

$e ::=$	expression
c	constant
$op(e_1, \dots, e_n)$	primitive operation
$\text{if } e_1 \text{ then } e_2 \text{ else } e_3$	conditional
$\text{let } x = e_1 \text{ in } e_2$	variable declaration
x	variable reference
$\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \ \text{in } e_2$	declaration of recursive function
$e \ e_1 \ \dots \ e_n$	function call
(e_1, \dots, e_n)	creation of a tuple
$\text{let } (x_1, \dots, x_n) = e_1 \ \text{in } e_2$	pattern matching against a tuple
$\text{Array.create } e_1 \ e_2$	creation of an array
$e_1.(e_2)$	array element dereference
$e_1.(e_2) \leftarrow e_3$	assignment to an array element

図 1: MinCaml の抽象構文 (型は省略)

Abstract syntax of MinCaml

$\tau ::=$	type
π	primitive type
$\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau$	functional type
$\tau_1 \times \dots \times \tau_n$	tuple type
$\tau \ \text{array}$	array type
α	type variable (for type inference)

図 2: MinCaml の型

MinCaml types

<p>c is a constant of the π type <i>c</i> は π 型の定数</p> $\frac{}{\Gamma \vdash c : \pi}$	<p>op is a primitive operation that takes values whose types are π_1, \dots, π_n and gives a value of π type <i>op</i> は π_1, \dots, π_n 型の値を受け取って π 型の値を返すプリミティブ演算</p> $\frac{\Gamma \vdash e_1 : \pi_1 \quad \dots \quad \Gamma \vdash e_n : \pi_n}{\Gamma \vdash op(e_1, \dots, e_n) : \pi}$	
$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau}$	$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2}$	$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$
$\frac{\Gamma, x : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau, y_1 : \tau_1, \dots, y_n : \tau_n \vdash e_1 : \tau \quad \Gamma, x : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2 : \tau'}$	$\frac{\Gamma \vdash e : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau \quad \Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash e \ e_1 \ \dots \ e_n : \tau}$	
$\frac{\Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash (e_1, \dots, e_n) : \tau_1 \times \dots \times \tau_n}$	$\frac{\Gamma \vdash e_1 : \tau_1 \times \dots \times \tau_n \quad \Gamma, x_1 : \tau_1, \dots, x_n : \tau_n \vdash e_2 : \tau}{\Gamma \vdash \text{let } (x_1, \dots, x_n) = e_1 \text{ in } e_2 : \tau}$	
$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{Array.create } e_1 \ e_2 : \tau \text{ array}}$		
$\frac{\Gamma \vdash e_1 : \tau \text{ array} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1.(e_2) : \tau}$	$\frac{\Gamma \vdash e_1 : \tau \text{ array} \quad \Gamma \vdash e_2 : \text{int} \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash e_1.(e_2) \leftarrow e_3 : \text{unit}}$	

図 3: MinCaml の型つけ規則

MinCaml's typing rule

<pre> e ::= c op(x₁, ..., x_n) if x = y then e₁ else e₂ if x ≤ y then e₁ else e₂ let x = e₁ in e₂ x let rec x y₁ ... y_n = e₁ in e₂ x y₁ ... y_n (x₁, ..., x_n) let (x₁, ..., x_n) = y in e x.(y) x.(y) ← z </pre>

図 4: MinCaml の K 正規形 (外部配列・外部関数適用は省略)

MinCaml's K normal form (external arrays and external function calls are not included)

$\mathcal{K} : \text{Syntax.t} \rightarrow \text{KNormal.t}$

$\mathcal{K}(c)$	$= c$
$\mathcal{K}(\text{not}(e))$	$= \mathcal{K}(\text{if } e \text{ then false else true})$
$\mathcal{K}(e_1 = e_2)$	$= \mathcal{K}(\text{if } e_1 = e_2 \text{ then true else false})$
$\mathcal{K}(e_1 \leq e_2)$	$= \mathcal{K}(\text{if } e_1 \leq e_2 \text{ then true else false})$
$\mathcal{K}(\text{op}(e_1, \dots, e_n))$	$= \text{let } x_1 = \mathcal{K}(e_1) \text{ in } \dots \text{ let } x_n = \mathcal{K}(e_n) \text{ in } \text{op}(x_1, \dots, x_n)$ when op is not a logical nor comparison operator
$\mathcal{K}(\text{if not } e_1 \text{ then } e_2 \text{ else } e_3)$	$= \mathcal{K}(\text{if } e_1 \text{ then } e_3 \text{ else } e_2)$
$\mathcal{K}(\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in}$ $\text{if } x = y \text{ then } \mathcal{K}(e_3) \text{ else } \mathcal{K}(e_4)$
$\mathcal{K}(\text{if } e_1 \leq e_2 \text{ then } e_3 \text{ else } e_4)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in}$ $\text{if } x \leq y \text{ then } \mathcal{K}(e_3) \text{ else } \mathcal{K}(e_4)$
$\mathcal{K}(\text{if } e_1 \text{ then } e_2 \text{ else } e_3)$	$= \mathcal{K}(\text{if } e_1 = \text{false} \text{ then } e_3 \text{ else } e_2)$ when e1 is not a logical nor a comparison operator
$\mathcal{K}(\text{let } x = e_1 \text{ in } e_2)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in } \mathcal{K}(e_2)$
$\mathcal{K}(x)$	$= x$
$\mathcal{K}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \text{let rec } x \ y_1 \ \dots \ y_n = \mathcal{K}(e_1) \text{ in } \mathcal{K}(e_2)$
$\mathcal{K}(e \ e_1 \ \dots \ e_n)$	$= \text{let } x = \mathcal{K}(e) \text{ in let } y_1 = \mathcal{K}(e_1) \text{ in } \dots \text{ let } y_n = \mathcal{K}(e_n) \text{ in}$ $x \ y_1 \ \dots \ y_n$
$\mathcal{K}(e_1, \dots, e_n)$	$= \text{let } x_1 = \mathcal{K}(e_1) \text{ in } \dots \text{ let } x_n = \mathcal{K}(e_n) \text{ in } (x_1, \dots, x_n)$
$\mathcal{K}(\text{let } (x_1, \dots, x_n) = e_1 \text{ in } e_2)$	$= \text{let } y = \mathcal{K}(e_1) \text{ in let } (x_1, \dots, x_n) = y \text{ in } \mathcal{K}(e_2)$
$\mathcal{K}(\text{Array.create } e_1 \ e_2)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in create_array } x \ y$
$\mathcal{K}(e_1.(e_2))$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in } x.(y)$
$\mathcal{K}(e_1.(e_2) \leftarrow e_3)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in let } z = \mathcal{K}(e_3) \text{ in}$ $x.(y) \leftarrow z$

図 5: K 正規化 (論理値の整数化と、insert_let による最適化は省略)。右辺に出現していて左辺に出現していない変数は、すべて新しい (fresh) とする。

K normal for (conversion of logical values to numbers and optimization using insert_let is abbreviated). A variable that occurs in RHS but not in LHS should be considered a new/fresh.

$\alpha : \text{Id.t M.t} \rightarrow \text{KNormal.t} \rightarrow \text{KNormal.t}$

$$\begin{aligned}
\alpha_\varepsilon(c) &= c \\
\alpha_\varepsilon(\text{op}(x_1, \dots, x_n)) &= \text{op}(\varepsilon(x_1), \dots, \varepsilon(x_n)) \\
\alpha_\varepsilon(\text{if } x = y \text{ then } e_1 \text{ else } e_2) &= \text{if } \varepsilon(x) = \varepsilon(y) \text{ then } \alpha_\varepsilon(e_1) \text{ else } \alpha_\varepsilon(e_2) \\
\alpha_\varepsilon(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2) &= \text{if } \varepsilon(x) \leq \varepsilon(y) \text{ then } \alpha_\varepsilon(e_1) \text{ else } \alpha_\varepsilon(e_2) \\
\alpha_\varepsilon(\text{let } x = e_1 \text{ in } e_2) &= \text{let } x' = \alpha_\varepsilon(e_1) \text{ in } \alpha_{\varepsilon, x \mapsto x'}(e_2) \\
\alpha_\varepsilon(x) &= \varepsilon(x) \\
\alpha_\varepsilon(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2) &= \text{let rec } x' \ y'_1 \ \dots \ y'_n = \alpha_{\varepsilon, x \mapsto x', y_1 \mapsto y'_1, \dots, y_n \mapsto y'_n}(e_1) \text{ in } \\
&\quad \alpha_{\varepsilon, x \mapsto x'}(e_2) \\
\alpha_\varepsilon(x \ y_1 \ \dots \ y_n) &= \varepsilon(x) \ \varepsilon(y_1) \ \dots \ \varepsilon(y_n) \\
\alpha_\varepsilon((x_1, \dots, x_n)) &= (\varepsilon(x_1), \dots, \varepsilon(x_n)) \\
\alpha_\varepsilon(\text{let } (x_1, \dots, x_n) = y \text{ in } e) &= \text{let } (x'_1, \dots, x'_n) = \varepsilon(y) \text{ in } \alpha_{\varepsilon, x_1 \mapsto x'_1, \dots, x_n \mapsto x'_n}(e) \\
\alpha_\varepsilon(x.(y)) &= \varepsilon(x).(\varepsilon(y)) \\
\alpha_\varepsilon(x.(y) \leftarrow z) &= \varepsilon(x).(\varepsilon(y)) \leftarrow \varepsilon(z)
\end{aligned}$$

図 6: α 変換。 ε は α 変換前の変数を受け取って、 α 変換後の変数を返す写像。右辺に出現していて左辺に出現していない変数 (x' など) は、すべて fresh とする。

α conversion: ε is a mapping that takes a variable name and gives its α -converted name. A name that occurs only in RHS should be considered new/fresh name.

$\beta : \text{Id.t M.t} \rightarrow \text{KNormal.t} \rightarrow \text{KNormal.t}$

$$\begin{aligned}
\beta_\varepsilon(c) &= c \\
\beta_\varepsilon(\text{op}(x_1, \dots, x_n)) &= \text{op}(\varepsilon(x_1), \dots, \varepsilon(x_n)) \\
\beta_\varepsilon(\text{if } x = y \text{ then } e_1 \text{ else } e_2) &= \text{if } \varepsilon(x) = \varepsilon(y) \text{ then } \beta_\varepsilon(e_1) \text{ else } \beta_\varepsilon(e_2) \\
\beta_\varepsilon(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2) &= \text{if } \varepsilon(x) \leq \varepsilon(y) \text{ then } \beta_\varepsilon(e_1) \text{ else } \beta_\varepsilon(e_2) \\
\beta_\varepsilon(\text{let } x = e_1 \text{ in } e_2) &= \beta_{\varepsilon, x \mapsto y}(e_2) \quad \beta_\varepsilon(e_1) \text{ is a variable "y"} \\
\beta_\varepsilon(\text{let } x = e_1 \text{ in } e_2) &= \text{let } x = \beta_\varepsilon(e_1) \text{ in } \beta_\varepsilon(e_2) \quad \beta_\varepsilon(e_1) \text{ is not a variable} \\
\beta_\varepsilon(x) &= \varepsilon(x) \\
\beta_\varepsilon(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2) &= \text{let rec } x \ y_1 \ \dots \ y_n = \beta_\varepsilon(e_1) \text{ in } \beta_\varepsilon(e_2) \\
\beta_\varepsilon(x \ y_1 \ \dots \ y_n) &= \varepsilon(x) \ \varepsilon(y_1) \ \dots \ \varepsilon(y_n) \\
\beta_\varepsilon((x_1, \dots, x_n)) &= (\varepsilon(x_1), \dots, \varepsilon(x_n)) \\
\beta_\varepsilon(\text{let } (x_1, \dots, x_n) = y \text{ in } e) &= \text{let } (x_1, \dots, x_n) = \varepsilon(y) \text{ in } \beta_\varepsilon(e) \\
\beta_\varepsilon(x.(y)) &= \varepsilon(x).(\varepsilon(y)) \\
\beta_\varepsilon(x.(y) \leftarrow z) &= \varepsilon(x).(\varepsilon(y)) \leftarrow \varepsilon(z)
\end{aligned}$$

図 7: β 簡約。 ε は β 簡約前の変数を受け取って、 β 簡約後の変数を返す写像。 $\varepsilon(x)$ が定義されていない場合は、 $\varepsilon(x) = x$ とみなす。

β reduction: ε is a mapping that takes a variable name and gives its β -converted name. We consider $\varepsilon(x) = x$ when ε is not defined for x .

$\mathcal{A} : \text{KNormal.t} \rightarrow \text{KNormal.t}$

$\mathcal{A}(c)$	$= c$
$\mathcal{A}(op(x_1, \dots, x_n))$	$= op(x_1, \dots, x_n)$
$\mathcal{A}(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x = y \text{ then } \mathcal{A}(e_1) \text{ else } \mathcal{A}(e_2)$
$\mathcal{A}(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x \leq y \text{ then } \mathcal{A}(e_1) \text{ else } \mathcal{A}(e_2)$
$\mathcal{A}(\text{let } x = e_1 \text{ in } e_2)$	$= \text{let } \dots \text{ in let } x = e'_1 \text{ in } \mathcal{A}(e_2)$
	$\mathcal{A}(e_1)$ has a form $\text{let } \dots \text{ in } e'$ ($\text{let } \dots$ is a sequence of one or more let 's) and e'_1 is not a let form.
$\mathcal{A}(x)$	$= x$
$\mathcal{A}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \text{let rec } x \ y_1 \ \dots \ y_n = \mathcal{A}(e_1) \text{ in } \mathcal{A}(e_2)$
$\mathcal{A}(x \ y_1 \ \dots \ y_n)$	$= x \ y_1 \ \dots \ y_n$
$\mathcal{A}((x_1, \dots, x_n))$	$= (x_1, \dots, x_n)$
$\mathcal{A}(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \text{let } (x_1, \dots, x_n) = y \text{ in } \mathcal{A}(e)$
$\mathcal{A}(x.(y))$	$= x.(y)$
$\mathcal{A}(x.(y) \leftarrow z)$	$= x.(y) \leftarrow z$

⊠ 8: Reduction of nested let 's

$\mathcal{I} : (\text{Id.t list} \times \text{KNormal.t}) \text{M.t} \rightarrow \text{KNormal.t} \rightarrow \text{KNormal.t}$

$\mathcal{I}_\varepsilon(c)$	$= c$
$\mathcal{I}_\varepsilon(\text{op}(x_1, \dots, x_n))$	$= \text{op}(x_1, \dots, x_n)$
$\mathcal{I}_\varepsilon(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x = y \text{ then } \mathcal{I}_\varepsilon(e_1) \text{ else } \mathcal{I}_\varepsilon(e_2)$
$\mathcal{I}_\varepsilon(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x \leq y \text{ then } \mathcal{I}_\varepsilon(e_1) \text{ else } \mathcal{I}_\varepsilon(e_2)$
$\mathcal{I}_\varepsilon(\text{let } x = e_1 \text{ in } e_2)$	$= \text{let } x = \mathcal{I}_\varepsilon(e_1) \text{ in } \mathcal{I}_\varepsilon(e_2)$
$\mathcal{I}_\varepsilon(x)$	$= x$
$\mathcal{I}_\varepsilon(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \varepsilon' = \varepsilon, x \mapsto ((y_1, \dots, y_n), e_1) \text{ として}$ $\text{let rec } x \ y_1 \ \dots \ y_n = \mathcal{I}_{\varepsilon'}(e_1) \text{ in } \mathcal{I}_{\varepsilon'}(e_2)$ $\text{size}(e_1) \leq \text{th}$
$\mathcal{I}_\varepsilon(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \text{let rec } x \ y_1 \ \dots \ y_n = \mathcal{I}_\varepsilon(e_1) \text{ in } \mathcal{I}_\varepsilon(e_2)$ $\text{size}(e_1) > \text{th}$
$\mathcal{I}_\varepsilon(x \ y_1 \ \dots \ y_n)$	$= \alpha_{y_1 \mapsto z_1, \dots, y_n \mapsto z_n}(e) \quad \varepsilon(x) = ((z_1, \dots, z_n), e)$
$\mathcal{I}_\varepsilon(x \ y_1 \ \dots \ y_n)$	$= x \ y_1 \ \dots \ y_n \quad \varepsilon(x) \text{ is undefined}$
$\mathcal{I}_\varepsilon((x_1, \dots, x_n))$	$= (x_1, \dots, x_n)$
$\mathcal{I}_\varepsilon(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \text{let } (x_1, \dots, x_n) = y \text{ in } \mathcal{I}_\varepsilon(e)$
$\mathcal{I}_\varepsilon(x.(y))$	$= x.(y)$
$\mathcal{I}_\varepsilon(x.(y) \leftarrow z)$	$= x.(y) \leftarrow z$
$\text{size}(c)$	$= 1$
$\text{size}(\text{op}(x_1, \dots, x_n))$	$= 1$
$\text{size}(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= 1 + \text{size}(e_1) + \text{size}(e_2)$
$\text{size}(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= 1 + \text{size}(e_1) + \text{size}(e_2)$
$\text{size}(\text{let } x = e_1 \text{ in } e_2)$	$= 1 + \text{size}(e_1) + \text{size}(e_2)$
$\text{size}(x)$	$= 1$
$\text{size}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= 1 + \text{size}(e_1) + \text{size}(e_2)$
$\text{size}(x \ y_1 \ \dots \ y_n)$	$= 1$
$\text{size}((x_1, \dots, x_n))$	$= 1$
$\text{size}(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= 1 + \text{size}(e)$
$\text{size}(x.(y))$	$= 1$
$\text{size}(x.(y) \leftarrow z)$	$= 1$

Inline expansion: ε is a mapping that takes the name of a smaller-sized function and gives its formal arguments and body. "th" is a user-specified threshold, that specifies the maximum function size that can be expanded.

$\mathcal{F} : \text{KNormal.t M.t} \rightarrow \text{KNormal.t} \rightarrow \text{KNormal.t}$

$\mathcal{F}_\varepsilon(c)$	$= c$	X の場合: when X それ以外の場合: otherwise
$\mathcal{F}_\varepsilon(\text{op}(x_1, \dots, x_n))$	$= c$	$\text{op}(\varepsilon(x_1), \dots, \varepsilon(x_n)) = c$ の場合
$\mathcal{F}_\varepsilon(\text{op}(x_1, \dots, x_n))$	$= \text{op}(x_1, \dots, x_n)$	それ以外の場合
$\mathcal{F}_\varepsilon(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \mathcal{F}_\varepsilon(e_1)$	$\varepsilon(x)$ と $\varepsilon(y)$ are the same constants
$\mathcal{F}_\varepsilon(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \mathcal{F}_\varepsilon(e_2)$	$\varepsilon(x)$ と $\varepsilon(y)$ are different constants
$\mathcal{F}_\varepsilon(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x = y \text{ then } \mathcal{F}_\varepsilon(e_1) \text{ else } \mathcal{F}_\varepsilon(e_2)$	それ以外の場合
$\mathcal{F}_\varepsilon(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \mathcal{F}_\varepsilon(e_1)$	$\varepsilon(x)$ と $\varepsilon(y)$: constants & $\varepsilon(x) \leq \varepsilon(y)$ の場合
$\mathcal{F}_\varepsilon(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \mathcal{F}_\varepsilon(e_2)$	$\varepsilon(x)$ と $\varepsilon(y)$: constants & $\varepsilon(x) > \varepsilon(y)$ の場合
$\mathcal{F}_\varepsilon(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x \leq y \text{ then } \mathcal{F}_\varepsilon(e_1) \text{ else } \mathcal{F}_\varepsilon(e_2)$	それ以外の場合
$\mathcal{F}_\varepsilon(\text{let } x = e_1 \text{ in } e_2)$	$= e'_1 = \mathcal{F}_\varepsilon(e_1)$ として $\text{let } x = e'_1 \text{ in } \mathcal{F}_{\varepsilon, x \mapsto e'_1}(e_2)$	
$\mathcal{F}_\varepsilon(x)$	$= x$	
$\mathcal{F}_\varepsilon(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \text{let rec } x \ y_1 \ \dots \ y_n = \mathcal{F}_\varepsilon(e_1) \text{ in } \mathcal{F}_\varepsilon(e_2)$	
$\mathcal{F}_\varepsilon(x \ y_1 \ \dots \ y_n)$	$= x \ y_1 \ \dots \ y_n$	
$\mathcal{F}_\varepsilon((x_1, \dots, x_n))$	$= (x_1, \dots, x_n)$	
$\mathcal{F}_\varepsilon(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \text{let } x_1 = y_1 \text{ in } \dots \text{ let } x_n = y_n \text{ in } \mathcal{F}_\varepsilon(e)$	$\varepsilon(y) = (y_1, \dots, y_n)$ の場合
$\mathcal{F}_\varepsilon(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \text{let } (x_1, \dots, x_n) = y \text{ in } \mathcal{F}_\varepsilon(e)$	
$\mathcal{F}_\varepsilon(x.(y))$	$= x.(y)$	
$\mathcal{F}_\varepsilon(x.(y) \leftarrow z)$	$= x.(y) \leftarrow z$	

Constant folding: ε is a mapping that takes a variable and gives a constant. ~~and gives a constant.~~ its associated expression

$\mathcal{E} : \text{KNormal.t} \rightarrow \text{KNormal.t}$

$\mathcal{E}(c)$	$= c$	"X の場合" (X no baai): "when X"
$\mathcal{E}(op(x_1, \dots, x_n))$	$= op(x_1, \dots, x_n)$	"それ以外の場合" (sore igai no baai): "otherwise"
$\mathcal{E}(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x = y \text{ then } \mathcal{E}(e_1) \text{ else } \mathcal{E}(e_2)$	
$\mathcal{E}(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x \leq y \text{ then } \mathcal{E}(e_1) \text{ else } \mathcal{E}(e_2)$	
$\mathcal{E}(\text{let } x = e_1 \text{ in } e_2)$	$= \mathcal{E}(e_2)$	$effect(\mathcal{E}(e_1)) = false$ かつ $x \notin FV(\mathcal{E}(e_2))$ の場合
$\mathcal{E}(\text{let } x = e_1 \text{ in } e_2)$	$= \text{let } x = \mathcal{E}(e_1) \text{ in } \mathcal{E}(e_2)$	それ以外の場合
$\mathcal{E}(x)$	$= x$	
$\mathcal{E}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \mathcal{E}(e_2)$	$x \notin FV(\mathcal{E}(e_2))$ の場合
$\mathcal{E}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \text{let rec } x \ y_1 \ \dots \ y_n = \mathcal{E}(e_1) \text{ in } \mathcal{E}(e_2)$	それ以外の場合
$\mathcal{E}(x \ y_1 \ \dots \ y_n)$	$= x \ y_1 \ \dots \ y_n$	
$\mathcal{E}((x_1, \dots, x_n))$	$= (x_1, \dots, x_n)$	
$\mathcal{E}(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \mathcal{E}(e)$	$\{x_1, \dots, x_n\} \cap FV(\mathcal{E}(e)) = \emptyset$ の場合
$\mathcal{E}(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \text{let } (x_1, \dots, x_n) = y \text{ in } \mathcal{E}(e)$	それ以外の場合
$\mathcal{E}(x.(y))$	$= x.(y)$	
$\mathcal{E}(x.(y) \leftarrow z)$	$= x.(y) \leftarrow z$	

$effect : \text{KNormal.t} \rightarrow \text{bool}$

$effect(c)$	$= false$
$effect(op(x_1, \dots, x_n))$	$= false$
$effect(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= effect(e_1) \vee effect(e_2)$
$effect(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= effect(e_1) \vee effect(e_2)$
$effect(\text{let } x = e_1 \text{ in } e_2)$	$= effect(e_1) \vee effect(e_2)$
$effect(x)$	$= false$
$effect(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= effect(e_2)$
$effect(x \ y_1 \ \dots \ y_n)$	$= true$
$effect((x_1, \dots, x_n))$	$= false$
$effect(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= effect(e)$
$effect(x.(y))$	$= false$
$effect(x.(y) \leftarrow z)$	$= true$

図 11: 不要定義削除 (1/2)

Elimination of redundant definition

$FV : \text{KNormal.t} \rightarrow \text{S.t}$	
$FV(c)$	$= \emptyset$
$FV(op(x_1, \dots, x_n))$	$= \{x_1, \dots, x_n\}$
$FV(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \{x, y\} \cup FV(e_1) \cup FV(e_2)$
$FV(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \{x, y\} \cup FV(e_1) \cup FV(e_2)$
$FV(\text{let } x = e_1 \text{ in } e_2)$	$= FV(e_1) \cup (FV(e_2) \setminus \{x\})$
$FV(x)$	$= \{x\}$
$FV(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= ((FV(e_1) \setminus \{y_1, \dots, y_n\}) \cup FV(e_2)) \setminus \{x\}$
$FV(x \ y_1 \ \dots \ y_n)$	$= \{x, y_1, \dots, y_n\}$
$FV((x_1, \dots, x_n))$	$= \{x_1, \dots, x_n\}$
$FV(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \{y\} \cup (FV(e) \setminus \{x_1, \dots, x_n\})$
$FV(x.(y))$	$= \{x, y\}$
$FV(x.(y) \leftarrow z)$	$= \{x, y, z\}$

图 12: 不要定义消除 (2/2)

Elimination of redundant definition

$P ::=$	The whole program
$(\{D_1, \dots, D_n\}, e)$	Definitions of the top-level functions and the main routine
$D ::=$	Definitions of the top-level functions
$L_x(y_1, \dots, y_m)(z_1, \dots, z_n) = e$	Function: label/formal arguments/free variables/body
$e ::=$	
c	
$op(x_1, \dots, x_n)$	
$\text{if } x = y \text{ then } e_1 \text{ else } e_2$	
$\text{if } x \leq y \text{ then } e_1 \text{ else } e_2$	
$\text{let } x = e_1 \text{ in } e_2$	
x	
$\text{make_closure } x = (L_x, (y_1, \dots, y_n)) \text{ in } e$	Closure creation
$\text{apply_closure}(x, y_1, \dots, y_n)$	Function call using a closure
$\text{apply_direct}(L_x, y_1, \dots, y_n)$	Function call without using a closure
(x_1, \dots, x_n)	(for known functions)
$\text{let } (x_1, \dots, x_n) = y \text{ in } e$	
$x.(y)$	
$x.(y) \leftarrow z$	

图 13: The closure language

$\mathcal{C} : \text{KNormal.t} \rightarrow \text{Closure.t}$

$\mathcal{C}(c)$	$= c$
$\mathcal{C}(op(x_1, \dots, x_n))$	$= op(x_1, \dots, x_n)$
$\mathcal{C}(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x = y \text{ then } \mathcal{C}(e_1) \text{ else } \mathcal{C}(e_2)$
$\mathcal{C}(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x \leq y \text{ then } \mathcal{C}(e_1) \text{ else } \mathcal{C}(e_2)$
$\mathcal{C}(\text{let } x = e_1 \text{ in } e_2)$	$= \text{let } x = \mathcal{C}(e_1) \text{ in } \mathcal{C}(e_2)$
$\mathcal{C}(x)$	$= x$
$\mathcal{C}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \mathcal{D}$ に $\mathbb{L}_x(y_1, \dots, y_n)(z_1, \dots, z_m) = e'_1$ を加え、 $make_closure \ x = (\mathbb{L}_x, (z_1, \dots, z_m))$ in e'_2 を返す ただし $e'_1 = \mathcal{C}(e_1)$, $e'_2 = \mathcal{C}(e_2)$, $FV(e'_1) \setminus \{x, y_1, \dots, y_n\} = \{z_1, \dots, z_m\}$
$\mathcal{C}(x \ y_1 \ \dots \ y_n)$	$= apply_closure(x, y_1, \dots, y_n)$
$\mathcal{C}((x_1, \dots, x_n))$	$= (x_1, \dots, x_n)$
$\mathcal{C}(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \text{let } (x_1, \dots, x_n) = y \text{ in } \mathcal{C}(e)$
$\mathcal{C}(x.(y))$	$= x.(y)$
$\mathcal{C}(x.(y) \leftarrow z)$	$= x.(y) \leftarrow z$

$FV : \text{Closure.t} \rightarrow \text{S.t}$

$FV(c)$	$= \emptyset$
$FV(op(x_1, \dots, x_n))$	$= \{x_1, \dots, x_n\}$
$FV(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \{x, y\} \cup FV(e_1) \cup FV(e_2)$
$FV(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \{x, y\} \cup FV(e_1) \cup FV(e_2)$
$FV(\text{let } x = e_1 \text{ in } e_2)$	$= FV(e_1) \cup (FV(e_2) \setminus \{x\})$
$FV(x)$	$= \{x\}$
$FV(make_closure \ x = (\mathbb{L}_x, (y_1, \dots, y_n)) \text{ in } e)$	$= \{y_1, \dots, y_n\} \cup (FV(e) \setminus \{x\})$
$FV(apply_closure(x, y_1, \dots, y_n))$	$= \{x, y_1, \dots, y_n\}$
$FV(apply_direct(\mathbb{L}_x, y_1, \dots, y_n))$	$= \{y_1, \dots, y_n\}$
$FV((x_1, \dots, x_n))$	$= \{x_1, \dots, x_n\}$
$FV(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \{y\} \cup (FV(e) \setminus \{x_1, \dots, x_n\})$
$FV(x.(y))$	$= \{x, y\}$
$FV(x.(y) \leftarrow z)$	$= \{x, y, z\}$

図 14: 賢くない Closure 変換 $\mathcal{C}(e)$ 。 \mathcal{D} はトップレベル関数定義の集合を記憶しておくためのグローバル

Simply-minded closure conversion.

The variable \mathcal{D} collects the set of top-level function definitions.

$\mathcal{C} : \text{S.t} \rightarrow \text{KNormal.t} \rightarrow \text{Closure.t}$

$$\begin{aligned}
\mathcal{C}_s(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \ \text{in } e_2) &= \mathcal{D} \text{ に } \mathbf{L}_x(y_1, \dots, y_n)() = e'_1 \text{ を加え、} \\
&\quad \text{make_closure } x = (\mathbf{L}_x, ()) \text{ in } e'_2 \text{ を返す} \\
&\quad \text{ただし } e'_1 = \mathcal{C}_{s'}(e_1), e'_2 = \mathcal{C}_{s'}(e_2), s' = s \cup \{x\}, \\
&\quad \text{FV}(e'_1) \setminus \{y_1, \dots, y_n\} = \emptyset \text{ の場合} \\
\mathcal{C}_s(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \ \text{in } e_2) &= \mathcal{D} \text{ に } \mathbf{L}_x(y_1, \dots, y_n)(z_1, \dots, z_m) = e'_1 \text{ を加え、} \\
&\quad \text{make_closure } x = (\mathbf{L}_x, (z_1, \dots, z_m)) \text{ in } e'_2 \text{ を返す} \\
&\quad \text{ただし } e'_1 = \mathcal{C}_s(e_1), e'_2 = \mathcal{C}_s(e_2), \\
&\quad \text{FV}(e'_1) \setminus \{y_1, \dots, y_n\} \neq \emptyset, \\
&\quad \text{FV}(e'_1) \setminus \{x, y_1, \dots, y_n\} = \{z_1, \dots, z_m\} \text{ の場合} \\
\mathcal{C}_s(x \ y_1 \ \dots \ y_n) &= \text{apply_closure}(x, y_1, \dots, y_n) \quad x \notin s \text{ の場合} \\
\mathcal{C}_s(x \ y_1 \ \dots \ y_n) &= \text{apply_direct}(\mathbf{L}_x, y_1, \dots, y_n) \quad x \in s \text{ の場合}
\end{aligned}$$

図 15: やや賢い Closure 変換 $\mathcal{C}_s(e)$ 。 s は自由変数がないとわかっている関数の名前の集合。

$\mathcal{C} : \text{S.t} \rightarrow \text{KNormal.t} \rightarrow \text{Closure.t}$

$$\begin{aligned}
\mathcal{C}_s(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \ \text{in } e_2) &= \mathcal{D} \text{ に } \mathbf{L}_x(y_1, \dots, y_n)() = e'_1 \text{ を加え、} \\
&\quad \text{make_closure } x = (\mathbf{L}_x, ()) \text{ in } e'_2 \text{ を返す} \\
&\quad \text{ただし } e'_1 = \mathcal{C}_{s'}(e_1), e'_2 = \mathcal{C}_{s'}(e_2), s' = s \cup \{x\}, \\
&\quad \text{FV}(e'_1) \setminus \{y_1, \dots, y_n\} = \emptyset \text{ かつ } x \in \text{FV}(e'_2) \text{ の場合} \\
\mathcal{C}_s(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \ \text{in } e_2) &= \mathcal{D} \text{ に } \mathbf{L}_x(y_1, \dots, y_n)() = e'_1 \text{ を加え、} e'_2 \text{ を返す} \\
&\quad \text{ただし } e'_1 = \mathcal{C}_{s'}(e_1), e'_2 = \mathcal{C}_{s'}(e_2), s' = s \cup \{x\}, \\
&\quad \text{FV}(e'_1) \setminus \{y_1, \dots, y_n\} = \emptyset \text{ かつ } x \notin \text{FV}(e'_2) \text{ の場合} \\
\mathcal{C}_s(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \ \text{in } e_2) &= \mathcal{D} \text{ に } \mathbf{L}_x(y_1, \dots, y_n)(z_1, \dots, z_m) = e'_1 \text{ を加え、} \\
&\quad \text{make_closure } x = (\mathbf{L}_x, (z_1, \dots, z_m)) \text{ in } e'_2 \text{ を返す} \\
&\quad \text{ただし } e'_1 = \mathcal{C}_s(e_1), e'_2 = \mathcal{C}_s(e_2), \\
&\quad \text{FV}(e'_1) \setminus \{y_1, \dots, y_n\} \neq \emptyset, \\
&\quad \text{FV}(e'_1) \setminus \{x, y_1, \dots, y_n\} = \{z_1, \dots, z_m\} \text{ の場合} \\
\mathcal{C}_s(x \ y_1 \ \dots \ y_n) &= \text{apply_closure}(x, y_1, \dots, y_n) \quad x \notin s \text{ の場合} \\
\mathcal{C}_s(x \ y_1 \ \dots \ y_n) &= \text{apply_direct}(\mathbf{L}_x, y_1, \dots, y_n) \quad x \in s \text{ の場合}
\end{aligned}$$

図 16: もっと賢い Closure 変換 $\mathcal{C}_s(e)$

$P ::=$		
$(\{D_1, \dots, D_n\}, E)$		
$D ::=$		
$L_x(y_1, \dots, y_n) = E$		
$E ::=$	命令の列	Instruction sequence
$x \leftarrow e; E$	代入	Assignment
e	返値	Return a value
$e ::=$	式	Expressions
c	即値	Immediate values
L_x	ラベル	Label
$op(x_1, \dots, x_n)$	算術演算	Arithmetic operation
$\text{if } x = y \text{ then } E_1 \text{ else } E_2$	比較&分岐	Comparison and branch
$\text{if } x \leq y \text{ then } E_1 \text{ else } E_2$	比較&分岐	Comparison and branch
x	mov 命令	"mov" operation
$apply_closure(x, y_1, \dots, y_n)$	クロージャを用いた関数呼び出し	Function call via a closure
$apply_direct(L_x, y_1, \dots, y_n)$	クロージャを用いない関数呼び出し	Function call
$x.(y)$	ロード	Load/store a value from/to
$x.(y) \leftarrow z$	ストア	a memory address indexed by y
save (x, y)	変数 x の値をスタック位置 y に退避する	
restore (y)	スタック位置 y から値を復元する	
		Save/restore the value to/from the stack.

図 17: 仮想マシンコードの構文

Virtual machine code language

$\mathcal{V} : \text{Closure.prog} \rightarrow \text{SparcAsm.prog}$	
$\mathcal{V}(\{\{D_1, \dots, D_n\}, e\})$	$= (\{\mathcal{V}(D_1), \dots, \mathcal{V}(D_n)\}, \mathcal{V}(e))$
$\mathcal{V} : \text{Closure.fundef} \rightarrow \text{SparcAsm.fundef}$	
$\mathcal{V}(\text{L}_x(y_1, \dots, y_n)(z_1, \dots, z_n) = e)$	$= \text{L}_x(y_1, \dots, y_n) = z_1 \leftarrow \text{R}_0.(4); \dots; z_n \leftarrow \text{R}_0.(4n); \mathcal{V}(e)$
$\mathcal{V} : \text{Closure.t} \rightarrow \text{SparcAsm.t}$	
$\mathcal{V}(c)$	$= c$
$\mathcal{V}(op(x_1, \dots, x_n))$	$= op(x_1, \dots, x_n)$
$\mathcal{V}(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x = y \text{ then } \mathcal{V}(e_1) \text{ else } \mathcal{V}(e_2)$
$\mathcal{V}(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x \leq y \text{ then } \mathcal{V}(e_1) \text{ else } \mathcal{V}(e_2)$
$\mathcal{V}(\text{let } x = e_1 \text{ in } e_2)$	$= x \leftarrow \mathcal{V}(e_1); \mathcal{V}(e_2)$
$\mathcal{V}(x)$	$= x$
$\mathcal{V}(\text{make_closure } x = (\text{L}_x, (y_1, \dots, y_n)) \text{ in } e)$	$= x \leftarrow \text{R}_{\text{hp}}; \text{R}_{\text{hp}} \leftarrow \text{R}_{\text{hp}} + 4(n+1); z \leftarrow \text{L}_x; x.(0) \leftarrow z;$ $x.(4) \leftarrow y_1; \dots; x.(4n) \leftarrow y_n; \mathcal{V}(e)$
$\mathcal{V}(\text{apply_closure}(x, y_1, \dots, y_n))$	$= \text{apply_closure}(x, y_1, \dots, y_n)$
$\mathcal{V}(\text{apply_direct}(\text{L}_x, y_1, \dots, y_n))$	$= \text{apply_direct}(\text{L}_x, y_1, \dots, y_n)$
$\mathcal{V}((x_1, \dots, x_n))$	$= y \leftarrow \text{R}_{\text{hp}}; \text{R}_{\text{hp}} \leftarrow \text{R}_{\text{hp}} + 4n;$ $y.(0) \leftarrow x_1; \dots; y.(4(n-1)) \leftarrow x_n; y$
$\mathcal{V}(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \{x_1, \dots, x_n\} \cap FV(e) = \{x_{i_1}, \dots, x_{i_m}\}$ として $x_{i_1} \leftarrow y.(4(i_1 - 1)); \dots; x_{i_m} \leftarrow y.(4(i_m - 1)); \mathcal{V}(e)$
$\mathcal{V}(x.(y))$	$= y' \leftarrow 4 \times y; x.(y')$
$\mathcal{V}(x.(y) \leftarrow z)$	$= y' \leftarrow 4 \times y; x.(y') \leftarrow z$

図 18: 仮想マシンコード生成 $\mathcal{V}(P)$, $\mathcal{V}(D)$ および $\mathcal{V}(e)$ 。右辺に出現して左辺に出現しない変数は fresh とする。 R_{hp} はヒープポインタ (専用レジスタ)。 $e_1; e_2$ はダミーの変数 x について $x \leftarrow e_1; e_2$ の略記。 $x \leftarrow E_1; E_2$ は、 $E_1 = (x_1 \leftarrow e_1; \dots; x_n \leftarrow e_n; e)$ として、 $x_1 \leftarrow e_1; \dots; x_n \leftarrow e_n; x \leftarrow e; E_2$ の略記。

English translation of the caption is found in the slide

$$\begin{aligned}
& FV : \mathbf{S.t} \rightarrow \mathbf{SparcAsm.t} \rightarrow \mathbf{S.t} \\
& FV_s(x \leftarrow e; E) = s' = FV_s(E) \setminus \{x\} \text{ として } FV_{s'}(e) \\
& FV_s(e) = FV_s(e) \\
\\
& FV : \mathbf{S.t} \rightarrow \mathbf{SparcAsm.exp} \rightarrow \mathbf{S.t} \\
& FV_s(c) = s \\
& FV_s(\mathbf{L}_x) = s \\
& FV_s(\mathit{op}(x_1, \dots, x_n)) = \{x_1, \dots, x_n\} \cup s \\
& FV_s(\mathit{if } x = y \text{ then } E_1 \text{ else } E_2) = \{x, y\} \cup FV_s(E_1) \cup FV_s(E_2) \\
& FV_s(\mathit{if } x \leq y \text{ then } E_1 \text{ else } E_2) = \{x, y\} \cup FV_s(E_1) \cup FV_s(E_2) \\
& FV_s(x) = \{x\} \cup s \\
& FV_s(\mathit{apply_closure}(x, y_1, \dots, y_n)) = \{x, y_1, \dots, y_n\} \cup s \\
& FV_s(\mathit{apply_direct}(\mathbf{L}_x, y_1, \dots, y_n)) = \{y_1, \dots, y_n\} \cup s \\
& FV_s(x.(y)) = \{x, y\} \cup s \\
& FV_s(x.(y) \leftarrow z) = \{x, y, z\} \cup s \\
& FV_s(\mathit{save}(x, y)) = \{x\} \cup s \\
& FV_s(\mathit{restore}(y)) = s
\end{aligned}$$

図 19: 命令の列 E および式 e において生きている変数の集合 $FV_s(E)$ および $FV_s(e)$ 。 s は E や e の後で使われる変数の集合。以後の $FV(E)$ は $FV_\emptyset(E)$ の略記。

$$\begin{aligned}
& \mathcal{R} : \text{SparcAsm.prog} \rightarrow \text{SparcAsm.prog} \\
& \mathcal{R}(\{\{D_1, \dots, D_n\}, E\}) = (\{\mathcal{R}(D_1), \dots, \mathcal{R}(D_n)\}, \mathcal{R}_\emptyset(E, x, ())) \quad \mathbf{x \text{ is a fresh dummy variable}} \\
\\
& \mathcal{R} : \text{SparcAsm.fundef} \rightarrow \text{SparcAsm.fundef} \\
& \mathcal{R}(\text{L}_x(y_1, \dots, y_n) = E) = \text{L}_x(\mathbf{R}_1, \dots, \mathbf{R}_n) = \mathcal{R}_{x \mapsto \mathbf{R}_0, y_1 \mapsto \mathbf{R}_1, \dots, y_n \mapsto \mathbf{R}_n}(E, \mathbf{R}_0, \mathbf{R}_0) \\
\\
& \mathcal{R} : \text{Id.t M.t} \rightarrow \text{SparcAsm.t} \times \text{Id.t} \times \text{SparcAsm.t} \rightarrow \text{SparcAsm.t} \times \text{Id.t M.t} \\
& \mathcal{R}_\varepsilon((x \leftarrow e; E), z_{\text{dest}}, E_{\text{cont}}) = E'_{\text{cont}} = (z_{\text{dest}} \leftarrow E; E_{\text{cont}}), \\
& \quad \mathcal{R}_\varepsilon(e, x, E'_{\text{cont}}) = (E', \varepsilon'), \\
& \quad r \notin \{\varepsilon'(y) \mid y \in FV(E'_{\text{cont}})\}, \quad \mathbf{choose \ an \ unused \ register} \\
& \quad \mathcal{R}_{\varepsilon', x \mapsto r}(E, z_{\text{dest}}, E_{\text{cont}}) = (E'', \varepsilon'') \text{ として} \\
& \quad ((r \leftarrow E'; E''), \varepsilon'') \quad \mathbf{when \ x \ is \ not \ a \ register} \\
\\
& \mathcal{R}_\varepsilon((r \leftarrow e; E), z_{\text{dest}}, E_{\text{cont}}) = E'_{\text{cont}} = (z_{\text{dest}} \leftarrow E; E_{\text{cont}}), \\
& \quad \mathcal{R}_\varepsilon(e, r, E'_{\text{cont}}) = (E', \varepsilon'), \\
& \quad \mathcal{R}_{\varepsilon'}(E, z_{\text{dest}}, E_{\text{cont}}) = (E'', \varepsilon'') \text{ として} \\
& \quad ((r \leftarrow E'; E''), \varepsilon'') \\
\\
& \mathcal{R}_\varepsilon(e, x, E_{\text{cont}}) = \mathcal{R}_\varepsilon(e, x, E_{\text{cont}}) \quad \text{(次図参照)}
\end{aligned}$$

図 20: 単純なレジスタ割り当て $\mathcal{R}(P)$, $\mathcal{R}(D)$ および $\mathcal{R}_\varepsilon(E, z_{\text{dest}}, E_{\text{cont}})$ 。 ε は変数からレジスタへの写像、 z_{dest} は E の結果をセットする変数、 E_{cont} は E の後に実行される命令の列。 $\mathcal{R}_\varepsilon(E, x, E_{\text{cont}})$ の返り値はレジスタ割り当てされた命令の列 E' と、 E の後のレジスタ割り当てを表す写像 ε' の組。 [ファイル `regAlloc.notarget-nospill.ml` 参照]

a simple register allocation $\mathcal{R}(P)$, $\mathcal{R}(D)$, and $\mathcal{R}(E, z_{\text{dest}}, E_{\text{cont}})$.

- ε is a mapping from variables to registers.
- z_{dest} is the variable where the result of E should be stored.
- E_{cont} is an instruction sequence that should be executed after the execution of E .
- The result of $\mathcal{R}_\varepsilon(E, x, E_{\text{cont}})$ is a tuple of a register-allocated instruction sequence (E') and variable-register mapping (ε').

$\mathcal{R} : \text{Id.t M.t} \rightarrow \text{SparcAsm.exp} \times \text{Id.t} \times \text{SparcAsm.t} \rightarrow \text{SparcAsm.t} \times \text{Id.t M.t}$	
$\mathcal{R}_\varepsilon(c, z_{\text{dest}}, E_{\text{cont}})$	$= (c, \varepsilon)$
$\mathcal{R}_\varepsilon(L_x, z_{\text{dest}}, E_{\text{cont}})$	$= (L_x, \varepsilon)$
$\mathcal{R}_\varepsilon(\text{op}(x_1, \dots, x_n), z_{\text{dest}}, E_{\text{cont}})$	$= (\text{op}(\varepsilon(x_1), \dots, \varepsilon(x_n)), \varepsilon)$
$\mathcal{R}_\varepsilon(\text{if } x = y \text{ then } E_1 \text{ else } E_2, z_{\text{dest}}, E_{\text{cont}})$	$= \mathcal{R}_\varepsilon(E_1, z_{\text{dest}}, E_{\text{cont}}) = (E'_1, \varepsilon_1),$ $\mathcal{R}_\varepsilon(E_2, z_{\text{dest}}, E_{\text{cont}}) = (E'_2, \varepsilon_2),$ $\varepsilon' = \{z \mapsto r \mid \varepsilon_1(z) = \varepsilon_2(z) = r\},$ $\{z_1, \dots, z_n\} =$ $(FV(E_{\text{cont}}) \setminus \{z_{\text{dest}}\} \setminus \text{dom}(\varepsilon')) \cap \text{dom}(\varepsilon) \text{ として}$ $((\text{save}(\varepsilon(z_1), z_1); \dots; \text{save}(\varepsilon(z_n), z_n);$ $\text{if } \varepsilon(x) \leq \varepsilon(y) \text{ then } E'_1 \text{ else } E'_2), \varepsilon')$
$\mathcal{R}_\varepsilon(\text{if } x \leq y \text{ then } E_1 \text{ else } E_2, z_{\text{dest}}, E_{\text{cont}})$	$= \text{同様}$
$\mathcal{R}_\varepsilon(x, z_{\text{dest}}, E_{\text{cont}})$	$= (\varepsilon(x), \varepsilon)$ z's: live variables except z_dest
$\mathcal{R}_\varepsilon(\text{apply_closure}(x, y_1, \dots, y_n), z_{\text{dest}}, E_{\text{cont}})$	$= \{z_1, \dots, z_n\} = (FV(E_{\text{cont}}) \setminus \{z_{\text{dest}}\}) \cap \text{dom}(\varepsilon) \text{ として}$ $((\text{save}(\varepsilon(z_1), z_1); \dots; \text{save}(\varepsilon(z_n), z_n);$ $\text{apply_closure}(\varepsilon(x), \varepsilon(y_1), \dots, \varepsilon(y_n))), \emptyset)$
$\mathcal{R}_\varepsilon(\text{apply_direct}(L_x, y_1, \dots, y_n), z_{\text{dest}}, E_{\text{cont}})$	$= \text{同様}$
$\mathcal{R}_\varepsilon(x.(y), z_{\text{dest}}, E_{\text{cont}})$	$= (\varepsilon(x).(\varepsilon(y)), \varepsilon)$
$\mathcal{R}_\varepsilon(x.(y) \leftarrow z, z_{\text{dest}}, E_{\text{cont}})$	$= (\varepsilon(x).(\varepsilon(y)) \leftarrow \varepsilon(z), \varepsilon)$
$\mathcal{R}_\varepsilon(\text{save}(x, y), z_{\text{dest}}, E_{\text{cont}})$	$= (\text{save}(\varepsilon(x), y), \varepsilon)$
$\mathcal{R}_\varepsilon(\text{restore}(y), z_{\text{dest}}, E_{\text{cont}})$	$= (\text{restore}(y), \varepsilon)$

図 21: 単純なレジスタ割り当て $\mathcal{R}_\varepsilon(e, z_{\text{dest}}, E_{\text{cont}})$ 。 $\mathcal{R}_\varepsilon(e)$ の右辺で変数 x のレジスタ $\varepsilon(x)$ が定義されていない場合は、 $\mathcal{R}_\varepsilon(e) = \mathcal{R}_\varepsilon(x \leftarrow \text{restore}(x); e)$ とする。ただしレジスタ r については $\varepsilon(r) = r$ とする。
 [ファイル `regAlloc.notarget-nospill.ml` 参照]

A simple register allocation $\mathcal{R}_\varepsilon(e, z_{\text{dest}}, E_{\text{cont}})$

- when $\varepsilon(x)$ is not defined for a variable x in the RHS of $\mathcal{R}_\varepsilon(e)$,
 $\mathcal{R}_\varepsilon(e) = \mathcal{R}_\varepsilon(x \leftarrow \text{restore}(x); e)$
- $\varepsilon(r) = r$ for all the register

E_{cont} is used to test liveness of variables

$$\begin{aligned}
& \mathcal{T} : \text{Id.t} \rightarrow \text{SparcAsm.t} \times \text{Id.t} \rightarrow \text{bool} \times \text{S.t} \\
& \mathcal{T}_x((y \leftarrow e; E), z_{\text{dest}}) = \mathcal{T}_x(e, y) = (c_1, s_1) \text{ として、もし } c_1 \text{ ならば } (true, s_1) \\
& \quad \text{そうでなければ } \mathcal{T}_x(E, z_{\text{dest}}) = (c_2, s_2) \text{ として } (c_2, s_1 \cup s_2) \\
& \mathcal{T}_x(e, z_{\text{dest}}) = \mathcal{T}_x(e, z_{\text{dest}}) \\
\\
& \mathcal{T} : \text{Id.t} \rightarrow \text{SparcAsm.exp} \times \text{Id.t} \rightarrow \text{bool} \times \text{S.t} \\
& \mathcal{T}_x(x, z_{\text{dest}}) = (false, \{z_{\text{dest}}\}) \\
& \mathcal{T}_x(\text{if } y = z \text{ then } E_1 \text{ else } E_2, z_{\text{dest}}) = \mathcal{T}_x(E_1, z_{\text{dest}}) = (c_1, s_1), \\
& \quad \mathcal{T}_x(E_2, z_{\text{dest}}) = (c_2, s_2) \text{ として} \\
& \quad (c_1 \wedge c_2, s_1 \cup s_2) \\
& \mathcal{T}_x(\text{if } y \leq z \text{ then } E_1 \text{ else } E_2, z_{\text{dest}}) = \text{同上} \\
& \mathcal{T}_x(\text{apply_closure}(y_0, y_1, \dots, y_n), z_{\text{dest}}) = (true, \{\mathbb{R}_i \mid x = y_i\}) \\
& \mathcal{T}_x(\text{apply_direct}(L_y, y_1, \dots, y_n), z_{\text{dest}}) = \text{同上} \\
& \mathcal{T}_x(e, z_{\text{dest}}) = (false, \emptyset) \quad \text{それ以外の場合}
\end{aligned}$$

図 22: 変数 x に割り当てるレジスタ r を選ぶときに使う targeting $\mathcal{T}_x(E, z_{\text{dest}})$ および $\mathcal{T}_x(e, z_{\text{dest}})$ 。 E や e で関数呼び出しがあったかどうかを表す論理値 c と、 x を割り当てると良いレジスタの集合 s の組を返す。前々図の「 x がレジスタでない場合」において、 $\mathcal{T}_x(E'_{\text{cont}}, z_{\text{dest}}) = (c, s)$ として、できれば $r \in s$ とする。 [ファイル `regAlloc.target-nospill.ml` 参照]

Register selection scheme for a variable x .

- c : if E or e issues function application
- s : recommended set of registers for use of variable x

The result of register targeting can be used in choosing a fresh register r for in Fig. 20

$$\begin{aligned}
& \mathcal{R} : \text{Id.t} \text{ M.t} \rightarrow \text{SparcAsm.t} \times \text{Id.t} \times \text{SparcAsm.t} \rightarrow \text{SparcAsm.t} \times \text{Id.t} \text{ M.t} \\
& \mathcal{R}_\varepsilon((x \leftarrow e; E), z_{\text{dest}}, E_{\text{cont}}) = E'_{\text{cont}} = (z_{\text{dest}} \leftarrow E; E_{\text{cont}}), \\
& \quad \mathcal{R}_\varepsilon(e, x, E'_{\text{cont}}) = (E', \varepsilon'), \\
& \quad y \in FV(E'_{\text{cont}}), \\
& \quad \mathcal{R}_{\varepsilon' \setminus \{y \mapsto \varepsilon'(y)\}, x \mapsto \varepsilon'(y)}(E, z_{\text{dest}}, E_{\text{cont}}) = (E'', \varepsilon'') \text{ として} \\
& \quad \begin{cases} ((\text{save}(\varepsilon(y), y); \varepsilon'(y) \leftarrow E'; E''), \varepsilon'') & y \in \text{dom}(\varepsilon) \text{ のとき} \\ ((\varepsilon'(y) \leftarrow E'; E''), \varepsilon'') & y \notin \text{dom}(\varepsilon) \text{ のとき} \end{cases} \\
& \quad x \text{ がレジスタでなく、} \\
& \quad r \notin \{\varepsilon'(y) \mid y \in FV(E'_{\text{cont}})\} \text{ なる } r \text{ がない場合}
\end{aligned}$$

図 23: spilling をするレジスタ割り当て $\mathcal{R}_\varepsilon(E, z_{\text{dest}}, E_{\text{cont}})$ [ファイル `regAlloc.target-latespill.ml` 参照]

```

S : SparcAsm.prog → string
S(({D1, ..., Dn}, E)) = .section ".text"
                                S(D1)
                                ...
                                S(Dn)
                                .global min_caml_start
                                min_caml_start:
                                save %sp, -112, %sp
                                S(E, %g0)
                                ret
                                restore

S : SparcAsm.fundef → string
S(Lx(y1, ..., yn) = E) = x:
                                S(E, R0)
                                retl
                                nop

S : SparcAsm.t × Id.t → string
S((x ← e; E), zdest) = S(e, x)
                                S(E, zdest)
S(e, zdest) = S(e, zdest)

```

図 24: 単純なアセンブリ生成 $\mathcal{S}(P)$, $\mathcal{S}(D)$ および $\mathcal{S}(E, z_{\text{dest}})$

$\mathcal{S} : \text{SparcAsm.exp} \times \text{Id.t} \rightarrow \text{string}$	
$\mathcal{S}(c, z_{\text{dest}})$	= set c, z_{dest}
$\mathcal{S}(L_x, z_{\text{dest}})$	= set L_x, z_{dest}
$\mathcal{S}(op(x_1, \dots, x_n), z_{\text{dest}})$	= $op\ x_1, \dots, x_n, z_{\text{dest}}$
$\mathcal{S}(\text{if } x = y \text{ then } E_1 \text{ else } E_2, z_{\text{dest}})$	= cmp x, y bne b_1 nop $\mathcal{S}(E_1, z_{\text{dest}})$ b b_2 nop $b_1 :$ $\mathcal{S}(E_2, z_{\text{dest}})$ $b_2 :$
$\mathcal{S}(\text{if } x \leq y \text{ then } E_1 \text{ else } E_2, z_{\text{dest}})$	= 同様
$\mathcal{S}(x, z_{\text{dest}})$	= mov x, z_{dest}
$\mathcal{S}(\text{apply_closure}(x, y_1, \dots, y_n), z_{\text{dest}})$	= $shuffle((x, y_1, \dots, y_n), (R_0, R_1, \dots, R_n))$ st $R_{ra}, [R_{st} + 4\#\varepsilon]$ Save return address in the stack ld $[R_0], R_{n+1}$ Load the function address in R_{n+1} call R_{n+1} Function call add $R_{st}, 4(\#\varepsilon + 1), R_{st}$! delay slot Allocate in stack area Execution resumes from here after funcall sub $R_{st}, 4(\#\varepsilon + 1), R_{st}$ Pop the stack ld $[R_{st} + 4\#\varepsilon], R_{ra}$ mov R_0, z_{dest}
$\mathcal{S}(\text{apply_direct}(L_x, y_1, \dots, y_n), z_{\text{dest}})$	= $shuffle((y_1, \dots, y_n), (R_1, \dots, R_n))$ st $R_{ra}, [R_{st} + 4\#\varepsilon]$ call x add $R_{st}, 4(\#\varepsilon + 1), R_{st}$! delay slot sub $R_{st}, 4(\#\varepsilon + 1), R_{st}$ ld $[R_{st} + 4\#\varepsilon], R_{ra}$ mov R_0, z_{dest}
$\mathcal{S}(x.(y), z_{\text{dest}})$	= ld $[x + y], z_{\text{dest}}$
$\mathcal{S}(x.(y) \leftarrow z, z_{\text{dest}})$	= st $z, [x + y]$
$\mathcal{S}(\text{save}(x, y), z_{\text{dest}})$	= もし $y \notin \text{dom}(\varepsilon)$ なら ε に $y \mapsto 4\#\varepsilon$ を加えて st $x, [R_{st} + \varepsilon(y)]$
$\mathcal{S}(\text{restore}(y), z_{\text{dest}})$	= ld $[R_{st} + \varepsilon(y)], z_{\text{dest}}$

図 25: 単純なアセンブリ生成 $\mathcal{S}(e, z_{\text{dest}})$ 。 ε はスタック位置を記憶するグローバル変数。 $\#\varepsilon$ は ε の要素の個数。 $shuffle((x_1, \dots, x_n), (r_1, \dots, r_n))$ は x_1, \dots, x_n を r_1, \dots, r_n に適切な順序で移動する命令。

$$\begin{aligned}
& \mathcal{S} : \mathbf{S.t} \rightarrow \text{SparcAsm.t} \times \text{Id.t} \rightarrow \mathbf{S.t} \times \text{string} \\
& \mathcal{S}_s((x \leftarrow e; E), z_{\text{dest}}) = \mathcal{S}_s(e, x) = (s', S), \\
& \qquad \qquad \qquad \mathcal{S}_{s'}(E, z_{\text{dest}}) = (s'', S') \text{ として} \\
& \qquad \qquad \qquad (s'', SS') \\
& \mathcal{S}_s(e, z_{\text{dest}}) = \mathcal{S}_s(e, z_{\text{dest}}) \\
\\
& \mathcal{S} : \mathbf{S.t} \rightarrow \text{SparcAsm.exp} \times \text{Id.t} \rightarrow \mathbf{S.t} \times \text{string} \\
& \mathcal{S}_s(\text{if } x = y \text{ then } E_1 \text{ else } E_2, z_{\text{dest}}) = \mathcal{S}_s(E_1, z_{\text{dest}}) = (s_1, S_1), \\
& \qquad \qquad \qquad \mathcal{S}_s(E_2, z_{\text{dest}}) = (s_2, S_2) \text{ として} \\
& \qquad \qquad \qquad (s_1 \cap s_2, \\
& \qquad \qquad \qquad \text{cmp } x, y \\
& \qquad \qquad \qquad \text{bne } b_1 \\
& \qquad \qquad \qquad \text{nop} \\
& \qquad \qquad \qquad S_1 \\
& \qquad \qquad \qquad \text{b } b_2 \\
& \qquad \qquad \qquad \text{nop} \\
& \qquad \qquad \qquad b_1 : \\
& \qquad \qquad \qquad S_2 \\
& \qquad \qquad \qquad b_2 :) \\
& \mathcal{S}_s(\text{if } x \leq y \text{ then } E_1 \text{ else } E_2, z_{\text{dest}}) = \text{同様} \\
& \mathcal{S}_s(\text{save}(x, y), z_{\text{dest}}) = (s, \text{nop}) \qquad \qquad \qquad y \in s \text{ の場合} \\
& \mathcal{S}_s(\text{save}(x, y), z_{\text{dest}}) = \text{もし } y \notin \text{dom}(\varepsilon) \text{ なら } \varepsilon \text{ に } y \mapsto 4\#\varepsilon \text{ を加えて} \\
& \qquad \qquad \qquad (s \cup \{y\}, \text{st } x, [\text{R}_{\text{st}} + \varepsilon(y)]) \qquad y \notin s \text{ の場合} \\
& \mathcal{S}_s(e, z_{\text{dest}}) = (s, \text{以前と同様}) \qquad \qquad \qquad \text{上述以外の場合}
\end{aligned}$$

図 26: 無駄な save を省略するアセンブリ生成 $\mathcal{S}_s(E, z_{\text{dest}})$ および $\mathcal{S}_s(e, z_{\text{dest}})$ 。 s はすでに save された変数の名前の集合。以前の $\mathcal{S}(E, z_{\text{dest}})$ は $\mathcal{S}_\emptyset(E, z_{\text{dest}}) = (s, S)$ として S の略記とする。

$\mathcal{S} : \text{SparcAsm.fundef} \rightarrow \text{string}$
 $\mathcal{S}(\text{L}_x(y_1, \dots, y_n) = E) = \mathcal{S}_\emptyset(E, \text{tail}) = (s, S)$ として
 $x:$
 S

$\mathcal{S} : \text{S.t} \rightarrow \text{SparcAsm.exp} \times \text{Id.t} \rightarrow \text{S.t} \times \text{string}$
 $\mathcal{S}_s(\text{if } x = y \text{ then } E_1 \text{ else } E_2, \text{tail}) = \mathcal{S}_s(E_1, \text{tail}) = (s_1, S_1),$
 $\mathcal{S}_s(E_2, \text{tail}) = (s_2, S_2)$ として
 $(\emptyset,$
 $\text{cmp } x, y$
 $\text{bne } b$
 nop
 S_1
 $b:$
 $S_2)$

$\mathcal{S}_s(\text{if } x \leq y \text{ then } E_1 \text{ else } E_2, \text{tail}) = \text{同様}$
 $\mathcal{S}_s(\text{apply_closure}(x, y_1, \dots, y_n), \text{tail}) = (\emptyset,$
 $\text{shuffle}((x, y_1, \dots, y_n), (\mathbf{R}_0, \mathbf{R}_1, \dots, \mathbf{R}_n))$
 $\text{ld } [\mathbf{R}_0], \mathbf{R}_{n+1}$
 $\text{jmp } \mathbf{R}_{n+1}$
 $\text{nop})$

$\mathcal{S}_s(\text{apply_direct}(\text{L}_x, y_1, \dots, y_n), \text{tail}) = (\emptyset,$
 $\text{shuffle}((y_1, \dots, y_n), (\mathbf{R}_1, \dots, \mathbf{R}_n))$
 $\text{b } x$
 $\text{nop})$

$\mathcal{S}_s(e, \text{tail}) = \mathcal{S}_s(e, \mathbf{R}_0) = (s', S)$ として
 $(\emptyset,$
 S
 retl
 $\text{nop})$ 上述以外の場合

図 27: 末尾呼び出し最適化をするアセンブリ生成 $\mathcal{S}_s(D)$ および $\mathcal{S}_s(e, z_{\text{dest}})$ 。 $z_{\text{dest}} = \text{tail}$ の場合が末尾。