\omega}Int),x_3:Int\to^{\omega}Int,x_4:Int\to^{\omega}Int))\\ \Rightarrow & (\phi,\phi,H,(x_1:Int\times^{\omega}(Int\times^{\omega}Int),x_2:Int\times^{\omega}Int,\\ & \qquad x_3:Int\to^{\omega}Int,x_4:Int\to^{\omega}Int))\end{array}$$

Theorem 3 [*Correctness of GC Algorithm*]
If $\vdash(H,S,V,e):\tau$, then there exists a collection H' of the program (H,S,V,e) and $\vdash(H',S,V,e):\tau$.

5. Conclusion

□ Conclusion

- GC scheme for a programming language with static memory management based on linear type system

□ Further Study

- Effectiveness of use analysis
- Reduction of run-time cost : dynamic check of uses
- Reduction of GC-time cost