e ::=	式	*Expression*
c	定数	*Constant*
$op(e_1, \ldots, e_n)$	プリミティブ演算	*Arithmetic*
`if` e_1 `then` e_2 `else` e_3	条件分岐	*Conditional*
`let` $x = e_1$ `in` e_2	変数定義	*Variable declaration*
x	変数の読み出し	*Variable dereference*
`let rec` $x\ y_1\ \ldots\ y_n = e_1$ `in` e_2	再帰関数定義	*Recursive function*
$e\ e_1\ \ldots\ e_n$	関数呼び出し	*Function call*
$(e_1, \ldots, e_n)$	組の作成	*Tuple*
`let` $(x_1, \ldots, x_n) = e_1$ `in` e_2	組の読み出し	*Decomposition of a tuple*
`Array.create` $e_1\ e_2$	配列の作成	*Array creation*
$e_1.(e_2)$	配列の読み出し	*Indexing an array*
$e_1.(e_2) \leftarrow e_3$	配列への書き込み	*Assignment to an array*

図 1: MinCaml の抽象構文（型は省略）

Abstract syntax of MinCaml (Type is omitted)

τ ::=	型	**Types in MinCaml is**
π	プリミティブ型	**l Primitive type**
$\tau_1 \to \ldots \to \tau_n \to \tau$	関数型	**l Functional type**
$\tau_1 \times \ldots \times \tau_n$	組型	**l Tuple type**
τ `array`	配列型	**l Array type**
α	型変数（型推論用）	**l Type variable (used by the type inference algorithm)**

図 2: MinCaml の型

MinCaml types

op is a primitive operator that takes values of π1, …, πn and gives a value of π

c is a constant member of π

$$\frac{c \text{ は } \pi \text{ 型の定数}}{\Gamma \vdash c : \pi}$$

$$\frac{\Gamma \vdash e_1 : \pi_1 \quad \dots \quad \Gamma \vdash e_n : \pi_n \quad op \text{ は } \pi_1, \dots, \pi_n \text{ 型の値を受け取って } \pi \text{ 型の値を返すプリミティブ演算}}{\Gamma \vdash op(e_1, \dots, e_n) : \pi}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let } x = e_1 \texttt{ in } e_2 : \tau_2} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

$$\frac{\begin{array}{c}\Gamma, x : \tau_1 \to \dots \to \tau_n \to \tau, y_1 : \tau_1, \dots, y_n : \tau_n \vdash e_1 : \tau \\ \Gamma, x : \tau_1 \to \dots \to \tau_n \to \tau \vdash e_2 : \tau'\end{array}}{\Gamma \vdash \texttt{let rec } x\ y_1\ \dots\ y_n = e_1 \texttt{ in } e_2 : \tau'} \qquad \frac{\begin{array}{c}\Gamma \vdash e : \tau_1 \to \dots \to \tau_n \to \tau \\ \Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n\end{array}}{\Gamma \vdash e\ e_1\ \dots\ e_n : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash (e_1, \dots, e_n) : \tau_1 \times \dots \times \tau_n} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \times \dots \times \tau_n \quad \Gamma, x_1 : \tau_1, \dots, x_n : \tau_n \vdash e_2 : \tau}{\Gamma \vdash \texttt{let } (x_1, \dots, x_n) = e_1 \texttt{ in } e_2 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{Array.create } e_1\ e_2 : \tau\ \texttt{array}}$$

$$\frac{\Gamma \vdash e_1 : \tau\ \texttt{array} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1.(e_2) : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau\ \texttt{array} \quad \Gamma \vdash e_2 : \texttt{int} \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash e_1.(e_2) \leftarrow e_3 : \texttt{unit}}$$

図 3: MinCaml の型つけ規則

MinCaml's typing rule

$$\begin{array}{ll}
e ::= & \\
& c \\
& op(x_1, \dots, x_n) \\
& \texttt{if } x = y \texttt{ then } e_1 \texttt{ else } e_2 \\
& \texttt{if } x \leq y \texttt{ then } e_1 \texttt{ else } e_2 \\
& \texttt{let } x = e_1 \texttt{ in } e_2 \\
& x \\
& \texttt{let rec } x\ y_1\ \dots\ y_n = e_1 \texttt{ in } e_2 \\
& x\ y_1\ \dots\ y_n \\
& (x_1, \dots, x_n) \\
& \texttt{let } (x_1, \dots, x_n) = y \texttt{ in } e \\
& x.(y) \\
& x.(y) \leftarrow z
\end{array}$$

図 4: MinCaml の K 正規形（外部配列・外部関数適用は省略）

K-normal-form in MinCaml
External arrays and External function application are omitted

$\mathcal{K}: \texttt{Syntax.t} \to \texttt{KNormal.t}$

$$
\begin{array}{lcl}
\mathcal{K}(c) & = & c \\
\mathcal{K}(\texttt{not}(e)) & = & \mathcal{K}(\texttt{if } e \texttt{ then false else true}) \\
\mathcal{K}(e_1 = e_2) & = & \mathcal{K}(\texttt{if } e_1 = e_2 \texttt{ then true else false}) \\
\mathcal{K}(e_1 \leq e_2) & = & \mathcal{K}(\texttt{if } e_1 \leq e_2 \texttt{ then true else false}) \\
\mathcal{K}(op(e_1, \ldots, e_n)) & = & \texttt{let } x_1 = \mathcal{K}(e_1) \texttt{ in } \ldots \texttt{ let } x_n = \mathcal{K}(e_n) \texttt{ in } op(x_1, \ldots, x_n) \\
 & & \hfill op \text{ が論理演算・比較以外の場合} \\
\mathcal{K}(\texttt{if not } e_1 \texttt{ then } e_2 \texttt{ else } e_3) & = & \mathcal{K}(\texttt{if } e_1 \texttt{ then } e_3 \texttt{ else } e_2) \\
\mathcal{K}(\texttt{if } e_1 = e_2 \texttt{ then } e_3 \texttt{ else } e_4) & = & \texttt{let } x = \mathcal{K}(e_1) \texttt{ in let } y = \mathcal{K}(e_2) \texttt{ in} \\
 & & \texttt{if } x = y \texttt{ then } \mathcal{K}(e_3) \texttt{ else } \mathcal{K}(e_4) \\
\mathcal{K}(\texttt{if } e_1 \leq e_2 \texttt{ then } e_3 \texttt{ else } e_4) & = & \texttt{let } x = \mathcal{K}(e_1) \texttt{ in let } y = \mathcal{K}(e_2) \texttt{ in} \\
 & & \texttt{if } x \leq y \texttt{ then } \mathcal{K}(e_3) \texttt{ else } \mathcal{K}(e_4) \\
\mathcal{K}(\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3) & = & \mathcal{K}(\texttt{if } e_1 = \texttt{false then } e_3 \texttt{ else } e_2) \\
 & & \hfill e_1 \text{ が論理演算・比較以外の場合} \\
\mathcal{K}(\texttt{let } x = e_1 \texttt{ in } e_2) & = & \texttt{let } x = \mathcal{K}(e_1) \texttt{ in } \mathcal{K}(e_2) \\
\mathcal{K}(x) & = & x \\
\mathcal{K}(\texttt{let rec } x\ y_1\ \ldots\ y_n = e_1 \texttt{ in } e_2) & = & \texttt{let rec } x\ y_1\ \ldots\ y_n = \mathcal{K}(e_1) \texttt{ in } \mathcal{K}(e_2) \\
\mathcal{K}(e\ e_1\ \ldots\ e_n) & = & \texttt{let } x = \mathcal{K}(e) \texttt{ in let } y_1 = \mathcal{K}(e_1) \texttt{ in } \ldots \texttt{ let } y_n = \mathcal{K}(e_n) \texttt{ in} \\
 & & x\ y_1\ \ldots\ y_n \\
\mathcal{K}(e_1, \ldots, e_n) & = & \texttt{let } x_1 = \mathcal{K}(e_1) \texttt{ in } \ldots \texttt{ let } x_n = \mathcal{K}(e_n) \texttt{ in } (x_1, \ldots, x_n) \\
\mathcal{K}(\texttt{let } (x_1, \ldots, x_n) = e_1 \texttt{ in } e_2) & = & \texttt{let } y = \mathcal{K}(e_1) \texttt{ in let } (x_1, \ldots, x_n) = y \texttt{ in } \mathcal{K}(e_2) \\
\mathcal{K}(\texttt{Array.create } e_1\ e_2) & = & \texttt{let } x = \mathcal{K}(e_1) \texttt{ in let } y = \mathcal{K}(e_2) \texttt{ in create\_array } x\ y \\
\mathcal{K}(e_1.(e_2)) & = & \texttt{let } x = \mathcal{K}(e_1) \texttt{ in let } y = \mathcal{K}(e_2) \texttt{ in } x.(y) \\
\mathcal{K}(e_1.(e_2) \leftarrow e_3) & = & \texttt{let } x = \mathcal{K}(e_1) \texttt{ in let } y = \mathcal{K}(e_2) \texttt{ in let } z = \mathcal{K}(e_3) \texttt{ in} \\
 & & x.(y) \leftarrow z
\end{array}
$$

op is NOT a logical operator nor comparator

e1 is not a logical expression nor comparation

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。

Conversion to K-normal-form (conversion of logical values to integers and optimization by insert=let is omitted). Variables that occur in right-hand-side but not in left- are regarded as fresh.

α : `Id.t M.t` $\rightarrow$ `KNormal.t` $\rightarrow$ `KNormal.t`

$$
\begin{array}{lcl}
\alpha_\varepsilon(c) & = & c \\
\alpha_\varepsilon(op(x_1,\ldots,x_n)) & = & op(\varepsilon(x_1),\ldots,\varepsilon(x_n)) \\
\alpha_\varepsilon(\texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ \varepsilon(x) = \varepsilon(y)\ \texttt{then}\ \alpha_\varepsilon(e_1)\ \texttt{else}\ \alpha_\varepsilon(e_2) \\
\alpha_\varepsilon(\texttt{if}\ x \le y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ \varepsilon(x) \le \varepsilon(y)\ \texttt{then}\ \alpha_\varepsilon(e_1)\ \texttt{else}\ \alpha_\varepsilon(e_2) \\
\alpha_\varepsilon(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & = & \texttt{let}\ x' = \alpha_\varepsilon(e_1)\ \texttt{in}\ \alpha_{\varepsilon,x\mapsto x'}(e_2) \\
\alpha_\varepsilon(x) & = & \varepsilon(x) \\
\alpha_\varepsilon(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \texttt{let rec}\ x'\ y'_1\ \ldots\ y'_n = \alpha_{\varepsilon,x\mapsto x',y_1\mapsto y'_1,\ldots,y_n\mapsto y'_n}(e_1)\ \texttt{in} \\
 & & \alpha_{\varepsilon,x\mapsto x'}(e_2) \\
\alpha_\varepsilon(x\ y_1\ \ldots\ y_n) & = & \varepsilon(x)\ \varepsilon(y_1)\ \ldots\ \varepsilon(y_n) \\
\alpha_\varepsilon((x_1,\ldots,x_n)) & = & (\varepsilon(x_1),\ldots,\varepsilon(x_n)) \\
\alpha_\varepsilon(\texttt{let}\ (x_1,\ldots,x_n) = y\ \texttt{in}\ e) & = & \texttt{let}\ (x'_1,\ldots,x'_n) = \varepsilon(y)\ \texttt{in}\ \alpha_{\varepsilon,x_1\mapsto x'_1,\ldots,x_n\mapsto x'_n}(e) \\
\alpha_\varepsilon(x.(y)) & = & \varepsilon(x).(\varepsilon(y)) \\
\alpha_\varepsilon(x.(y) \leftarrow z) & = & \varepsilon(x).(\varepsilon(y)) \leftarrow \varepsilon(z)
\end{array}
$$

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 maps a variable that occur in the original expression to the corresponding variable in the resulting expression. Variables that occur in the right-hand-side and not in the left- (such as x') are considered fresh.

β : `Id.t M.t` $\rightarrow$ `KNormal.t` $\rightarrow$ `KNormal.t`

$$
\begin{array}{lcll}
\beta_\varepsilon(c) & = & c & \\
\beta_\varepsilon(op(x_1,\ldots,x_n)) & = & op(\varepsilon(x_1),\ldots,\varepsilon(x_n)) & \\
\beta_\varepsilon(\texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ \varepsilon(x) = \varepsilon(y)\ \texttt{then}\ \beta_\varepsilon(e_1)\ \texttt{else}\ \beta_\varepsilon(e_2) & \\
\beta_\varepsilon(\texttt{if}\ x \le y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ \varepsilon(x) \le \varepsilon(y)\ \texttt{then}\ \beta_\varepsilon(e_1)\ \texttt{else}\ \beta_\varepsilon(e_2) & \text{Case 1: } \beta\varepsilon(\text{e1}) = \text{y} \\
\beta_\varepsilon(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & = & \beta_{\varepsilon,x\mapsto y}(e_2) & \beta_\varepsilon(e_1)\ \text{が変数}\ y\ \text{の場合} \\
\beta_\varepsilon(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & = & \texttt{let}\ x = \beta_\varepsilon(e_1)\ \texttt{in}\ \beta_\varepsilon(e_2) & \beta_\varepsilon(e_1)\ \text{が変数でない場合} \\
\beta_\varepsilon(x) & = & \varepsilon(x) & \text{Case 2: } \beta\varepsilon(\text{e1}) \text{ is not a variable} \\
\beta_\varepsilon(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \texttt{let rec}\ x\ y_1\ \ldots\ y_n = \beta_\varepsilon(e_1)\ \texttt{in}\ \beta_\varepsilon(e_2) & \\
\beta_\varepsilon(x\ y_1\ \ldots\ y_n) & = & \varepsilon(x)\ \varepsilon(y_1)\ \ldots\ \varepsilon(y_n) & \\
\beta_\varepsilon((x_1,\ldots,x_n)) & = & (\varepsilon(x_1),\ldots,\varepsilon(x_n)) & \\
\beta_\varepsilon(\texttt{let}\ (x_1,\ldots,x_n) = y\ \texttt{in}\ e) & = & \texttt{let}\ (x_1,\ldots,x_n) = \varepsilon(y)\ \texttt{in}\ \beta_\varepsilon(e) & \\
\beta_\varepsilon(x.(y)) & = & \varepsilon(x).(\varepsilon(y)) & \\
\beta_\varepsilon(x.(y) \leftarrow z) & = & \varepsilon(x).(\varepsilon(y)) \leftarrow \varepsilon(z) &
\end{array}
$$

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 maps a variable that occurs in the original expression to a variable in the resulting expression. When ε does not map x, we regard ε(x) = x; in other words, ε maps x to itself, by default.

in case, A(e1) is formed " let ... in e1' " and e1' is not a let-form.
Here "let ... in" stands for 0 or more nesting of let bindings.

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

$\mathcal{A}(c)$	$=$	c
$\mathcal{A}(op(x_1, \ldots, x_n))$	$=$	$op(x_1, \ldots, x_n)$
$\mathcal{A}(\texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2)$	$=$	$\texttt{if}\ x = y\ \texttt{then}\ \mathcal{A}(e_1)\ \texttt{else}\ \mathcal{A}(e_2)$
$\mathcal{A}(\texttt{if}\ x \leq y\ \texttt{then}\ e_1\ \texttt{else}\ e_2)$	$=$	$\texttt{if}\ x \leq y\ \texttt{then}\ \mathcal{A}(e_1)\ \texttt{else}\ \mathcal{A}(e_2)$
$\mathcal{A}(\texttt{let}\ x = e_1\ \texttt{in}\ e_2)$	$=$	$\texttt{let}\ \ldots\ \texttt{in}\ \texttt{let}\ x = e_1'\ \texttt{in}\ \mathcal{A}(e_2)$ $\mathcal{A}(e_1) = \texttt{let}\ \ldots\ \texttt{in}\ e_1'$ という形で ($\texttt{let}\ \ldots\ \texttt{in}$ は 0 個以上の let の列)、 e_1' は let でない
$\mathcal{A}(x)$	$=$	x
$\mathcal{A}(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2)$	$=$	$\texttt{let rec}\ x\ y_1\ \ldots\ y_n = \mathcal{A}(e_1)\ \texttt{in}\ \mathcal{A}(e_2)$
$\mathcal{A}(x\ y_1\ \ldots\ y_n)$	$=$	$x\ y_1\ \ldots\ y_n$
$\mathcal{A}((x_1, \ldots, x_n))$	$=$	$(x_1, \ldots, x_n)$
$\mathcal{A}(\texttt{let}\ (x_1, \ldots, x_n) = y\ \texttt{in}\ e)$	$=$	$\texttt{let}\ (x_1, \ldots, x_n) = y\ \texttt{in}\ \mathcal{A}(e)$
$\mathcal{A}(x.(y))$	$=$	$x.(y)$
$\mathcal{A}(x.(y) \leftarrow z)$	$=$	$x.(y) \leftarrow z$

図 8: ネストした let の簡約

Reduction of nested "let"

Example:

Suppose that:
A(M) =
let y = e1' in
let z = e2' in
M1

A(let x = M in N) will be
let y = e1' in
let z = e2' in
let x = M1 in
A(N)

... （の）場合 = in case ...

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

$$
\begin{array}{lcl}
\mathcal{I}_\varepsilon(c) & = & c \\
\mathcal{I}_\varepsilon(op(x_1, \ldots, x_n)) & = & op(x_1, \ldots, x_n) \\
\mathcal{I}_\varepsilon(\texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ x = y\ \texttt{then}\ \mathcal{I}_\varepsilon(e_1)\ \texttt{else}\ \mathcal{I}_\varepsilon(e_2) \\
\mathcal{I}_\varepsilon(\texttt{if}\ x \le y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ x \le y\ \texttt{then}\ \mathcal{I}_\varepsilon(e_1)\ \texttt{else}\ \mathcal{I}_\varepsilon(e_2) \\
\mathcal{I}_\varepsilon(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & = & \texttt{let}\ x = \mathcal{I}_\varepsilon(e_1)\ \texttt{in}\ \mathcal{I}_\varepsilon(e_2) \\
\mathcal{I}_\varepsilon(x) & = & x \\
\mathcal{I}_\varepsilon(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \varepsilon' = \varepsilon, x \mapsto ((y_1, \ldots, y_n), e_1)\ \text{として} \\
 & & \texttt{let rec}\ x\ y_1\ \ldots\ y_n = \mathcal{I}_{\varepsilon'}(e_1)\ \texttt{in}\ \mathcal{I}_{\varepsilon'}(e_2) \\
 & & \qquad\qquad\qquad size(e_1) \le th\ \text{の場合} \\
\mathcal{I}_\varepsilon(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \texttt{let rec}\ x\ y_1\ \ldots\ y_n = \mathcal{I}_\varepsilon(e_1)\ \texttt{in}\ \mathcal{I}_\varepsilon(e_2) \\
 & & \qquad\qquad\qquad size(e_1) > th\ \text{の場合} \\
\mathcal{I}_\varepsilon(x\ y_1\ \ldots\ y_n) & = & \alpha_{y_1 \mapsto z_1, \ldots, y_n \mapsto z_n}(e) \qquad \varepsilon(x) = ((z_1, \ldots, z_n), e)\ \text{の場合} \\
\mathcal{I}_\varepsilon(x\ y_1\ \ldots\ y_n) & = & x\ y_1\ \ldots\ y_n \qquad \varepsilon(x)\ \text{が定義されていない場合} \\
\mathcal{I}_\varepsilon((x_1, \ldots, x_n)) & = & (x_1, \ldots, x_n) \\
\mathcal{I}_\varepsilon(\texttt{let}\ (x_1, \ldots, x_n) = y\ \texttt{in}\ e) & = & \texttt{let}\ (x_1, \ldots, x_n) = y\ \texttt{in}\ \mathcal{I}_\varepsilon(e) \\
\mathcal{I}_\varepsilon(x.(y)) & = & x.(y) \\
\mathcal{I}_\varepsilon(x.(y) \leftarrow z) & = & x.(y) \leftarrow z
\end{array}
$$

ε is not defined for x

$$
\begin{array}{lcl}
size(c) & = & 1 \\
size(op(x_1, \ldots, x_n)) & = & 1 \\
size(\texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & 1 + size(e_1) + size(e_2) \\
size(\texttt{if}\ x \le y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & 1 + size(e_1) + size(e_2) \\
size(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & = & 1 + size(e_1) + size(e_2) \\
size(x) & = & 1 \\
size(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & 1 + size(e_1) + size(e_2) \\
size(x\ y_1\ \ldots\ y_n) & = & 1 \\
size((x_1, \ldots, x_n)) & = & 1 \\
size(\texttt{let}\ (x_1, \ldots, x_n) = y\ \texttt{in}\ e) & = & 1 + size(e) \\
size(x.(y)) & = & 1 \\
size(x.(y) \leftarrow z) & = & 1
\end{array}
$$

図 9: インライン展開。ε はサイズの小さい関数名を受け取って、仮引数と本体を返す写像。th はインライン展開する関数の最大サイズ（ユーザ指定）。

Inline expansion: the environment ε takes a name of small-sized function and gives its virtual argument names and body. "th" stands for the expansion threshold: the maximum-allowed size of the function. Its value is specified by the user (compiler's command-line option?).

Japanese translation:
1. C の場合: In case a condition C holds
2. それ以外の場合: otherwise
3. ε(x) と ε(y) が等しい定数: ε gives the same constant values for x and y
4. ε(x) と ε(y) が異なる定数: ε gives different contant values for x and y
5. ε(x) と ε(y) が定数: both ε(x) and ε(y) are constant values (more strictly, ε is defined for both x and y)

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

$$
\begin{array}{lcll}
\mathcal{F}_\varepsilon(c) & = & c & \\
\mathcal{F}_\varepsilon(op(x_1,\ldots,x_n)) & = & c & op(\varepsilon(x_1),\ldots,\varepsilon(x_n)) = c \text{ の場合} \\
\mathcal{F}_\varepsilon(op(x_1,\ldots,x_n)) & = & op(x_1,\ldots,x_n) & \text{それ以外の場合} \\
\mathcal{F}_\varepsilon(\texttt{if } x = y \texttt{ then } e_1 \texttt{ else } e_2) & = & \mathcal{F}_\varepsilon(e_1) & \varepsilon(x) \text{ と } \varepsilon(y) \text{ が等しい定数の場合} \\
\mathcal{F}_\varepsilon(\texttt{if } x = y \texttt{ then } e_1 \texttt{ else } e_2) & = & \mathcal{F}_\varepsilon(e_2) & \varepsilon(x) \text{ と } \varepsilon(y) \text{ が異なる定数の場合} \\
\mathcal{F}_\varepsilon(\texttt{if } x = y \texttt{ then } e_1 \texttt{ else } e_2) & = & \texttt{if } x = y \texttt{ then } \mathcal{F}_\varepsilon(e_1) \texttt{ else } \mathcal{F}_\varepsilon(e_2) & \text{それ以外の場合} \\
\mathcal{F}_\varepsilon(\texttt{if } x \leq y \texttt{ then } e_1 \texttt{ else } e_2) & = & \mathcal{F}_\varepsilon(e_1) & \varepsilon(x) \text{ と } \varepsilon(y) \text{ が定数で、} \varepsilon(x) \leq \varepsilon(y) \text{ の場合} \\
\mathcal{F}_\varepsilon(\texttt{if } x \leq y \texttt{ then } e_1 \texttt{ else } e_2) & = & \mathcal{F}_\varepsilon(e_2) & \varepsilon(x) \text{ と } \varepsilon(y) \text{ が定数で、} \varepsilon(x) > \varepsilon(y) \text{ の場合} \\
\mathcal{F}_\varepsilon(\texttt{if } x \leq y \texttt{ then } e_1 \texttt{ else } e_2) & = & \texttt{if } x \leq y \texttt{ then } \mathcal{F}_\varepsilon(e_1) \texttt{ else } \mathcal{F}_\varepsilon(e_2) & \text{それ以外の場合} \\
\mathcal{F}_\varepsilon(\texttt{let } x = e_1 \texttt{ in } e_2) & = & e_1' = \mathcal{F}_\varepsilon(e_1) \text{ として} & \\
 & & \texttt{let } x = e_1' \texttt{ in } \mathcal{F}_{\varepsilon, x \mapsto e_1'}(e_2) & \\
\mathcal{F}_\varepsilon(x) & = & x & \\
\mathcal{F}_\varepsilon(\texttt{let rec } x\ y_1\ \ldots\ y_n = e_1 \texttt{ in } e_2) & = & \texttt{let rec } x\ y_1\ \ldots\ y_n = \mathcal{F}_\varepsilon(e_1) \texttt{ in } \mathcal{F}_\varepsilon(e_2) & \\
\mathcal{F}_\varepsilon(x\ y_1\ \ldots\ y_n) & = & x\ y_1\ \ldots\ y_n & \\
\mathcal{F}_\varepsilon((x_1,\ldots,x_n)) & = & (x_1,\ldots,x_n) & \\
\mathcal{F}_\varepsilon(\texttt{let } (x_1,\ldots,x_n) = y \texttt{ in } e) & = & \texttt{let } x_1 = y_1 \texttt{ in } \ldots \texttt{ let } x_n = y_n \texttt{ in } \mathcal{F}_\varepsilon(e) & \\
 & & & \varepsilon(y) = (y_1,\ldots,y_n) \text{ の場合} \\
\mathcal{F}_\varepsilon(\texttt{let } (x_1,\ldots,x_n) = y \texttt{ in } e) & = & \texttt{let } (x_1,\ldots,x_n) = y \texttt{ in } \mathcal{F}_\varepsilon(e) & \\
\mathcal{F}_\varepsilon(x.(y)) & = & x.(y) & \\
\mathcal{F}_\varepsilon(x.(y) \leftarrow z) & = & x.(y) \leftarrow z &
\end{array}
$$

図 10: 定数畳み込み。ε は変数を受け取って、定数を返す写像。

Constant folding: ε is a mapping that takes a variable name and gives its associated constant value, if any.

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

$$
\begin{array}{lcll}
\mathcal{E}(c) & = & c & \\
\mathcal{E}(op(x_1,\ldots,x_n)) & = & op(x_1,\ldots,x_n) & \\
\mathcal{E}(\texttt{if}\ x=y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ x=y\ \texttt{then}\ \mathcal{E}(e_1)\ \texttt{else}\ \mathcal{E}(e_2) & \\
\mathcal{E}(\texttt{if}\ x\le y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ x\le y\ \texttt{then}\ \mathcal{E}(e_1)\ \texttt{else}\ \mathcal{E}(e_2) & \\
\mathcal{E}(\texttt{let}\ x=e_1\ \texttt{in}\ e_2) & = & \mathcal{E}(e_2) & \mathit{effect}(\mathcal{E}(e_1))=\mathit{false}\ \text{かつ}\ x\notin FV(\mathcal{E}(e_2))\ \text{の場合} \\
\mathcal{E}(\texttt{let}\ x=e_1\ \texttt{in}\ e_2) & = & \texttt{let}\ x=\mathcal{E}(e_1)\ \texttt{in}\ \mathcal{E}(e_2) & \text{それ以外の場合} \\
\mathcal{E}(x) & = & x & \\
\mathcal{E}(\texttt{let rec}\ x\ y_1\ \ldots\ y_n=e_1\ \texttt{in}\ e_2) & = & \mathcal{E}(e_2) & x\notin FV(\mathcal{E}(e_2))\ \text{の場合} \\
\mathcal{E}(\texttt{let rec}\ x\ y_1\ \ldots\ y_n=e_1\ \texttt{in}\ e_2) & = & \texttt{let rec}\ x\ y_1\ \ldots\ y_n=\mathcal{E}(e_1)\ \texttt{in}\ \mathcal{E}(e_2) & \text{それ以外の場合} \\
\mathcal{E}(x\ y_1\ \ldots\ y_n) & = & x\ y_1\ \ldots\ y_n & \\
\mathcal{E}((x_1,\ldots,x_n)) & = & (x_1,\ldots,x_n) & \\
\mathcal{E}(\texttt{let}\ (x_1,\ldots,x_n)=y\ \texttt{in}\ e) & = & \mathcal{E}(e) & \{x_1,\ldots,x_n\}\cap FV(\mathcal{E}(e))=\emptyset\ \text{の場合} \\
\mathcal{E}(\texttt{let}\ (x_1,\ldots,x_n)=y\ \texttt{in}\ e) & = & \texttt{let}\ (x_1,\ldots,x_n)=y\ \texttt{in}\ \mathcal{E}(e) & \text{それ以外の場合} \\
\mathcal{E}(x.(y)) & = & x.(y) & \\
\mathcal{E}(x.(y)\leftarrow z) & = & x.(y)\leftarrow z &
\end{array}
$$

effect : KNormal.t $\rightarrow$ bool

$$
\begin{array}{lcl}
\mathit{effect}(c) & = & \mathit{false} \\
\mathit{effect}(op(x_1,\ldots,x_n)) & = & \mathit{false} \\
\mathit{effect}(\texttt{if}\ x=y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \mathit{effect}(e_1)\vee\mathit{effect}(e_2) \\
\mathit{effect}(\texttt{if}\ x\le y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \mathit{effect}(e_1)\vee\mathit{effect}(e_2) \\
\mathit{effect}(\texttt{let}\ x=e_1\ \texttt{in}\ e_2) & = & \mathit{effect}(e_1)\vee\mathit{effect}(e_2) \\
\mathit{effect}(x) & = & \mathit{false} \\
\mathit{effect}(\texttt{let rec}\ x\ y_1\ \ldots\ y_n=e_1\ \texttt{in}\ e_2) & = & \mathit{effect}(e_2) \\
\mathit{effect}(x\ y_1\ \ldots\ y_n) & = & \mathit{true} \\
\mathit{effect}((x_1,\ldots,x_n)) & = & \mathit{false} \\
\mathit{effect}(\texttt{let}\ (x_1,\ldots,x_n)=y\ \texttt{in}\ e) & = & \mathit{effect}(e) \\
\mathit{effect}(x.(y)) & = & \mathit{false} \\
\mathit{effect}(x.(y)\leftarrow z) & = & \mathit{true}
\end{array}
$$

図 11: 不要定義削除 (1/2)

Elimination of redundant definitions (1/2)

FV: stands for "a set of free variables": x is free in e iff x's definition is not given in e. Free variables of e is a set of variables that are free in e.

The effect system used in this analysis collects all the defined variables in an expression.

Set notation
Φ: empty set
A ∪ B: set union,   A ∩ B: set intersection
A \ B: { a in A | a not in B }

FV : KNormal.t → S.t

$$
\begin{array}{lcl}
FV(c) & = & \emptyset \\
FV(op(x_1,\ldots,x_n)) & = & \{x_1,\ldots,x_n\} \\
FV(\texttt{if } x = y \texttt{ then } e_1 \texttt{ else } e_2) & = & \{x,y\} \cup FV(e_1) \cup FV(e_2) \\
FV(\texttt{if } x \le y \texttt{ then } e_1 \texttt{ else } e_2) & = & \{x,y\} \cup FV(e_1) \cup FV(e_2) \\
FV(\texttt{let } x = e_1 \texttt{ in } e_2) & = & FV(e_1) \cup (FV(e_2) \setminus \{x\}) \\
FV(x) & = & \{x\} \\
FV(\texttt{let rec } x\ y_1\ \ldots\ y_n = e_1 \texttt{ in } e_2) & = & ((FV(e_1) \setminus \{y_1,\ldots,y_n\}) \cup FV(e_2)) \setminus \{x\} \\
FV(x\ y_1\ \ldots\ y_n) & = & \{x, y_1,\ldots,y_n\} \\
FV((x_1,\ldots,x_n)) & = & \{x_1,\ldots,x_n\} \\
FV(\texttt{let } (x_1,\ldots,x_n) = y \texttt{ in } e) & = & \{y\} \cup (FV(e) \setminus \{x_1,\ldots,x_n\}) \\
FV(x.(y)) & = & \{x,y\} \\
FV(x.(y) \leftarrow z) & = & \{x,y,z\}
\end{array}
$$

図 12: 不要定義削除 (2/2)

Elimination of redundant definisions (2/2)

$$
\begin{array}{ll}
P\ ::= & \text{プログラム全体} \\
\quad (\{D_1,\ldots,D_n\}, e) & \text{トップレベル関数定義の集合とメインルーチンの式} \\
D\ ::= & \text{トップレベル関数定義} \\
\quad \mathrm{L}_x(y_1,\ldots,y_m)(z_1,\ldots,z_n) = e & \text{関数のラベルと仮引数、自由変数、および本体} \\
e\ ::= & \\
\quad c & \\
\quad op(x_1,\ldots,x_n) & \\
\quad \texttt{if } x = y \texttt{ then } e_1 \texttt{ else } e_2 & \\
\quad \texttt{if } x \le y \texttt{ then } e_1 \texttt{ else } e_2 & \\
\quad \texttt{let } x = e_1 \texttt{ in } e_2 & \\
\quad x & \\
\quad \mathit{make\_closure}\ x = (\mathrm{L}_x, (y_1,\ldots,y_n)) \texttt{ in } e & \text{クロージャ生成} \\
\quad \mathit{apply\_closure}(x, y_1,\ldots,y_n) & \text{クロージャを用いた関数呼び出し} \\
\quad \mathit{apply\_direct}(\mathrm{L}_x, y_1,\ldots,y_n) & \text{クロージャを用いない関数呼び出し (known function call)} \\
\quad (x_1,\ldots,x_n) & \\
\quad \texttt{let } (x_1,\ldots,x_n) = y \texttt{ in } e & \\
\quad x.(y) & \\
\quad x.(y) \leftarrow z &
\end{array}
$$

P: a set of the whole program that consists of definitions of top-level function definitions and a main routine.
D: the syntax of top-level function definition, which consists of function label (= name), a list of formal arguments, a list of free variables, and its function body.

make_closure: closure creation
apply_closure: function call using a closure
apply_direct: function call without using a closure (this more efficient than apply_closure)

Closure language
Closure conversion convertes a KNF form into an instance of the closure language. This syntactic definition gives the structure of the MinCaml program after closure conversion is performed on KNF.
Note that this definition does not have "let rec" any more

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

$$
\begin{array}{lcl}
\mathcal{C}(c) & = & c \\
\mathcal{C}(op(x_1,\ldots,x_n)) & = & op(x_1,\ldots,x_n) \\
\mathcal{C}(\texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ x = y\ \texttt{then}\ \mathcal{C}(e_1)\ \texttt{else}\ \mathcal{C}(e_2) \\
\mathcal{C}(\texttt{if}\ x \leq y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ x \leq y\ \texttt{then}\ \mathcal{C}(e_1)\ \texttt{else}\ \mathcal{C}(e_2) \\
\mathcal{C}(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & = & \texttt{let}\ x = \mathcal{C}(e_1)\ \texttt{in}\ \mathcal{C}(e_2) \\
\mathcal{C}(x) & = & x \\
\mathcal{C}(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \mathcal{D}\text{ に }\mathtt{L}_x(y_1,\ldots,y_n)(z_1,\ldots,z_m) = e_1'\text{ を加え、} \\
 & & \mathit{make\_closure}\ x = (\mathtt{L}_x,(z_1,\ldots,z_m))\ \texttt{in}\ e_2'\text{を返す} \\
 & & \quad\text{ただし } e_1' = \mathcal{C}(e_1),\ e_2' = \mathcal{C}(e_2), \\
 & & \quad FV(e_1') \setminus \{x,y_1,\ldots,y_n\} = \{z_1,\ldots,z_m\} \\
\mathcal{C}(x\ y_1\ \ldots\ y_n) & = & \mathit{apply\_closure}(x,y_1,\ldots,y_n) \\
\mathcal{C}((x_1,\ldots,x_n)) & = & (x_1,\ldots,x_n) \\
\mathcal{C}(\texttt{let}\ (x_1,\ldots,x_n) = y\ \texttt{in}\ e) & = & \texttt{let}\ (x_1,\ldots,x_n) = y\ \texttt{in}\ \mathcal{C}(e) \\
\mathcal{C}(x.(y)) & = & x.(y) \\
\mathcal{C}(x.(y) \leftarrow z) & = & x.(y) \leftarrow z
\end{array}
$$

FV : `Closure.t` $\rightarrow$ `S.t`

$$
\begin{array}{lcl}
FV(c) & = & \emptyset \\
FV(op(x_1,\ldots,x_n)) & = & \{x_1,\ldots,x_n\} \\
FV(\texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \{x,y\} \cup FV(e_1) \cup FV(e_2) \\
FV(\texttt{if}\ x \leq y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \{x,y\} \cup FV(e_1) \cup FV(e_2) \\
FV(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & = & FV(e_1) \cup (FV(e_2) \setminus \{x\}) \\
FV(x) & = & \{x\} \\
FV(\mathit{make\_closure}\ x = (\mathtt{L}_x,(y_1,\ldots,y_n))\ \texttt{in}\ e) & = & \{y_1,\ldots,y_n\} \cup (FV(e) \setminus \{x\}) \\
FV(\mathit{apply\_closure}(x,y_1,\ldots,y_n)) & = & \{x,y_1,\ldots,y_n\} \\
FV(\mathit{apply\_direct}(\mathtt{L}_x,y_1,\ldots,y_n)) & = & \{y_1,\ldots,y_n\} \\
FV((x_1,\ldots,x_n)) & = & \{x_1,\ldots,x_n\} \\
FV(\texttt{let}\ (x_1,\ldots,x_n) = y\ \texttt{in}\ e) & = & \{y\} \cup (FV(e) \setminus \{x_1,\ldots,x_n\}) \\
FV(x.(y)) & = & \{x,y\} \\
FV(x.(y) \leftarrow z) & = & \{x,y,z\}
\end{array}
$$

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変数。

Naive closure conversion. D is a global variable that holds a set of top-level function definitions.

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

$$
\begin{array}{lcll}
\mathcal{C}_s(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \mathcal{D}\text{ に }\mathrm{L}_x(y_1,\ldots,y_n)() = e_1'\text{ を加え、} & \\
 & & \mathit{make\_closure}\ x = (\mathrm{L}_x, ())\ \texttt{in}\ e_2'\text{を返す} & \\
 & & \quad\text{ただし } e_1' = \mathcal{C}_{s'}(e_1),\ e_2' = \mathcal{C}_{s'}(e_2),\ s' = s \cup \{x\}, & \\
 & & \quad FV(e_1') \setminus \{y_1,\ldots,y_n\} = \emptyset\text{ の場合} & \\
\mathcal{C}_s(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \mathcal{D}\text{ に }\mathrm{L}_x(y_1,\ldots,y_n)(z_1,\ldots,z_m) = e_1'\text{ を加え、} & \\
 & & \mathit{make\_closure}\ x = (\mathrm{L}_x, (z_1,\ldots,z_m))\ \texttt{in}\ e_2'\text{を返す} & \\
 & & \quad\text{ただし } e_1' = \mathcal{C}_s(e_1),\ e_2' = \mathcal{C}_s(e_2), & \\
 & & \quad FV(e_1') \setminus \{y_1,\ldots,y_n\} \neq \emptyset, & \\
 & & \quad FV(e_1') \setminus \{x, y_1,\ldots,y_n\} = \{z_1,\ldots,z_m\}\text{ の場合} & \\
\mathcal{C}_s(x\ y_1\ \ldots\ y_n) & = & \mathit{apply\_closure}(x, y_1,\ldots,y_n) & x \notin s\text{ の場合} \\
\mathcal{C}_s(x\ y_1\ \ldots\ y_n) & = & \mathit{apply\_direct}(\mathrm{L}_x, y_1,\ldots,y_n) & x \in s\text{ の場合}
\end{array}
$$

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。

An improved but not so smart enough closure conversion. s is a set of names of functions that are known to contain no free variables in their definitions.

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

$$
\begin{array}{lcll}
\mathcal{C}_s(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \mathcal{D}\text{ に }\mathrm{L}_x(y_1,\ldots,y_n)() = e_1'\text{ を加え、} & \\
 & & \mathit{make\_closure}\ x = (\mathrm{L}_x, ())\ \texttt{in}\ e_2'\text{を返す} & \\
 & & \quad\text{ただし } e_1' = \mathcal{C}_{s'}(e_1),\ e_2' = \mathcal{C}_{s'}(e_2),\ s' = s \cup \{x\}, & \\
 & & \quad FV(e_1') \setminus \{y_1,\ldots,y_n\} = \emptyset\text{ かつ } x \in FV(e_2')\text{ の場合} & \\
\mathcal{C}_s(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \mathcal{D}\text{ に }\mathrm{L}_x(y_1,\ldots,y_n)() = e_1'\text{ を加え、}e_2'\text{を返す} & \\
 & & \quad\text{ただし } e_1' = \mathcal{C}_{s'}(e_1),\ e_2' = \mathcal{C}_{s'}(e_2),\ s' = s \cup \{x\}, & \\
 & & \quad FV(e_1') \setminus \{y_1,\ldots,y_n\} = \emptyset\text{ かつ } x \notin FV(e_2')\text{ の場合} & \\
\mathcal{C}_s(\texttt{let rec}\ x\ y_1\ \ldots\ y_n = e_1\ \texttt{in}\ e_2) & = & \mathcal{D}\text{ に }\mathrm{L}_x(y_1,\ldots,y_n)(z_1,\ldots,z_m) = e_1'\text{ を加え、} & \\
 & & \mathit{make\_closure}\ x = (\mathrm{L}_x, (z_1,\ldots,z_m))\ \texttt{in}\ e_2'\text{を返す} & \\
 & & \quad\text{ただし } e_1' = \mathcal{C}_s(e_1),\ e_2' = \mathcal{C}_s(e_2), & \\
 & & \quad FV(e_1') \setminus \{y_1,\ldots,y_n\} \neq \emptyset, & \\
 & & \quad FV(e_1') \setminus \{x, y_1,\ldots,y_n\} = \{z_1,\ldots,z_m\}\text{ の場合} & \\
\mathcal{C}_s(x\ y_1\ \ldots\ y_n) & = & \mathit{apply\_closure}(x, y_1,\ldots,y_n) & x \notin s\text{ の場合} \\
\mathcal{C}_s(x\ y_1\ \ldots\ y_n) & = & \mathit{apply\_direct}(\mathrm{L}_x, y_1,\ldots,y_n) & x \in s\text{ の場合}
\end{array}
$$

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

Smarter closure conversion.

P ::=	
$(\{D_1, \ldots, D_n\}, E)$	
D ::=	
$\mathtt{L}_x(y_1, \ldots, y_n) = E$	
E ::=	命令の列
$x \leftarrow e; E$	代入
e	返値
e ::=	式
c	即値
$\mathtt{L}_x$	ラベル
$op(x_1, \ldots, x_n)$	算術演算
$\mathtt{if}\ x = y\ \mathtt{then}\ E_1\ \mathtt{else}\ E_2$	比較＆分岐
$\mathtt{if}\ x \leq y\ \mathtt{then}\ E_1\ \mathtt{else}\ E_2$	比較＆分岐
x	`mov` 命令
$apply\_closure(x, y_1, \ldots, y_n)$	クロージャを用いた関数呼び出し
$apply\_direct(\mathtt{L}_x, y_1, \ldots, y_n)$	クロージャを用いない関数呼び出し
$x.(y)$	ロード
$x.(y) \leftarrow z$	ストア
$\mathtt{save}(x, y)$	変数 x の値をスタック位置 y に退避する
$\mathtt{restore}(y)$	スタック位置 y から値を復元する

図 17: 仮想マシンコードの構文

$$
\begin{array}{lcl}
\mathcal{V} : \texttt{Closure.prog} \to \texttt{SparcAsm.prog} & & \\
\mathcal{V}((\{D_1,\ldots,D_n\},e)) & = & (\{\mathcal{V}(D_1),\ldots,\mathcal{V}(D_n)\},\mathcal{V}(e)) \\
 & & \\
\mathcal{V} : \texttt{Closure.fundef} \to \texttt{SparcAsm.fundef} & & \\
\mathcal{V}(\texttt{L}_x(y_1,\ldots,y_n)(z_1,\ldots,z_n)=e) & = & \texttt{L}_x(y_1,\ldots,y_n)=z_1 \leftarrow \texttt{R}_0.(4);\ldots;z_n \leftarrow \texttt{R}_0.(4n);\mathcal{V}(e) \\
 & & \\
\mathcal{V} : \texttt{Closure.t} \to \texttt{SparcAsm.t} & & \\
\mathcal{V}(c) & = & c \\
\mathcal{V}(op(x_1,\ldots,x_n)) & = & op(x_1,\ldots,x_n) \\
\mathcal{V}(\texttt{if}\ x=y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ x=y\ \texttt{then}\ \mathcal{V}(e_1)\ \texttt{else}\ \mathcal{V}(e_2) \\
\mathcal{V}(\texttt{if}\ x\le y\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ x\le y\ \texttt{then}\ \mathcal{V}(e_1)\ \texttt{else}\ \mathcal{V}(e_2) \\
\mathcal{V}(\texttt{let}\ x=e_1\ \texttt{in}\ e_2) & = & x \leftarrow \mathcal{V}(e_1);\mathcal{V}(e_2) \\
\mathcal{V}(x) & = & x \\
\mathcal{V}(\mathit{make\_closure}\ x=(\texttt{L}_x,(y_1,\ldots,y_n))\ \texttt{in}\ e) & = & x \leftarrow \texttt{R}_{\texttt{hp}};\texttt{R}_{\texttt{hp}} \leftarrow \texttt{R}_{\texttt{hp}}+4(n+1);z \leftarrow \texttt{L}_x;x.(0) \leftarrow z; \\
 & & x.(4) \leftarrow y_1;\ldots;x.(4n) \leftarrow y_n;\mathcal{V}(e) \\
\mathcal{V}(\mathit{apply\_closure}(x,y_1,\ldots,y_n)) & = & \mathit{apply\_closure}(x,y_1,\ldots,y_n) \\
\mathcal{V}(\mathit{apply\_direct}(\texttt{L}_x,y_1,\ldots,y_n)) & = & \mathit{apply\_direct}(\texttt{L}_x,y_1,\ldots,y_n) \\
\mathcal{V}((x_1,\ldots,x_n)) & = & y \leftarrow \texttt{R}_{\texttt{hp}};\texttt{R}_{\texttt{hp}} \leftarrow \texttt{R}_{\texttt{hp}}+4n; \\
 & & y.(0) \leftarrow x_1;\ldots;y.(4(n-1)) \leftarrow x_n;y \\
\mathcal{V}(\texttt{let}\ (x_1,\ldots,x_n)=y\ \texttt{in}\ e) & = & \{x_1,\ldots,x_n\} \cap FV(e) = \{x_{i_1},\ldots,x_{i_m}\}\ \text{として} \\
 & & x_{i_1} \leftarrow y.(4(i_1-1));\ldots;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
\end{array}
$$

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。$\texttt{R}_{\texttt{hp}}$ 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、$E_1=(x_1 \leftarrow e_1;\ldots;x_n \leftarrow e_n;e)$ として、$x_1 \leftarrow e_1;\ldots;x_n \leftarrow e_n;x \leftarrow e;E_2$ の略記。

$$
\begin{array}{lcl}
FV : \texttt{S.t} \to \texttt{SparcAsm.t} \to \texttt{S.t} & & \\
FV_s(x \leftarrow e; E) & = & s' = FV_s(E) \setminus \{x\} \text{ として } FV_{s'}(e) \\
FV_s(e) & = & FV_s(e) \\
 & & \\
FV : \texttt{S.t} \to \texttt{SparcAsm.exp} \to \texttt{S.t} & & \\
FV_s(c) & = & s \\
FV_s(\texttt{L}_x) & = & s \\
FV_s(op(x_1, \ldots, x_n)) & = & \{x_1, \ldots, x_n\} \cup s \\
FV_s(\texttt{if } x = y \texttt{ then } E_1 \texttt{ else } E_2) & = & \{x, y\} \cup FV_s(E_1) \cup FV_s(E_2) \\
FV_s(\texttt{if } x \leq y \texttt{ then } E_1 \texttt{ else } E_2) & = & \{x, y\} \cup FV_s(E_1) \cup FV_s(E_2) \\
FV_s(x) & = & \{x\} \cup s \\
FV_s(\mathit{apply\_closure}(x, y_1, \ldots, y_n)) & = & \{x, y_1, \ldots, y_n\} \cup s \\
FV_s(\mathit{apply\_direct}(\texttt{L}_x, y_1, \ldots, y_n)) & = & \{y_1, \ldots, y_n\} \cup s \\
FV_s(x.(y)) & = & \{x, y\} \cup s \\
FV_s(x.(y) \leftarrow z) & = & \{x, y, z\} \cup s \\
FV_s(\texttt{save}(x, y)) & = & \{x\} \cup s \\
FV_s(\texttt{restore}(y)) & = & s
\end{array}
$$

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。

$$
\begin{array}{lcll}
\multicolumn{4}{l}{\mathcal{R} : \texttt{SparcAsm.prog} \to \texttt{SparcAsm.prog}} \\
\mathcal{R}((\{D_1, \ldots, D_n\}, E)) & = & (\{\mathcal{R}(D_1), \ldots, \mathcal{R}(D_n)\}, \mathcal{R}_\emptyset(E, x, ())) & x \text{ はダミーの fresh な変数} \\
\\
\multicolumn{4}{l}{\mathcal{R} : \texttt{SparcAsm.fundef} \to \texttt{SparcAsm.fundef}} \\
\mathcal{R}(\texttt{L}_x(y_1, \ldots, y_n) = E) & = & \texttt{L}_x(\texttt{R}_1, \ldots, \texttt{R}_n) = \mathcal{R}_{x \mapsto \texttt{R}_0, y_1 \mapsto \texttt{R}_1, \ldots, y_n \mapsto \texttt{R}_n}(E, \texttt{R}_0, \texttt{R}_0) & \\
\\
\multicolumn{4}{l}{\mathcal{R} : \texttt{Id.t M.t} \to \texttt{SparcAsm.t} \times \texttt{Id.t} \times \texttt{SparcAsm.t} \to \texttt{SparcAsm.t} \times \texttt{Id.t M.t}} \\
\mathcal{R}_\varepsilon((x \leftarrow e; E), z_{\texttt{dest}}, E_{\texttt{cont}}) & = & E'_{\texttt{cont}} = (z_{\texttt{dest}} \leftarrow E; E_{\texttt{cont}}), & \\
 & & \mathcal{R}_\varepsilon(e, x, E'_{\texttt{cont}}) = (E', \varepsilon'), & \\
 & & r \notin \{\varepsilon'(y) \mid y \in FV(E'_{\texttt{cont}})\}, & \\
 & & \mathcal{R}_{\varepsilon', x \mapsto r}(E, z_{\texttt{dest}}, E_{\texttt{cont}}) = (E'', \varepsilon'') \text{ として} & \\
 & & ((r \leftarrow E'; E''), \varepsilon'') & x \text{ がレジスタでない場合} \\
\mathcal{R}_\varepsilon((r \leftarrow e; E), z_{\texttt{dest}}, E_{\texttt{cont}}) & = & E'_{\texttt{cont}} = (z_{\texttt{dest}} \leftarrow E; E_{\texttt{cont}}), & \\
 & & \mathcal{R}_\varepsilon(e, r, E'_{\texttt{cont}}) = (E', \varepsilon'), & \\
 & & \mathcal{R}_{\varepsilon'}(E, z_{\texttt{dest}}, E_{\texttt{cont}}) = (E'', \varepsilon'') \text{ として} & \\
 & & ((r \leftarrow E'; E''), \varepsilon'') & \\
\mathcal{R}_\varepsilon(e, x, E_{\texttt{cont}}) & = & \mathcal{R}_\varepsilon(e, x, E_{\texttt{cont}}) & \text{(次図参照)}
\end{array}
$$

図 20: 単純なレジスタ割り当て $\mathcal{R}(P)$, $\mathcal{R}(D)$ および $\mathcal{R}_\varepsilon(E, z_{\texttt{dest}}, E_{\texttt{cont}})$。$\varepsilon$ は変数からレジスタへの写像、$z_{\texttt{dest}}$ は E の結果をセットする変数、$E_{\texttt{cont}}$ は E の後に実行される命令の列。$\mathcal{R}_\varepsilon(E, x, E_{\texttt{cont}})$ 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]

$$
\begin{array}{lcl}
\mathcal{R} : \texttt{Id.t M.t} \to \texttt{SparcAsm.exp} \times \texttt{Id.t} \times \texttt{SparcAsm.t} \to \texttt{SparcAsm.t} \times \texttt{Id.t M.t} & & \\
\mathcal{R}_\varepsilon(c, z_{\texttt{dest}}, E_{\texttt{cont}}) & = & (c, \varepsilon) \\
\mathcal{R}_\varepsilon(\texttt{L}_x, z_{\texttt{dest}}, E_{\texttt{cont}}) & = & (\texttt{L}_x, \varepsilon) \\
\mathcal{R}_\varepsilon(op(x_1, \ldots, x_n), z_{\texttt{dest}}, E_{\texttt{cont}}) & = & (op(\varepsilon(x_1), \ldots, \varepsilon(x_n)), \varepsilon) \\
\mathcal{R}_\varepsilon(\texttt{if}\ x = y\ \texttt{then}\ E_1\ \texttt{else}\ E_2, z_{\texttt{dest}}, E_{\texttt{cont}}) & = & \mathcal{R}_\varepsilon(E_1, z_{\texttt{dest}}, E_{\texttt{cont}}) = (E'_1, \varepsilon_1), \\
 & & \mathcal{R}_\varepsilon(E_2, z_{\texttt{dest}}, E_{\texttt{cont}}) = (E'_2, \varepsilon_2), \\
 & & \varepsilon' = \{z \mapsto r \mid \varepsilon_1(z) = \varepsilon_2(z) = r\}, \\
 & & \{z_1, \ldots, z_n\} = \\
 & & \quad\quad (FV(E_{\texttt{cont}}) \setminus \{z_{\texttt{dest}}\} \setminus dom(\varepsilon')) \cap dom(\varepsilon)\ \text{として} \\
 & & ((\texttt{save}(\varepsilon(z_1), z_1); \ldots; \texttt{save}(\varepsilon(z_n), z_n); \\
 & & \quad \texttt{if}\ \varepsilon(x) \le \varepsilon(y)\ \texttt{then}\ E'_1\ \texttt{else}\ E'_2), \varepsilon') \\
\mathcal{R}_\varepsilon(\texttt{if}\ x \le y\ \texttt{then}\ E_1\ \texttt{else}\ E_2, z_{\texttt{dest}}, E_{\texttt{cont}}) & = & \text{同様} \\
\mathcal{R}_\varepsilon(x, z_{\texttt{dest}}, E_{\texttt{cont}}) & = & (\varepsilon(x), \varepsilon) \\
\mathcal{R}_\varepsilon(apply\_closure(x, y_1, \ldots, y_n), z_{\texttt{dest}}, E_{\texttt{cont}}) & = & \{z_1, \ldots, z_n\} = (FV(E_{\texttt{cont}}) \setminus \{z_{\texttt{dest}}\}) \cap dom(\varepsilon)\ \text{として} \\
 & & ((\texttt{save}(\varepsilon(z_1), z_1); \ldots; \texttt{save}(\varepsilon(z_n), z_n); \\
 & & \quad apply\_closure(\varepsilon(x), \varepsilon(y_1), \ldots, \varepsilon(y_n))), \emptyset) \\
\mathcal{R}_\varepsilon(apply\_direct(\texttt{L}_x, y_1, \ldots, y_n), z_{\texttt{dest}}, E_{\texttt{cont}}) & = & \text{同様} \\
\mathcal{R}_\varepsilon(x.(y), z_{\texttt{dest}}, E_{\texttt{cont}}) & = & (\varepsilon(x).(\varepsilon(y)), \varepsilon) \\
\mathcal{R}_\varepsilon(x.(y) \leftarrow z, z_{\texttt{dest}}, E_{\texttt{cont}}) & = & (\varepsilon(x).(\varepsilon(y)) \leftarrow \varepsilon(z), \varepsilon) \\
\mathcal{R}_\varepsilon(\texttt{save}(x, y), z_{\texttt{dest}}, E_{\texttt{cont}}) & = & (\texttt{save}(\varepsilon(x), y), \varepsilon) \\
\mathcal{R}_\varepsilon(\texttt{restore}(y), z_{\texttt{dest}}, E_{\texttt{cont}}) & = & (\texttt{restore}(y), \varepsilon)
\end{array}
$$

図 21: 単純なレジスタ割り当て $\mathcal{R}_\varepsilon(e, z_{\texttt{dest}}, E_{\texttt{cont}})$。$\mathcal{R}_\varepsilon(e)$ 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、$\mathcal{R}_\varepsilon(e) = \mathcal{R}_\varepsilon(x \leftarrow \texttt{restore}(x); e)$ 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]

$$
\begin{array}{lcl}
\mathcal{T} : \mathtt{Id.t} \to \mathtt{SparcAsm.t} \times \mathtt{Id.t} \to \mathtt{bool} \times \mathtt{S.t} & & \\
\mathcal{T}_x((y \leftarrow e; E), z_{\mathtt{dest}}) & = & \mathcal{T}_x(e, y) = (c_1, s_1) \text{ として、もし } c_1 \text{ ならば } (\mathit{true}, s_1) \\
 & & \text{そうでなければ } \mathcal{T}_x(E, z_{\mathtt{dest}}) = (c_2, s_2) \text{ として } (c_2, s_1 \cup s_2) \\
\mathcal{T}_x(e, z_{\mathtt{dest}}) & = & \mathcal{T}_x(e, z_{\mathtt{dest}})
\end{array}
$$

$$
\begin{array}{lcll}
\mathcal{T} : \mathtt{Id.t} \to \mathtt{SparcAsm.exp} \times \mathtt{Id.t} \to \mathtt{bool} \times \mathtt{S.t} & & & \\
\mathcal{T}_x(x, z_{\mathtt{dest}}) & = & (\mathit{false}, \{z_{\mathtt{dest}}\}) & \\
\mathcal{T}_x(\mathtt{if}\ y = z\ \mathtt{then}\ E_1\ \mathtt{else}\ E_2, z_{\mathtt{dest}}) & = & \mathcal{T}_x(E_1, z_{\mathtt{dest}}) = (c_1, s_1), & \\
 & & \mathcal{T}_x(E_2, z_{\mathtt{dest}}) = (c_2, s_2) \text{ として} & \\
 & & (c_1 \wedge c_2, s_1 \cup s_2) & \\
\mathcal{T}_x(\mathtt{if}\ y \leq z\ \mathtt{then}\ E_1\ \mathtt{else}\ E_2, z_{\mathtt{dest}}) & = & \text{同上} & \\
\mathcal{T}_x(\mathit{apply\_closure}(y_0, y_1, \ldots, y_n), z_{\mathtt{dest}}) & = & (\mathit{true}, \{\mathtt{R}_i \mid x = y_i\}) & \\
\mathcal{T}_x(\mathit{apply\_direct}(\mathtt{L}_y, y_1, \ldots, y_n), z_{\mathtt{dest}}) & = & \text{同上} & \\
\mathcal{T}_x(e, z_{\mathtt{dest}}) & = & (\mathit{false}, \emptyset) & \text{それ以外の場合}
\end{array}
$$

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 $\mathcal{T}_x(E, z_{\mathtt{dest}})$ および $\mathcal{T}_x(e, z_{\mathtt{dest}})$。$E$ 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、$\mathcal{T}_x(E'_{\mathtt{cont}}, z_{\mathtt{dest}}) = (c, s)$ として、できれば $r \in s$ とする。[ファイル `regAlloc.target-nospill.ml` 参照]

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

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

$$
\begin{array}{lcl}
\mathcal{S} : \texttt{SparcAsm.prog} \to \texttt{string} & & \\
\mathcal{S}((\{D_1, \ldots, D_n\}, E)) & = & \texttt{.section ".text"} \\
 & & \mathcal{S}(D_1) \\
 & & \ldots \\
 & & \mathcal{S}(D_n) \\
 & & \texttt{.global min\_caml\_start} \\
 & & \texttt{min\_caml\_start:} \\
 & & \texttt{save \%sp, -112, \%sp} \\
 & & \mathcal{S}(E, \texttt{\%g0}) \\
 & & \texttt{ret} \\
 & & \texttt{restore} \\
 & & \\
\mathcal{S} : \texttt{SparcAsm.fundef} \to \texttt{string} & & \\
\mathcal{S}(\texttt{L}_x(y_1, \ldots, y_n) = E) & = & x\texttt{:} \\
 & & \mathcal{S}(E, \texttt{R}_0) \\
 & & \texttt{retl} \\
 & & \texttt{nop} \\
 & & \\
\mathcal{S} : \texttt{SparcAsm.t} \times \texttt{Id.t} \to \texttt{string} & & \\
\mathcal{S}((x \leftarrow e; E), z_{\texttt{dest}}) & = & \mathcal{S}(e, x) \\
 & & \mathcal{S}(E, z_{\texttt{dest}}) \\
\mathcal{S}(e, z_{\texttt{dest}}) & = & \mathcal{S}(e, z_{\texttt{dest}})
\end{array}
$$

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

$$
\begin{array}{lcl}
\mathcal{S} : \texttt{SparcAsm.exp} \times \texttt{Id.t} \to \texttt{string} & & \\
\mathcal{S}(c, z_{\texttt{dest}}) & = & \texttt{set}\ c, z_{\texttt{dest}} \\
\mathcal{S}(\texttt{L}_x, z_{\texttt{dest}}) & = & \texttt{set}\ \texttt{L}_x, z_{\texttt{dest}} \\
\mathcal{S}(op(x_1, \ldots, x_n), z_{\texttt{dest}}) & = & op\ x_1, \ldots, x_n, z_{\texttt{dest}} \\
\mathcal{S}(\texttt{if}\ x = y\ \texttt{then}\ E_1\ \texttt{else}\ E_2, z_{\texttt{dest}}) & = & \texttt{cmp}\ x, y \\
 & & \texttt{bne}\ b_1 \\
 & & \texttt{nop} \\
 & & \mathcal{S}(E_1, z_{\texttt{dest}}) \\
 & & \texttt{b}\ b_2 \\
 & & \texttt{nop} \\
 & & b_1\texttt{:} \\
 & & \mathcal{S}(E_2, z_{\texttt{dest}}) \\
 & & b_2\texttt{:} \\
\mathcal{S}(\texttt{if}\ x \leq y\ \texttt{then}\ E_1\ \texttt{else}\ E_2, z_{\texttt{dest}}) & = & \text{同様} \\
\mathcal{S}(x, z_{\texttt{dest}}) & = & \texttt{mov}\ x, z_{\texttt{dest}} \\
\mathcal{S}(apply\_closure(x, y_1, \ldots, y_n), z_{\texttt{dest}}) & = & shuffle((x, y_1, \ldots, y_n), (\texttt{R}_0, \texttt{R}_1, \ldots, \texttt{R}_n)) \\
 & & \texttt{st}\ \texttt{R}_{\texttt{ra}}, [\texttt{R}_{\texttt{st}} + 4\#\varepsilon] \\
 & & \texttt{ld}\ [\texttt{R}_0], \texttt{R}_{n+1} \\
 & & \texttt{call}\ \texttt{R}_{n+1} \\
 & & \texttt{add}\ \texttt{R}_{\texttt{st}}, 4(\#\varepsilon + 1), \texttt{R}_{\texttt{st}}\ \texttt{!}\ \textit{delay slot} \\
 & & \texttt{sub}\ \texttt{R}_{\texttt{st}}, 4(\#\varepsilon + 1), \texttt{R}_{\texttt{st}} \\
 & & \texttt{ld}\ [\texttt{R}_{\texttt{st}} + 4\#\varepsilon], \texttt{R}_{\texttt{ra}} \\
 & & \texttt{mov}\ \texttt{R}_0, z_{\texttt{dest}} \\
\mathcal{S}(apply\_direct(\texttt{L}_x, y_1, \ldots, y_n), z_{\texttt{dest}}) & = & shuffle((y_1, \ldots, y_n), (\texttt{R}_1, \ldots, \texttt{R}_n)) \\
 & & \texttt{st}\ \texttt{R}_{\texttt{ra}}, [\texttt{R}_{\texttt{st}} + 4\#\varepsilon] \\
 & & \texttt{call}\ x \\
 & & \texttt{add}\ \texttt{R}_{\texttt{st}}, 4(\#\varepsilon + 1), \texttt{R}_{\texttt{st}}\ \texttt{!}\ \textit{delay slot} \\
 & & \texttt{sub}\ \texttt{R}_{\texttt{st}}, 4(\#\varepsilon + 1), \texttt{R}_{\texttt{st}} \\
 & & \texttt{ld}\ [\texttt{R}_{\texttt{st}} + 4\#\varepsilon], \texttt{R}_{\texttt{ra}} \\
 & & \texttt{mov}\ \texttt{R}_0, z_{\texttt{dest}} \\
\mathcal{S}(x.(y), z_{\texttt{dest}}) & = & \texttt{ld}\ [x + y], z_{\texttt{dest}} \\
\mathcal{S}(x.(y) \leftarrow z, z_{\texttt{dest}}) & = & \texttt{st}\ z, [x + y] \\
\mathcal{S}(\texttt{save}(x, y), z_{\texttt{dest}}) & = & \text{もし}\ y \notin dom(\varepsilon)\ \text{なら}\ \varepsilon\ \text{に}\ y \mapsto 4\#\varepsilon\ \text{を加えて} \\
 & & \texttt{st}\ x, [\texttt{R}_{\texttt{st}} + \varepsilon(y)] \\
\mathcal{S}(\texttt{restore}(y), z_{\texttt{dest}}) & = & \texttt{ld}\ [\texttt{R}_{\texttt{st}} + \varepsilon(y)], z_{\texttt{dest}}
\end{array}
$$

図 25: 単純なアセンブリ生成 $\mathcal{S}(e, z_{\texttt{dest}})$。$\varepsilon$ 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。$shuffle((x_1, \ldots, x_n), (r_1, \ldots, r_n))$ は x_1, …, x_n を r_1, …, r_n に適切な順序で移動する命令。

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

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

$$
\begin{array}{lcl}
\mathcal{S} : \texttt{SparcAsm.fundef} \to \texttt{string} & & \\
\mathcal{S}(\texttt{L}_x(y_1, \ldots, y_n) = E) & = & \mathcal{S}_\emptyset(E, \texttt{tail}) = (s, S) \text{ として} \\
& & x\texttt{:} \\
& & S \\
\\
\mathcal{S} : \texttt{S.t} \to \texttt{SparcAsm.exp} \times \texttt{Id.t} \to \texttt{S.t} \times \texttt{string} & & \\
\mathcal{S}_s(\texttt{if } x = y \texttt{ then } E_1 \texttt{ else } E_2, \texttt{tail}) & = & \mathcal{S}_s(E_1, \texttt{tail}) = (s_1, S_1), \\
& & \mathcal{S}_s(E_2, \texttt{tail}) = (s_2, S_2) \text{ として} \\
& & (\emptyset, \\
& & \ \texttt{cmp } x, y \\
& & \ \texttt{bne } b \\
& & \ \texttt{nop} \\
& & \ S_1 \\
& & \ b\texttt{:} \\
& & \ S_2) \\
\mathcal{S}_s(\texttt{if } x \le y \texttt{ then } E_1 \texttt{ else } E_2, \texttt{tail}) & = & \text{同様} \\
\mathcal{S}_s(\mathit{apply\_closure}(x, y_1, \ldots, y_n), \texttt{tail}) & = & (\emptyset, \\
& & \ \mathit{shuffle}((x, y_1, \ldots, y_n), (\texttt{R}_0, \texttt{R}_1, \ldots, \texttt{R}_n)) \\
& & \ \texttt{ld } [\texttt{R}_0], \texttt{R}_{n+1} \\
& & \ \texttt{jmp } \texttt{R}_{n+1} \\
& & \ \texttt{nop}) \\
\mathcal{S}_s(\mathit{apply\_direct}(\texttt{L}_x, y_1, \ldots, y_n), \texttt{tail}) & = & (\emptyset, \\
& & \ \mathit{shuffle}((y_1, \ldots, y_n), (\texttt{R}_1, \ldots, \texttt{R}_n)) \\
& & \ \texttt{b } x \\
& & \ \texttt{nop}) \\
\mathcal{S}_s(e, \texttt{tail}) & = & \mathcal{S}_s(e, \texttt{R}_0) = (s', S) \text{ として} \\
& & (\emptyset, \\
& & \ S \\
& & \ \texttt{retl} \\
& & \ \texttt{nop}) \qquad\qquad\qquad \text{上述以外の場合}
\end{array}
$$

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 $\mathcal{S}_s(e, z_{\texttt{dest}})$。$z_{\texttt{dest}} = \texttt{tail}$ の場合が末尾。