

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

図 1: MinCaml の抽象構文 (型は省略)

**Abstract syntax of MinCaml (Type is omitted)**

$\tau ::=$	型	<b>Types in MinCaml is</b>
$\pi$	プリミティブ型	<b>  Primitive type</b>
$\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau$	関数型	<b>  Functional type</b>
$\tau_1 \times \dots \times \tau_n$	組型	<b>  Tuple type</b>
$\tau \text{ array}$	配列型	<b>  Array type</b>
$\alpha$	型変数 (型推論用)	<b>  Type variable (used by the type inference algorithm)</b>

図 2: MinCaml の型

**MinCaml types**

op is a primitive operator that takes values  
of  $\pi_1, \dots, \pi_n$  and gives a value of  $\pi$

c is a constant member of  $\pi$

s is a constant member of  $\pi$

$c$  は  $\pi$  型の定数

$op$  は  $\pi_1, \dots, \pi_n$  型の値を受け取って  $\pi$  型の値を返すプリミティブ演算

$\frac{}{\Gamma \vdash c : \pi}$	$\frac{\Gamma \vdash e_1 : \pi_1 \quad \dots \quad \Gamma \vdash e_n : \pi_n}{\Gamma \vdash op(e_1, \dots, e_n) : \pi}$	
$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau}$	$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2}$	$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$
$\frac{\Gamma, x : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau, y_1 : \tau_1, \dots, y_n : \tau_n \vdash e_1 : \tau \quad \Gamma, x : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2 : \tau'}$		$\frac{\Gamma \vdash e : \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \tau \quad \Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash e \ e_1 \ \dots \ e_n : \tau}$
$\frac{\Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash (e_1, \dots, e_n) : \tau_1 \times \dots \times \tau_n}$	$\frac{\Gamma \vdash e_1 : \tau_1 \times \dots \times \tau_n \quad \Gamma, x_1 : \tau_1, \dots, x_n : \tau_n \vdash e_2 : \tau}{\Gamma \vdash \text{let } (x_1, \dots, x_n) = e_1 \text{ in } e_2 : \tau}$	
$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{Array.create } e_1 \ e_2 : \tau \text{ array}}$		
$\frac{\Gamma \vdash e_1 : \tau \text{ array} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1.(e_2) : \tau}$	$\frac{\Gamma \vdash e_1 : \tau \text{ array} \quad \Gamma \vdash e_2 : \text{int} \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash e_1.(e_2) \leftarrow e_3 : \text{unit}}$	

図 3: MinCaml の型つけ規則

MinCaml's typing rule

```

e ::=
  c
  op(x1, ..., xn)
  if x = y then e1 else e2
  if x ≤ y then e1 else e2
  let x = e1 in e2
  x
  let rec x y1 ... yn = e1 in e2
  x y1 ... yn
  (x1, ..., xn)
  let (x1, ..., xn) = y in e
  x.(y)
  x.(y) ← z

```

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

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

$\mathcal{K}(c)$	$= c$	
$\mathcal{K}(\text{not}(e))$	$= \mathcal{K}(\text{if } e \text{ then false else true})$	
$\mathcal{K}(e_1 = e_2)$	$= \mathcal{K}(\text{if } e_1 = e_2 \text{ then true else false})$	
$\mathcal{K}(e_1 \leq e_2)$	$= \mathcal{K}(\text{if } e_1 \leq e_2 \text{ then true else false})$	
$\mathcal{K}(op(e_1, \dots, e_n))$	$= \text{let } x_1 = \mathcal{K}(e_1) \text{ in } \dots \text{let } x_n = \mathcal{K}(e_n) \text{ in } op(x_1, \dots, x_n)$	$op$ が論理演算・比較以外の場合
$\mathcal{K}(\text{if not } e_1 \text{ then } e_2 \text{ else } e_3)$	$= \mathcal{K}(\text{if } e_1 \text{ then } e_3 \text{ else } e_2)$	Case 1: $op$ is NOT a logical operator nor comparator
$\mathcal{K}(\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in}$ $\text{if } x = y \text{ then } \mathcal{K}(e_3) \text{ else } \mathcal{K}(e_4)$	
$\mathcal{K}(\text{if } e_1 \leq e_2 \text{ then } e_3 \text{ else } e_4)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in}$ $\text{if } x \leq y \text{ then } \mathcal{K}(e_3) \text{ else } \mathcal{K}(e_4)$	
$\mathcal{K}(\text{if } e_1 \text{ then } e_2 \text{ else } e_3)$	$= \mathcal{K}(\text{if } e_1 = \text{false} \text{ then } e_3 \text{ else } e_2)$	$e_1$ が論理演算・比較以外の場合
$\mathcal{K}(\text{let } x = e_1 \text{ in } e_2)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in } \mathcal{K}(e_2)$	Case 2: $op$ is a logical operator or comparator
$\mathcal{K}(x)$	$= x$	
$\mathcal{K}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \text{let rec } x \ y_1 \ \dots \ y_n = \mathcal{K}(e_1) \text{ in } \mathcal{K}(e_2)$	
$\mathcal{K}(e \ e_1 \ \dots \ e_n)$	$= \text{let } x = \mathcal{K}(e) \text{ in let } y_1 = \mathcal{K}(e_1) \text{ in } \dots \text{let } y_n = \mathcal{K}(e_n) \text{ in}$ $x \ y_1 \ \dots \ y_n$	
$\mathcal{K}(e_1, \dots, e_n)$	$= \text{let } x_1 = \mathcal{K}(e_1) \text{ in } \dots \text{let } x_n = \mathcal{K}(e_n) \text{ in } (x_1, \dots, x_n)$	
$\mathcal{K}(\text{let } (x_1, \dots, x_n) = e_1 \text{ in } e_2)$	$= \text{let } y = \mathcal{K}(e_1) \text{ in let } (x_1, \dots, x_n) = y \text{ in } \mathcal{K}(e_2)$	
$\mathcal{K}(\text{Array.create } e_1 \ e_2)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in create\_array } x \ y$	
$\mathcal{K}(e_1.(e_2))$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in } x.(y)$	
$\mathcal{K}(e_1.(e_2) \leftarrow e_3)$	$= \text{let } x = \mathcal{K}(e_1) \text{ in let } y = \mathcal{K}(e_2) \text{ in let } z = \mathcal{K}(e_3) \text{ in}$ $x.(y) \leftarrow z$	

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。

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.

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

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

図 6:  $\alpha$  変換。  $\varepsilon$  は  $\alpha$  変換前の変数を受け取って、 $\alpha$  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。

#### $\alpha$ conversion:

$\varepsilon$  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.

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

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

図 7:  $\beta$  簡約。  $\varepsilon$  は  $\beta$  簡約前の変数を受け取って、 $\beta$  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。

#### $\beta$ -reduction

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

in case,  $A(e_1)$  is formed "let ... in  $e_1'$ " and  $e_1'$  is not a let-form.  
Here "let ... in" stands for 0 or more nesting of let bindings.

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

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

図 8: ネストした let の簡約

## Reduction of nested "let"

Example:

Suppose that:

$A(M) =$

let  $y = e_1'$  in

let  $z = e_2'$  in

$M_1$

$A(\text{let } x = M \text{ in } N)$  will be

let  $y = e_1'$  in

let  $z = e_2'$  in

let  $x = M_1$  in

$A(N)$

... (の) 場合 = in case ...

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

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

$\varepsilon$  is not defined for x

図 9: インライン展開。 $\varepsilon$  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  $\varepsilon$  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?).

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

$\mathcal{F}_\varepsilon(c)$	$=$	$c$	
$\mathcal{F}_\varepsilon(\text{op}(x_1, \dots, x_n))$	$=$	$c$	$\text{op}(\varepsilon(x_1), \dots, \varepsilon(x_n)) = c$ 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 $\varepsilon(y)$ が等しい定数の場合
$\mathcal{F}_\varepsilon(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$=$	$\mathcal{F}_\varepsilon(e_2)$	$\varepsilon(x)$ と $\varepsilon(y)$ が異なる定数の場合
$\mathcal{F}_\varepsilon(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$=$	$\text{if } x = y \text{ then } \mathcal{F}_\varepsilon(e_1) \text{ else } \mathcal{F}_\varepsilon(e_2)$	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 $\varepsilon(y)$ が定数で、 $\varepsilon(x) \leq \varepsilon(y)$ 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 $\varepsilon(y)$ が定数で、 $\varepsilon(x) > \varepsilon(y)$ 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
$\mathcal{F}_\varepsilon(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$=$	$\text{let } (x_1, \dots, x_n) = y \text{ in } \mathcal{F}_\varepsilon(e)$	
$\mathcal{F}_\varepsilon(x.(y))$	$=$	$x.(y)$	
$\mathcal{F}_\varepsilon(x.(y) \leftarrow z)$	$=$	$x.(y) \leftarrow z$	

図 10: 定数畳み込み。 $\varepsilon$  は変数を受け取って、定数を返す写像。

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

$\mathcal{E}(c)$	$= c$
$\mathcal{E}(op(x_1, \dots, x_n))$	$= op(x_1, \dots, x_n)$
$\mathcal{E}(\text{if } x = y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x = y \text{ then } \mathcal{E}(e_1) \text{ else } \mathcal{E}(e_2)$
$\mathcal{E}(\text{if } x \leq y \text{ then } e_1 \text{ else } e_2)$	$= \text{if } x \leq y \text{ then } \mathcal{E}(e_1) \text{ else } \mathcal{E}(e_2)$
$\mathcal{E}(\text{let } x = e_1 \text{ in } e_2)$	$= \mathcal{E}(e_2) \quad \text{effect}(\mathcal{E}(e_1)) = \text{false} \text{ かつ } x \notin FV(\mathcal{E}(e_2)) \text{ の場合}$
$\mathcal{E}(\text{let } x = e_1 \text{ in } e_2)$	$= \text{let } x = \mathcal{E}(e_1) \text{ in } \mathcal{E}(e_2) \quad \text{それ以外の場合}$
$\mathcal{E}(x)$	$= x$
$\mathcal{E}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \mathcal{E}(e_2) \quad x \notin FV(\mathcal{E}(e_2)) \text{ の場合}$
$\mathcal{E}(\text{let rec } x \ y_1 \ \dots \ y_n = e_1 \text{ in } e_2)$	$= \text{let rec } x \ y_1 \ \dots \ y_n = \mathcal{E}(e_1) \text{ in } \mathcal{E}(e_2) \quad \text{それ以外の場合}$
$\mathcal{E}(x \ y_1 \ \dots \ y_n)$	$= x \ y_1 \ \dots \ y_n$
$\mathcal{E}((x_1, \dots, x_n))$	$= (x_1, \dots, x_n)$
$\mathcal{E}(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \mathcal{E}(e) \quad \{x_1, \dots, x_n\} \cap FV(\mathcal{E}(e)) = \emptyset \text{ の場合}$
$\mathcal{E}(\text{let } (x_1, \dots, x_n) = y \text{ in } e)$	$= \text{let } (x_1, \dots, x_n) = y \text{ in } \mathcal{E}(e) \quad \text{それ以外の場合}$
$\mathcal{E}(x.(y))$	$= x.(y)$
$\mathcal{E}(x.(y) \leftarrow z)$	$= x.(y) \leftarrow z$

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

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

図 11: 不要定義削除 (1/2)



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

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

図 12: 不要定義削除 (2/2)

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

図 13: クロージャ変換後の構文

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

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

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

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

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変数。

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

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

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。

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

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

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
$x \leftarrow e; E$	代入
$e$	返値
$e ::=$	式
$c$	即値
$L_x$	ラベル
$op(x_1, \dots, x_n)$	算術演算
$\text{if } x = y \text{ then } E_1 \text{ else } E_2$	比較&分岐
$\text{if } x \leq y \text{ then } E_1 \text{ else } E_2$	比較&分岐
$x$	mov 命令
$\text{apply\_closure}(x, y_1, \dots, y_n)$	クロージャを用いた関数呼び出し
$\text{apply\_direct}(L_x, y_1, \dots, y_n)$	クロージャを用いない関数呼び出し
$x.(y)$	ロード
$x.(y) \leftarrow z$	ストア
<b>save</b> ( $x, y$ )	変数 $x$ の値をスタック位置 $y$ に退避する
<b>restore</b> ( $y$ )	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

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

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。 $\text{R}_{\text{hp}}$  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。

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

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。

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

図 20: 単純なレジスタ割り当て  $\mathcal{R}(P)$ ,  $\mathcal{R}(D)$  および  $\mathcal{R}_\varepsilon(E, z_{\text{dest}}, E_{\text{cont}})$ 。  $\varepsilon$  は変数からレジスタへの写像、  $z_{\text{dest}}$  は  $E$  の結果をセットする変数、  $E_{\text{cont}}$  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  $\varepsilon'$  の組。 [ファイル `regAlloc.notarget-nospill.ml` 参照]

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

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]



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

図 22: 変数  $x$  に割り当てるレジスタ  $r$  を選ぶときに使う targetting  $T_x(E, z_{\text{dest}})$  および  $T_x(e, z_{\text{dest}})$ 。  $E$  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、  $T_x(E'_{\text{cont}}, z_{\text{dest}}) = (c, s)$  として、できれば  $r \in s$  とする。[ファイル `regAlloc.target-nospill.ml` 参照]

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

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]

```

 $\mathcal{S} : \text{SparcAsm.prog} \rightarrow \text{string}$ 
 $\mathcal{S}(\{D_1, \dots, D_n\}, E) = \begin{array}{l} \text{.section ".text"} \\ \mathcal{S}(D_1) \\ \dots \\ \mathcal{S}(D_n) \\ \text{.global min\_caml\_start} \\ \text{min\_caml\_start:} \\ \text{save \%sp, -112, \%sp} \\ \mathcal{S}(E, \%g0) \\ \text{ret} \\ \text{restore} \end{array}$ 

 $\mathcal{S} : \text{SparcAsm.fundef} \rightarrow \text{string}$ 
 $\mathcal{S}(\text{L}_x(y_1, \dots, y_n) = E) = \begin{array}{l} x: \\ \mathcal{S}(E, \text{R}_0) \\ \text{retl} \\ \text{nop} \end{array}$ 

 $\mathcal{S} : \text{SparcAsm.t} \times \text{Id.t} \rightarrow \text{string}$ 
 $\mathcal{S}((x \leftarrow e; E), z_{\text{dest}}) = \begin{array}{l} \mathcal{S}(e, x) \\ \mathcal{S}(E, z_{\text{dest}}) \end{array}$ 
 $\mathcal{S}(e, z_{\text{dest}}) = \mathcal{S}(e, z_{\text{dest}})$ 

```

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

$\mathcal{S} : \text{SparcAsm.exp} \times \text{Id.t} \rightarrow \text{string}$	
$\mathcal{S}(c, z_{\text{dest}})$	<code>= set c, z<sub>dest</sub></code>
$\mathcal{S}(L_x, z_{\text{dest}})$	<code>= set L<sub>x</sub>, z<sub>dest</sub></code>
$\mathcal{S}(op(x_1, \dots, x_n), z_{\text{dest}})$	<code>= op x<sub>1</sub>, ..., x<sub>n</sub>, z<sub>dest</sub></code>
$\mathcal{S}(\text{if } x = y \text{ then } E_1 \text{ else } E_2, z_{\text{dest}})$	<code>= cmp x, y</code> <code>bne b<sub>1</sub></code> <code>nop</code> $\mathcal{S}(E_1, z_{\text{dest}})$ <code>b b<sub>2</sub></code> <code>nop</code> $b_1 :$ $\mathcal{S}(E_2, z_{\text{dest}})$ $b_2 :$
$\mathcal{S}(\text{if } x \leq y \text{ then } E_1 \text{ else } E_2, z_{\text{dest}})$	<code>= 同様</code>
$\mathcal{S}(x, z_{\text{dest}})$	<code>= mov x, z<sub>dest</sub></code>
$\mathcal{S}(\text{apply\_closure}(x, y_1, \dots, y_n), z_{\text{dest}})$	<code>= shuffle((x, y<sub>1</sub>, ..., y<sub>n</sub>), (R<sub>0</sub>, R<sub>1</sub>, ..., R<sub>n</sub>))</code> <code>st R<sub>ra</sub>, [R<sub>st</sub> + 4#ε]</code> <code>ld [R<sub>0</sub>], R<sub>n+1</sub></code> <code>call R<sub>n+1</sub></code> <code>add R<sub>st</sub>, 4(#ε + 1), R<sub>st</sub> ! delay slot</code> <code>sub R<sub>st</sub>, 4(#ε + 1), R<sub>st</sub></code> <code>ld [R<sub>st</sub> + 4#ε], R<sub>ra</sub></code> <code>mov R<sub>0</sub>, z<sub>dest</sub></code>
$\mathcal{S}(\text{apply\_direct}(L_x, y_1, \dots, y_n), z_{\text{dest}})$	<code>= shuffle((y<sub>1</sub>, ..., y<sub>n</sub>), (R<sub>1</sub>, ..., R<sub>n</sub>))</code> <code>st R<sub>ra</sub>, [R<sub>st</sub> + 4#ε]</code> <code>call x</code> <code>add R<sub>st</sub>, 4(#ε + 1), R<sub>st</sub> ! delay slot</code> <code>sub R<sub>st</sub>, 4(#ε + 1), R<sub>st</sub></code> <code>ld [R<sub>st</sub> + 4#ε], R<sub>ra</sub></code> <code>mov R<sub>0</sub>, z<sub>dest</sub></code>
$\mathcal{S}(x.(y), z_{\text{dest}})$	<code>= ld [x + y], z<sub>dest</sub></code>
$\mathcal{S}(x.(y) \leftarrow z, z_{\text{dest}})$	<code>= st z, [x + y]</code>
$\mathcal{S}(\text{save}(x, y), z_{\text{dest}})$	<code>= もし <math>y \notin \text{dom}(\varepsilon)</math> なら <math>\varepsilon</math> に <math>y \mapsto 4\#\varepsilon</math> を加えて</code> <code>st x, [R<sub>st</sub> + ε(y)]</code>
$\mathcal{S}(\text{restore}(y), z_{\text{dest}})$	<code>= ld [R<sub>st</sub> + ε(y)], z<sub>dest</sub></code>

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。#ε は ε の要素の個数。 $\text{shuffle}((x_1, \dots, x_n), (r_1, \dots, r_n))$  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。

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

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。

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

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

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

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

$\mathcal{S}_s(e, \text{tail}) = \mathcal{S}_s(e, \text{R}_0) = (s', S) \text{ として}$   
 $(\emptyset,$   
 $S$   
 $\text{retl}$   
 $\text{nop})$

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。