MapleLight

范伟杰

本文给出了 MapleLight——MapleIR 的一个子集的形式化的语法和语义，这个子集包含了 MapleIR 中比较核心的要素，包括面向对象、内存管理、指针、异常处理等。第一章给出了 MapleLight 的语法，第二章给出了 MapleLight 的语义。

1 语法

下面是 MapleLight 程序的形式化定义：

定义 1.

$$
\begin{array}{lll}
program ::= \text{Record } \{ & & \\
\quad \text{prog\_vars} & : \text{ list } (ident * var\_def); & \text{全局变量声明} \\
\quad \text{prog\_comps} & : \text{ list } (ident * composite); & \text{复合体定义} \\
\quad \text{prog\_funcs} & : \text{ list } (ident * function); & \text{函数定义} \\
\quad \text{prog\_main} & : ident; & \text{主函数名} \\
\} & &
\end{array}
$$

其中 $function$ 的定义如下，包括函数原型和函数体，其中函数体可缺省，当函数体缺省时即为函数声明，函数体存在时即为函数定义（$concrete\_function$）：

定义 2.

$$function ::= function\_prototype * \text{option } function\_body$$
$$concrete\_function ::= function\_prototype * function\_body$$

$function\_prototype$ ::= Record {

fun_attr : $func\_attr$; 函数属性声明

fun_params : list $(ident * type)$; 参数声明

fun_returns : list $type$; 返回值类型声明

}

$function\_body$ ::= Record {

fun_vars : list $(ident * var\_def)$; 局部变量声明

fun_pregs : list $(ident * type)$; 伪寄存器声明

fun_body : $statement$; 指令

}

$func\_attr$ ::= Record {

fa_access : $access\_modifier$;

fa_abstract : $bool$;

fa_final : $bool$;

fa_static : $bool$;

fa_virtual : $bool$;

fa_constructor : $bool$;

}

$access\_modifier$ ::= AM_friendly

| AM_public

| AM_protected

| AM_private

这里函数返回值的类型是 list $type$, 因为 MapleLight 的函数可以有多个返回值。

其他用到的定义会在本章后面几节中描述。

1.1 标识符

标识符用来标识变量、函数、类型、label、寄存器等，MapleLight 用正整数 (coq 中的 positive) 作为标识符 ($ident$)，并将全局变量和局部变量进行区分。而函数和类型只能是全局的，label 和伪寄存器只能是局部的，特殊寄存器只能是全局的，因此

它们可以直接用 *ident* 来表示。

$$
\begin{aligned}
ident &::= \text{positive} \\
var\_id &::= \text{Vglobal } (id : ident) \\
&\quad | \text{ Vlocal } (id : ident) \\
reg\_id &::= \text{Preg } (id : ident) \\
&\quad | \text{ Thrownval}
\end{aligned}
$$

1.2 类型

MapleLight 的类型包括 primitive type 和 derived type. 下面是 primitive type 的定义：

定义 3.

$$
\begin{aligned}
prim\_type &::= \text{Tvoid} \\
&\quad | \text{ Tint } (b : signedness)\ (n : intsize) \\
&\quad | \text{ Tbool} \\
&\quad | \text{ Tptr} \\
&\quad | \text{ Tref} \\
&\quad | \text{ Taddr } (n : addrsize) \\
&\quad | \text{ Tfloat } (n : floatsize) \\
&\quad | \text{ Tagg} \\
signedness &::= \text{Signed} \mid \text{Unsigned} \\
intsize &::= \text{I8} \mid \text{I16} \mid \text{I32} \mid \text{I64} \\
floatsize &::= \text{F32} \mid \text{F64} \\
addrsize &::= \text{A32} \mid \text{A64}
\end{aligned}
$$

derived types 由 primitive types 组合而来，包括 array, composite, function 和 pointer, 其中 composite 对应 primitive types 中的 Tagg, 而 array, pointer 和 function 对应 primitive types 中的 Taddr. composite 包括 struct, union, class 和 interface(见下一节的定义). 下面是 type 的形式化定义：

定义 4.

$$
\begin{aligned}
type \ ::= & \ \text{Tprim} \ (t : prim\_type) \\
& \mid \text{Tpointer} \ (t : type) \\
& \mid \text{Tarray} \ (t : type) \ (n : nat) \\
& \mid \text{Tfunction} \ (tl1 : \text{list} \ type) \ (tl2 : \text{list} \ type) \\
& \mid \text{Tcomposite} \ (id : ident)
\end{aligned}
$$

1.3 复合体定义

复合体 (*composite*) 包括 struct, union, class 和 interface, 复合体的定义只能是全局的。struct 和 union 的定义中只包含成员变量列表 ($membervars$), class 的定义中包含该类的父类 (可选)、该类实现的接口列表、该类的成员变量列表和该类的成员函数列表 ($memberfuncs$), interface 的定义中包含该接口的父接口列表、该接口的成员函数列表。程序中所有的复合体定义构成了一个 list, 即定义 1 中的 $prog\_comps : \text{list} \ (ident * composite)$, 其中 *composite* 的定义如下：

定义 5.

$$
\begin{aligned}
composite ::= \ & \text{Cstruct } (mv : membervars) \\
| \ & \text{Cunion } (mv : membervars) \\
| \ & \text{Cclass } (id : \text{option } ident)\ (li : \text{list } ident)\ (mv : membervars) \\
& \qquad (mf : memberfuncs)\ (ca : class\_attr) \\
| \ & \text{Cinterface } (li : \text{list } ident)\ (mf : memberfuncs)\ (ca : class\_attr)
\end{aligned}
$$

$$
membervars ::= \text{list } (ident * type * type\_attr * field\_attr)
$$

$$
memberfuncs ::= \text{list } (ident * ident * type * func\_attr)
$$

$$
\begin{aligned}
class\_attr ::= \ & \text{Record } \{ \\
& \text{ca\_access} : access\_modifier; \\
& \text{ca\_abstract} : bool; \\
& \text{ca\_final} : bool; \\
\}&
\end{aligned}
$$

$$
\begin{aligned}
field\_attr ::= \ & \text{Record } \{ \\
& \text{fa\_access} : access\_modifier; \\
\}&
\end{aligned}
$$

$$
\begin{aligned}
type\_attr ::= \ & \text{Record } \{ \\
& \text{ta\_alignment} : \text{option N} \\
\}&
\end{aligned}
$$

1.4 变量声明

在 MapleLight 程序中,所有全局变量声明构成一个 list,即定义 1 中的 $prog\_vars : list\ (ident * var\_def)$. 函数中的所有局部变量声明构成一个 list, 即定义 2 中的 $fun\_vars : \text{list } (ident * var\_def)$. 其中 var_def 的定义如下：

定义 6.

$$
\begin{aligned}
var\_def &::= type * storage\_class * type\_attr \\
storage\_calss &::= \text{SC\_default}
\end{aligned}
$$

1.5 伪寄存器声明

伪寄存器是指函数中永远不会被取地址的局部变量，伪寄存器的类型只能是 primitive type. 在 MapleLight 中，伪寄存器可以不被声明直接在函数体中使用。一个函数定义中的所有伪寄存器声明构成一个 list, 即定义 2 中的 $fun\_pregs : list\ (ident * prim\_type)$.

1.6 入口函数

每个 MapleLight 程序有一个入口函数（主函数），即定义 1 中的 $prog\_main : ident$.

1.7 一元算术运算符

MapleLight 中的一元算术运算包括求绝对值、按位取反、逻辑取反、求相反数、求倒数、sign extension、zero extension 和求平方根。其中 sign extension 和 zero extension 各需要附带一个参数用来表示扩展（或截断）到多少位。一元算术运算符的具体定义如下：

定义 7.

$$
\begin{array}{rll}
unary\_operation ::= & \text{O\_abs} & \text{求绝对值} \\
| & \text{O\_bnot} & \text{按位取反} \\
| & \text{O\_lnot} & \text{逻辑取反} \\
| & \text{O\_neg} & \text{求相反数} \\
| & \text{O\_recip} & \text{求倒数} \\
| & \text{O\_sext } (b : \text{N}) & \text{sign extension} \\
| & \text{O\_zext } (b : \text{N}) & \text{zero extension} \\
| & \text{O\_sqrt} & \text{求平方根}
\end{array}
$$

1.8 二元算术运算符

MapleLight 中的二元算术运算包括加减乘除、大于、小于、等于、不大于、不小于、不等于、比大小、按位与、按位或、按位异或、逻辑与、逻辑或、左移、逻辑右移、算术右移、取较大值、取较小值、求余。其中比大小运算包括三个版本，即 cmp, cmpg 和 cmpl。cmp 要求两个操作数都不是 NaN，当第一个操作数比第二个

大时返回 1，当第一个操作数比第二个操作数小时返回 −1，两个操作数相等时返回 0. cmpg 和 cmpl 在两个操作数都不是 NaN 时运算结果与 cmp 相同，如果两个操作数中有 NaN，则 cmpg 返回 1 而 cmpl 返回 −1。用于大小比较的运算（包括比大小、大于、小于等）都需要附带一个类型参数用来表示两个操作数的类型。二元算术运算符的具体定义如下：

定义 8.

binary_operation ::=	O_add	加法
	| O_ashr	算术右移
	| O_band	按位与
	| O_bior	按位或
	| O_bxor	按位异或
	| O_cmp ($pt : prim\_type$)	比大小
	| O_cmpg ($pt : prim\_type$)	比大小
	| O_cmpl ($pt : prim\_type$)	比大小
	| O_div	除法
	| O_eq ($pt : prim\_type$)	等于
	| O_ge ($pt : prim\_type$)	不小于
	| O_gt ($pt : prim\_type$)	大于
	| O_land	逻辑与
	| O_lior	逻辑或
	| O_le ($pt : prim\_type$)	不大于
	| O_lshr	逻辑右移
	| O_lt ($pt : prim\_type$)	小于
	| O_max	取较大值
	| O_min	取较小值
	| O_mul	乘法
	| O_ne ($pt : prim\_type$)	不等于
	| O_rem	求余
	| O_shl	左移
	| O_sub	减法

1.9 常量

MapleLight 中的常量既可以用于常量表达式，也是表达式求值的结果类型，复用了 CompCert 的常量定义，具体定义如下：

定义 9.

$$
\begin{aligned}
val ::= & \text{ Vundef} \\
| & \text{ Vint } (i : \text{int}) \\
| & \text{ Vlong } (i : \text{int64}) \\
| & \text{ Vfloat } (f : \text{float32}) \\
| & \text{ Vdouble } (f : \text{float64}) \\
| & \text{ Vptr } (b : block)\ (p : ptrofs) \\
block ::= & \text{ positive} \\
ptrofs ::= & \text{ int}
\end{aligned}
$$

其中 Vptr b p 是指针类型的常量，其中 b 是指针指向的地址在内存中的 block number, p 是该地址在这个 block 中的字节偏移量。(内存的定义见定义 14)

1.10 表达式

MapleLight 表达式的定义如下：

定义 10.

$expr$ ::= E_dread (t : $prim\_type$) (v : var_id) (fi : N)
| E_iread ($t1$: $prim\_type$) ($t2$: $type$) (fi : N) (e : $expr$)
| E_regread (t : $prim\_type$) (r : reg_id)
| E_addrof (t : $prim\_type$) (v : var_id) (fi : N)
| E_addroffunc (t : $prim\_type$) (f : $ident$)
| E_constval (t : $prim\_type$) (v : val)
| E_sizeoftype ($t1$: $prim\_type$) ($t2$: $type$)
| E_unary (uop : $unary\_operation$) (pt : $prim\_type$) (e : $expr$)
| E_iaddrof (pt : $prim\_type$) (t : $type$) (fi : N) (e : $expr$)
| E_ceil ($pt1$: $prim\_type$) ($pt2$: $prim\_type$) (e : $expr$)
| E_cvt ($t1$: $prim\_type$) ($t2$: $prim\_type$) (e : $expr$)
| E_floor (pt : $prim\_type$) (pt : $prim\_type$) (e : $expr$)
| E_retype ($t1$: $prim\_type$) ($t2$: $type$) (e : $expr$)
| E_trunc ($pt1$: $prim\_type$) ($pt2$: $prim\_type$) (e : $expr$)
| E_binary (bop : $binary\_operation$) (pt : $prim\_type$) ($e1$: $expr$) ($e2$: $expr$)
| E_cand (pt : $prim\_type$) ($e1$: $expr$) ($e2$: $expr$)
| E_cior (pt : $prim\_type$) ($e1$: $expr$) ($e2$: $expr$)
| E_select (pt : $prim\_type$) ($e1$: $expr$) ($e2$: $expr$) ($e3$: $expr$)
| E_array (b : bool) ($t1$: $type$) ($t2$: $type$) (el : list $expr$)

每个表达式的第一个参数指定了结果的类型。表达式的操作数也可以是表达式。如果构成操作数的子表达式的类型和结果类型不同，则操作数类型需由其他类型参数来指定（如一元运算符 sext 和 zext 携带的类型参数），若该表达式没有其他的类型参数，则该类型同时也是子表达式的类型。下面是每个表达式的语法和含义的描述。

1. dread: 直接读变量

   语法：E_dread (t : $prim\_type$) (v : var_id) (fi : N)

   含义：读取变量 v 的 fi 字段的值。若 v 不是 aggregate, 则 fi 必须为 0.

2. iread: 根据指针读变量

语法：E_iread ($t1 : prim\_type$) ($t2 : type$) (fi : N) ($e : expr$)

含义：先对 e 进行求值，然后读取求值结果所指向的变量的 fi 字段的值。$t2$ 指定了 e 的类型且必须为空指针类型。

3. regread: 读特殊寄存器或伪寄存器中的值

   语法：E_regread ($t : prim\_type$) ($r : reg\_id$)

   含义：特殊寄存器或伪寄存器 r 的值。

4. addrof: 对变量取地址

   语法：E_addrof ($t : prim\_type$) ($v : var\_id$) (fi : N)

   含义：变量 v 的 fi 字段的地址。

5. addroffunc: 对函数取地址

   语法：E_addroffunc ($t : prim\_type$) ($f : ident$)

   含义：函数 f 的地址。t 必须为 A32 或 A64.

6. constval: 常量

   语法：E_constval ($t : prim\_type$) ($v : val$)

   含义：常量 v 的值。

7. sizeoftype: 求类型大小

   语法：E_sizeoftype ($t1 : prim\_type$) ($t2 : type$)

   含义：类型 $t2$ 的大小（以字节为单位），$t1$ 必须是整数类型。

8. unary: 一元算术运算

   语法：E_unary ($uop : unary\_operation$) ($pt : prim\_type$) ($e : expr$)

   含义：先对 e 进行求值，然后根据运算符 uop 对 e 的求值结果进行对应的运算。

9. iaddrof: 根据指针取地址

   语法：E_iaddrof ($pt : prim\_type$) ($t : type$) (fi : N) ($e : expr$)

   含义：先对 e 进行求值，然后返回求值结果所指向的变量的 fi 字段的地址。t 指定了 e 的类型且必须位指针类型。

10. ceil: 向上取整

    语法：E_ceil $(pt1 : prim\_type)$ $(pt2 : prim\_type)$ $(e : expr)$

    含义：先对 e 进行求值，然后返回不小于求值结果的最小整数值。$pt1$ 是整数类型，$pt2$ 是浮点数类型。

11. cvt: 整数类型之间的转换

    语法：E_cvt $(t1 : prim\_type)$ $(t2 : prim\_type)$ $(e : expr)$

    含义：先对 e 进行求值，然后将求值结果从 $t2$ 类型转换为 $t1$ 类型。$t1$ 和 $t2$ 必须都是整数类型。

12. floor: 向下取整

    语法：E_floor $(pt1 : prim\_type)$ $(pt2 : prim\_type)$ $(e : expr)$

    含义：先对 e 进行求值，然后返回不大于求值结果的最小整数值。$pt1$ 是整数类型，$pt2$ 是浮点数类型。

13. retype: 类型转换

    语法：E_retype $(t1 : prim\_type)$ $(t2 : type)$ $(e : expr)$

    含义：先对 e 进行求值，然后将求值结果从 $t2$ 类型转换的 $t1$ 类型。

14. trunc: 向零取整

    语法：E_trunc $(pt1 : prim\_type)$ $(pt2 : prim\_type)$ $(e : expr)$

    含义：先对 e 进行求值，当求值结果大于等于 0 时向下取整，否则向上取整。

15. binary: 二元算术运算

    语法：E_binary $(bop : binary\_operation)$ $(pt : prim\_type)$ $(e1 : expr)$ $(e2 : expr)$

    含义：先对 $e1$ 和 $e2$ 进行求值，然后根据运算符 bop 对 $e1$ 和 $e2$ 的求值结果进行对应的运算。

16. cand: 短路逻辑与

    语法：E_cand $(pt : prim\_type)$ $(e1 : expr)$ $(e2 : expr)$

    含义：先对 $e1$ 进行求值，若 $e1$ 的求值结果为 0 则直接返回 0，否则对 $e2$ 进行求值，然后对 $e1$ 和 $e2$ 的求值结果进行逻辑与运算。pt 必须是整数类型。

17. cior: 短路逻辑或

    语法：E_cior $(pt : prim\_type)$ $(e1 : expr)$ $(e2 : expr)$

    含义：先对 $e1$ 进行求值，若 $e1$ 的求值结果非 0 则直接返回 1，否则对 $e2$ 进行求值，然后对 $e1$ 和 $e2$ 的求值结果进行逻辑与运算。pt 必须是整数类型。

18. select: 选择

    语法：E_select $(pt : prim\_type)$ $(e1 : expr)$ $(e2 : expr)$ $(e3 : expr)$

    含义：先对 $e1$ 进行求值，若 $e1$ 的求值结果非 0 则对 $e2$ 进行求值并返回结果，否则对 $e3$ 进行求值并返回结果。

19. array: 从多维数组中根据下标取值

    语法：E_array $(b : \text{bool})$ $(t1 : type)$ $(t2 : type)$ $(el : \text{list } expr)$

    含义：先对 el 进行求值，求出的结果列表中的第一个元素是数组在内存中的首地址，根据 el 剩余的元素进行从高维到低维索引，返回被索引元素的地址。b 为 true 时进行边界检查，否则不进行。$t2$ 给出了指向数组的指针的类型。

1.11 拓展表达式

MapleLight 的表达式不涉及内存分配和回收，因此不会表达式求职过程中不会对内存产生影响。而拓展表达式在求值过程中会进行内存分配和回收操作。下面是拓展表达式的形式化定义：

定义 11.

$$
\begin{aligned}
ext\_expr ::= &\ \text{EE\_pure } (e : expr) \\
&| \ \text{EE\_malloc } (pt : prim\_type)\ (e : expr) \\
&| \ \text{EE\_gcmalloc } (pt : prim\_type)\ (t : type) \\
&| \ \text{EE\_gcmallocjarray } (pt : prim\_type)\ (t : type)\ (e : expr) \\
&| \ \text{EE\_gcpermalloc } (pt : prim\_type)\ (t : type)
\end{aligned}
$$

1. pure: 普通表达式

   语法：EE_pure $(e : expr)$

   含义：对表达式 e 进行求值。

2. malloc: 堆内存分配

   语法：EE_malloc $(pt : prim\_type)$ $(e : expr)$

   含义：先对 e 进行求值，将求值结果作为分配内存的大小（以字节为单位），在堆空间中分配内存，返回新分配的内存的起始地址。malloc 分配的内存直到执行 free 操作才会被释放。

3. gcmalloc: 为对象分配内存 (可回收)

   语法：EE_gcmalloc $(pt : prim\_type)$ $(t : type)$

   含义：在堆空间中为 $t2$ 类型的对象分配内存，大小由 $t2$ 决定，返回新分配的内存的起始地址。gcmalloc 分配的内存不能被 free 指令释放，只能由 gc 释放。

4. gcmallocjarray: 为数组对象分配内存

   语法：EE_gcmallocjarray $(pt : prim\_type)$ $(t : type)(e : expr)$

   含义：先对 e 进行求值，然后在堆空间中为 $t2$ 类型的数组对象分配内存，大小由 $t2$ 和 e 的求值结果决定，返回新分配的内存的起始地址。gcmallocjarray 分配的内存不能被 free 指令释放，只能由 gc 释放。

5. gcpermalloc: 为对象分配内存 (不可回收)

   语法：EE_gcpermalloc $(pt : prim\_type)$ $(t : type)$

   含义：在堆空间中为 $t2$ 类型的对象分配内存，大小由 $t2$ 决定，返回新分配的内存的起始地址。gcpermalloc 分配的内存不能被 free 指令释放，也不能被 gc 释放。

1.12 指令

MapleIR 中的指令可以分为写内存指令、控制流指令、函数调用指令、内存分配回收指令、异常处理指令和其他特殊指令。本章给出了 MapleIR 指令的一个子集，共包含 34 条指令，我对该子集的形式化定义如下：

定义 12.

$statement$ ::= S_skip

| S_dassign (v : var_id) (fi : N) (ee : ext_expr)

| S_iassign (t : $type$) (fi : N) ($e1$: $expr$) ($e2$: ext_expr)

| S_regassign (t : $prim\_type$) (r : reg_id) (e : ext_expr)

| S_seq ($s1$: $statement$) ($s2$: $statement$)

| S_label (lbl : $ident$) (s : $statement$)

| S_if (e : $expr$) ($s1$: $statement$) ($s2$: $statement$)

| S_while (e : $expr$) (s : $statement$)

| S_goto (lbl : $ident$)

| S_return (el : list $expr$)

| S_switch (e : $expr$) (lbl : $ident$) (l : list (Z $*$ $ident$))

| S_callassigned (f : $ident$) (el : list $expr$) (l : list (var_id $*$ N))

| S_icallassigned (e : $expr$)(el : list $expr$) (l : list (var_id $*$ N))

| S_virtualcallassigned (c : $ident$) (f : $ident$) (e : $expr$)
(el : list $expr$) (l : list (var_id $*$ N))

| S_interfacecallassigned (c : $ident$) (f : $ident$) (e : $expr$)
(el : list $expr$) (l : list (var_id $*$ N))

| S_javatry (ll : list $ident$) (s : $statement$)

| S_throw (e : $expr$)

| S_javacatch (lbl : $ident$) (tl : list $type$)

| S_free (e : $expr$)

| S_incref (e : $expr$)

| S_decref (e : $expr$)

| S_eval (e : $expr$)

下面是每条指令的语法和含义的描述。

1. skip: 无操作

   语法：S_skip

   含义：无任何操作。

2. dassign: 直接赋值

   语法：S_dassign $(v : var\_id)$ $(fi : \mathrm{N})$ $(ee : ext\_expr)$

   含义：计算 e 的值并赋值给 v 的 fi 字段。如果 v 不是 aggregate, 则 fi 必须为 0。

3. iassign: 通过指针赋值

   语法：S_iassign $(t : type)$ $(fi : \mathrm{N})$ $(e1 : expr)$ $(e2 : ext\_expr)$

   含义：计算 $e2$ 的值并赋值给 $e1$ 所指向的变量的 fi 字段。t 指定了 $e1$ 的类型且必须为指针类型。

4. regassign: 对特殊寄存器或伪寄存器赋值

   语法：S_regassign $(t : prim\_type)$ $(r : reg\_id)$ $(e : ext\_expr)$

   含义：计算 e 的值并赋值给特殊寄存器或伪寄存器 r. t 指定了 r 的类型。

5. seq: 顺序结构

   语法：S_seq $(s1 : statement)$ $(s2 : statement)$

   含义：先执行 $s1$, 再执行 $s2$.

6. label

   语法：S_label $(lbl : ident)$ $(s : statement)$

   含义：执行 s. lbl 是 s 的 label, 可供跳转指令使用。

7. if: 条件分支

   语法：S_if $(e : expr)$ $(s1 : statement)$ $(s2 : statement)$

   含义：计算 e 的值，如果非零，则执行 $s1$，否则执行 $s2$.

8. while: 循环

   语法：S_while $(e : expr)$ $(s : statement)$

   含义：先计算 e 的值，如果非零，则执行 s, 然后进入下一轮循环，否则跳出循环。

9. goto: 无条件跳转

   语法：S_goto $(l : ident)$

   含义：跳转到 lbl 处开始执行。

10. return: 函数返回

    语法：S_return (el : list $expr$)

    含义：返回 el.

11. switch: 多重条件分支

    语法：S_switch (e : $expr$) (lbl : $ident$) (l : list (Z $*$ $ident$))

    含义：计算 e 的值并与 l 中的整数值进行匹配，跳转到第一个匹配成功的 label, 如果没有匹配成功的，则跳转到默认的 lbl.

12. callassigned: 普通函数调用

    语法：S_callassigned (f : $ident$) (el : list $expr$) (l : list (var_id $*$ N))

    含义：调用函数 f 并传递参数 el, 将返回的结果依次赋值给 l 中的变量的对应字段。

13. icallassigned: 通过指针调用函数

    语法：S_icallassigned (e : $expr$)(el : list $expr$) (l : list (var_id $*$ N))

    含义：计算 e 的值，调用其指向的函数并传递参数 el, 将返回的结果依次赋值给 l 中的变量的对应字段。

14. virtualcallassigned: 成员函数调用

    语法：S_virtualcallassigned (c : $ident$) (f : $ident$) (e : $expr$) (el : list $expr$) (l : list (var_id $*$ N))

    含义：计算 e 的值，在其指向的对象所在类的层次结构中搜索需要调用的虚函数 f, 若没有则不断向上搜索该类的父类，直到找到为止。调用查找到的函数并传递参数 e 和 el, 将返回的结果依次赋值给 l 中的变量的对应字段。c 是类名，e 所指向的对象所在类必须是类 c 的子类。

15. interfacecallassigned: 接口函数调用

    语法：S_interfacecallassigned (c : $ident$) (f : $ident$) (e : $expr$) (el : list $expr$) (l : list (var_id $*$ N))

    含义：计算 e 的值，在其指向的对象所在类的层次结构中搜索需要调用的虚函数 f, 若没有则不断向上搜索该类的父类，直到找到为止。调用查找到的函数并传递参数 e 和 el, 将返回的结果依次赋值给 l 中的变量的对应字段。c 是类名，e 所指向的对象所在类必须实现了接口 c。

16. javatry

    语法：S_javatry (ll : list $ident$) (s : $statement$)

    含义：执行 s, 在执行过程中如果发生异常，则根据异常类型搜索匹配的 exception handler. 一个 javatry block 可以有多个 exception handler, 每个 exception handler 有一个 label, ll 是这些 label 的列表。

17. throw

    语法：S_throw (e : $expr$)

    含义：抛出异常并计算 e 的值赋值给特殊寄存器 Thrownval。

18. javacatch

    语法：S_javacatch (lbl : $ident$) (tl : list $type$)

    含义：标志着一个 exception handler 的开始。lbl 是 exception handler 的 label, 一个 exception handler 可以捕捉多种类型的异常，tl 是捕捉的异常的类型的列表。

19. free: 内存释放

    语法：S_free (e : $expr$)

    含义：计算 e 的值并释放其所指向的内存空间。

20. incref: 增加对象引用记数

    语法：S_incref (e : $expr$)

    含义：计算 e 的值，并将其指向的对象的引用计数值增加 1，e 的类型必须是 ref。

21. decref: 减少对象引用记数

    语法：S_decref (e : $expr$)

    含义：计算 e 的值，并将其指向的对象的引用计数值减少 1，e 的类型必须是 ref。

22. eval: 求值

    语法：S_eval (e : $expr$)

    含义：计算 e 的值但不保留计算的结果。如果 e 中含有对 volatile 变量的引用, 则该指令不能被优化。

2 语义

2.1 MapleLight 程序运行的 configuration

定义 13. configuration

程序运行的 configuration 由两部分组成，一个是程序运行的全局环境 ($genv$), 它在初始化时构建，在程序运行过程中不会发生变化。另一个是程序的状态 ($state$), 包括四种状态：正常运行状态 (NormalState), 异常状态 (ExceptionState), 调用函数 (CallState), 函数返回 (ReturnState)。在这四种状态下都要记录的信息包括对象环境 ($oenv$), 内存状态 (mem) 以及控制流信息 ($cont$). 另外，在 NormalState 状态下需要记录函数运行的局部环境 ($lenv$), 当前运行的函数 ($concrete\_function$) 和剩下的代码 ($statement$)。在 ExceptionState 状态下需要记录函数运行的局部环境 ($lenv$) 和当前运行的函数 ($concrete\_function$)。在 CallState 状态下需要记录被调函数 ($function$) 和参数值列表 (list val)。在 ReturnState 状态下需要记录返回值列表 (list val)。

$$
\begin{aligned}
configuration ::= {} & genv * state \\
state ::= {} & \text{NormalState } (f : concrete\_function)\ (s : statement)\ (k : cont) \\
& \qquad (le : lenv)\ (oe : oenv)\ (m : mem) \\
& \mid \text{ExceptionState } (f : concrete\_function)\ (k : cont) \\
& \qquad (oe : oenv)\ (le : lenv)\ (m : mem) \\
& \mid \text{CallState } (f : function)\ (args : \text{list } val)\ (k : cont) \\
& \qquad (oe : oenv)\ (m : mem) \\
& \mid \text{ReturnState } (res : \text{list } val)\ (k : cont)\ (oe : oenv)\ (m : mem)
\end{aligned}
$$

下面给出内存状态，全局环境，局部环境，对象环境和控制流信息的定义。

定义 14. 内存状态

复用了 CompCert 对内存状态的定义，为每个变量/函数/对象分配一个 block, 每个 block 含有若干个字节，通过 block number 和字节偏移量可以从内存中读取特定的成员变量或数组中的元素，任意两个 block 之间是不相交的。简化的形式化定义如下：

$$mem ::= block \to \text{option } (ptrofs \to \text{option byte})$$

实际上 memory 的定义比这复杂得多，这里进行了简化。

定义 15. 全局环境

程序运行所需的全局环境是在初始化阶段生成的，在程序运行期间不会发生变化，其中 genv_cenv 将复合体的 id 映射到复合体定义，genv_vars 将全局变量的 id 映射到其在内存中的 block number 以及变量定义, genv_funcs 将函数的 id 映射到其在内存中的 block number, genv_fundefs 将内存中的 block number 映射到以及函数定义。

$$
\begin{aligned}
&\mathit{composite\_env} ::= \mathit{ident} \to \text{option } \mathit{composite} \\
&\mathit{genv} ::= \{ \\
&\quad \text{genv\_cenv} \quad : \mathit{composite\_env}; \\
&\quad \text{genv\_vars} \quad : \mathit{ident} \to \text{option } (\mathit{block} * \mathit{var\_def}); \\
&\quad \text{genv\_funcs} \quad : \mathit{ident} \to \text{option } \mathit{block}; \\
&\quad \text{genv\_fundefs} \ : \mathit{block} \to \text{option } \mathit{function}; \\
&\}
\end{aligned}
$$

定义 16. 局部环境

函数运行所需的局部环境是在调用函数时生成的，在函数运行期间可能会发生变化，其中 lenv_vars 将局部变量的 id 映射到其在内存中的 block number 以及变量定义。lenv_pregs 将伪寄存器的 id 映射到其值和类型，lenv_thrownval 记录了特殊寄存器 Thrownval 的值和类型。

$$
\begin{aligned}
&\mathit{lenv} ::= \{ \\
&\quad \text{lenv\_vars} \quad : \mathit{ident} \to \mathit{option}\ (\mathit{block} * \mathit{var\_def}); \\
&\quad \text{lenv\_pregs} \quad : \mathit{ident} \to (\mathit{val} * \mathit{prim\_type}); \\
&\quad \text{lenv\_thrownval} : \text{option } (\mathit{val} * \mathit{prim\_type}); \\
&\}
\end{aligned}
$$

定义 17. 对象环境

对象环境将程序运行期间创建的每个对象的在内存中的 block number 映射到该对象的类型，该对象被引用的次数以及该对象是否可以被 GC 释放 (由 gcpermalloc 创建的对象不可被 GC 释放)。

$$\mathit{oenv} ::= \mathit{ident} \to \mathit{option}\ (\mathit{type} * \text{nat} * \mathit{bool})$$

定义 18. 控制流信息

程序运行的控制流信息本质上是一个栈，当控制流发生变化时将当前控制流压入栈中，跳转到其他代码运行，待其他代码运行结束后 (当前剩余代码为 skip 时) 从栈中弹出控制流生成新的代码继续运行。栈底元素 Kstop 标志着程序运行的结束，程序开始时将 Kstop 压入栈中，当 Kstop 从栈中弹出时意味着程序运行结束。

$cont$ ::= Kstop

| Kseq (s : $statement$) (k : $cont$)

| Kwhile (e : $expr$) (s : $statement$) (k : $cont$)

| Kjavatry (ll : list $ident$) (k : $cont$)

| Kcall (n : list ($var\_id * \mathrm{N}$)) (f : $concrete\_function$) (le : $lenv$) (k : $cont$)

2.2 表达式求值的语义

本章给出了 MapleLight 表达式求值的语义。表达式求值过程中会用到全局环境、局部环境和对象环境但不会改变它们，因此它们是表达式求值过程中的不变量，分别用 ge, le 和 m 表示。

下面是表达式求值的语义中需要用到的辅助函数及其作用，这些辅助函数的定义比较繁琐，并且复用了大量的 CompCert 的代码，故在这里省略不写。

1. find_var (vid : var_id) : $option$ ($block * var\_def$)

   作用：查找全局或局部变量 vid 所处的内存块和变量定义。

2. deref_loc (t : $type$) (m : mem) (loc : $block$) (ofs : $ptrofs$) : option val

   作用：根据类型 t 在内存 m 中查找位于某个内存块 loc 和偏移量 ofs 处的值。

3. fieldoffset (ce : $composite\_env$) (t : $type$) (fi : N) : option ($type * \mathrm{Z}$)

   作用：计算类型 t 中的 fi 字段的类型和偏移量。

4. sizeof (ce : $composite\_env$) (t : $type$) (ta : $type\_attr$) : Z

   作用：根据类型 t 和类型属性 ta 计算该类型的变量在内存中所占空间的大小。

5. default_type_attr : $type\_attr$

   作用：默认的类型属性。

6. eval_array (b : $bool$) (t : $type$) (vl : list val) (ofs : $ptrofs$) : option $ptrofs$

作用：根据多维数组的类型 t 和起始地址在内存块中的偏移量 ofs 以及不同维度的下标值 vl 计算被索引元素在内存块中的偏移量，b 为 true 时进行下标越界检查，b 为 false 时不进行下标越界检查。

7. sem_cast (v : val) ($t1$: $type$) ($t2$: $type$) (m : mem) (ce : $composite\_env$) : option val)

   作用：将常量 v 从 $t1$ 类型转换到 $t2$ 类型。

8. find_reg (rid : reg_id) : option ($val * prim\_type$)

   作用：查找特殊寄存器或伪寄存器 rid 的值和类型。

9. is_pointer_prim_type (pt : $prim\_type$) : bool

   作用：判断类型 pt 是不是指针类型。

10. is_int_prim_type (pt : $prim\_type$) : bool

    作用：判断类型 pt 是不是整数类型。

11. sem_ceil (v : val) ($pt1$: $prim\_type$) ($pt2$: $prim\_type$) : option val

    作用：对类型为 $pt1$ 的值 v 向上取整并将结果转换到 $pt2$ 类型。

12. sem_floor (v : val) ($pt1$: $prim\_type$) ($pt2$: $prim\_type$) : option val

    作用：对类型为 $pt1$ 的值 v 向下取整并将结果转换到 $pt2$ 类型。

13. sem_trunc (v : val) ($pt1$: $prim\_type$) ($pt2$: $prim\_type$) : option val

    作用：对类型为 $pt1$ 的值 v 向零取整并将结果转换到 $pt2$ 类型。

14. sem_unary (uop : $unary\_operation$) (v : val) ($t1$: $type$) ($t2$: $type$) (m : mem) (ce : $composite\_env$) : option val

    作用：根据一元算术运算符 uop 对类型为 $t1$ 的值 v 进行相应的运算并将结果转换到 $t2$ 类型。

15. sem_binary (bop : $binary\_operation$) ($v1$: val) ($t1$: $type$) ($v2$: val) ($t2$: $type$) ($t3$: $type$) (m : mem) (ce : $composite\_env$) : option val

    作用：根据二元算术运算符 bop 对类型为 $t1$ 的值 $v1$ 和类型为 $t2$ 的值 $v2$ 进行相应的运算并将结果转换到 $t3$ 类型。

16. bool_val $(v : val)$ $(t : type)$ $(m : mem)$: option bool

    作用：判断类型为 t 的值 v 是否为 0，若为 0 则返回 false 否则返回 true.

17. typeof $(e : expr)$: $type$

    作用：获取表达式 e 的结果类型。

18. prim_type_of $(e : expr)$: $prim\_type$

    作用：获取表达式 e 的结果类型。

19. val_to_ptrofs $(v : val)$: option $ptrofs$

    作用：将值 v 转换为字节偏移量。

下面是表达式求值的语义，用 $e \to v$ 的形式表示表达式 e 的求值结果为 v。

1. constval: 常量

$$\frac{\text{sem\_cast (Vint } i)(\text{Tprim (PTint I32 Unsigned)) (Tprim } pt)\ m\ (genv\_cenv\ ge) = \text{Some } v'}{\text{E\_constval } pt\ (\text{Vint } i) \to v'}$$

$$\frac{\text{sem\_cast (Vlong } i)(\text{Tprim (PTint I64 Unsigned)) (Tprim } pt)\ m\ (genv\_cenv\ ge) = \text{Some } v'}{\text{E\_constval } pt\ (\text{Vlong } i) \to v'}$$

$$\frac{\text{sem\_cast (Vsingle } f)(\text{Tprim (PTfloat F32 Unsigned)) (Tprim } pt)\ m\ (genv\_cenv\ ge) = \text{Some } v'}{\text{E\_constval } pt\ (\text{Vsingle } f) \to v'}$$

$$\frac{\text{sem\_cast (Vfloat } f)(\text{Tprim (PTfloat F64 Unsigned)) (Tprim } pt)\ m\ (\text{genv\_cenv } ge) = \text{Some } v'}{\text{E\_constval } pt\ (\text{Vfloat } f) \to v'}$$

2. dread: 直接读变量

$$\frac{\begin{array}{c}\text{find\_var } vid = \text{Some } (loc, (t, va)) \\ \text{fieldoffset (genv\_comps } ge)\ t\ fi = \text{Some } (t', ofs) \\ \text{deref\_loc } t'\ m\ loc\ (\text{Ptrofs.repr } ofs) = \text{Some } v \\ \text{sem\_cast } v\ t'\ (\text{Tprim } pt)\ m\ (\text{genv\_cenv } ge) = \text{Some } v'\end{array}}{\text{E\_dread } pt\ vid\ fi \to v'}$$

3. iread: 根据指针读变量

$$\frac{\begin{array}{c}e \to \text{Vptr } loc\ ofs \\ t = \text{Tpointer } t' \\ \text{fieldoffset (genv\_comps } ge)\ t'\ fi = \text{Some } (t'', delta) \\ \text{deref\_loc } t''\ m\ loc\ (\text{Ptrofs.add } ofs\ (\text{Ptrofs.repr } delta)) = \text{Some } v \\ \text{sem\_cast } v\ t''\ (\text{Tprim } pt)\ m\ (\text{genv\_cenv } ge) = \text{Some } v'\end{array}}{\text{E\_iread } pt\ t\ fi\ e \to v'}$$

4. regread: 读寄存器的值

$$\frac{\text{find\_reg } rid = \text{Some } (pt, v) \quad \text{sem\_cast } v\ (\text{Tprim } pt)\ (\text{Tprim } pt')\ m\ (\text{genv\_cenv } ge) = \text{Some } v'}{\text{E\_regread } pt\ rid \to v'}$$

5. addrof: 对变量取地址

$$\frac{\text{is\_pointer\_prim\_type } pt = \text{true} \quad \text{find\_var } vid = \text{Some } (loc, (t, va)) \quad \text{fieldoffset } (\text{genv\_comps } ge)\ t\ fi = \text{Some } (t', ofs)}{\text{E\_addrof } pt\ vid\ fi \to \text{Vptr } loc\ (\text{Ptrofs.repr } ofs)}$$

6. iaddrof: 根据指针取地址

$$\frac{\text{is\_pointer\_prim\_type } pt = \text{true} \quad e \to \text{Vptr } b\ ofs \quad t = \text{Tpointer } t' \quad \text{fieldoffset } (\text{genv\_cenv } ge)\ t'\ fi = \text{Some } (t'', delta)}{\text{E\_iaddrof } pt\ t\ fi\ e \to\ \text{Vptr } b\ (\text{Ptrofs.add } ofs\ (\text{Ptrofs.repr } delta))}$$

7. addroffunc: 对函数取地址

$$\frac{\text{is\_pointer\_prim\_type } pt = \text{true} \quad (\text{genv\_funcs } ge)\ id = \text{Some } loc}{\text{E\_addroffunc } pt\ id \to \text{Vptr } loc\ (\text{Ptrofs.repr } 0)}$$

8. sizeoftype: 求类型大小

$$\frac{\text{sizeof } (\text{genv\_comps } ge)\ t\ \text{default\_type\_attr} = z}{\text{E\_sizeoftype } pt\ t \to \text{Vptrofs } (\text{Ptrofs.repr } z)}$$

9. retype: 类型转换

$$\frac{e \to v \quad \text{sem\_cast } v\ t\ (\text{Tprim } pt)\ m\ (\text{genv\_cenv } ge) = \text{Some } v'}{\text{E\_retype } pt\ t\ e \to v'}$$

10. cvt: 整数类型之间的转换

$$\frac{e \to v \qquad \text{sem\_cast } v\ (\text{Tprim } pt)\ (\text{Tprim } pt')\ m\ (\text{genv\_cenv } ge) = \text{Some } v'}{\text{E\_cvt } pt'\ pt\ e \to v'}$$

11. ceil: 向上取整

$$\frac{e \to v \qquad \text{sem\_ceil } v\ (\text{Tprim } pt)\ (\text{Tprim } pt')\ m\ (\text{genv\_cenv } ge) = \text{Some } v'}{\text{E\_ceil } pt'\ pt\ e \to v'}$$

12. floor: 向下取整

$$\frac{e \to v \qquad \text{sem\_floor } v\ (\text{Tprim } pt)\ (\text{Tprim } pt')\ m\ (\text{genv\_cenv } ge) = \text{Some } v'}{\text{E\_floor } pt'\ pt\ e \to v'}$$

13. trunc: 向零取整

$$\frac{e \to v \qquad \text{sem\_trunc } v\ (\text{Tprim } pt)\ (\text{Tprim } pt')\ m\ (\text{genv\_cenv } ge) = \text{Some } v'}{\text{E\_trunc } pt'\ pt\ e \to v'}$$

14. unary: 一元算术运算

$$\frac{e \to v \qquad \text{sem\_unary } uop\ v\ (\text{typeof } e)\ (\text{Tprim } pt)\ m\ (\text{genv\_cenv } ge) = \text{Some } v'}{\text{E\_unary } uop\ pt\ e \to v'}$$

15. binary 二元算术运算

$$\frac{e1 \to v1 \qquad e2 \to v2 \qquad \text{sem\_binary } bop\ v1\ (\text{typeof } e1)\ v2\ (\text{typeof } e2)\ (\text{Tprim } pt)\ m\ (\text{genv\_cenv } ge) = \text{Some } v}{\text{E\_binary } bop\ pt\ e1\ e2 \to v}$$

16. cand: 短路逻辑与

$$\frac{\text{is\_int\_prim\_type } pt = \text{true} \quad e1 \to v1 \quad \text{bool\_val } v1 \text{ (typeof } e1) \ m = \text{Some true} \quad e2 \to v2 \quad \text{bool\_val } v2 \text{ (typeof } e2) \ m = \text{Some } b}{\text{E\_cand } pt \ e1 \ e2 \to \text{Val.of\_bool } b}$$

$$\frac{\text{is\_int\_prim\_type } pt = \text{true} \quad e1 \to v1 \quad \text{bool\_val } v1 \text{ (typeof } e1) \ m = \text{Some false}}{\text{E\_cand } pt \ e1 \ e2 \to \text{Val.of\_bool false}}$$

17. cior: 短路逻辑或

$$\frac{\text{is\_int\_prim\_type } pt = \text{true} \quad e1 \to v1 \quad \text{bool\_val } v1 \text{ (typeof } e1) \ m = \text{Some true}}{\text{E\_cior } pt \ e1 \ e2 \to \text{Val.of\_bool true}}$$

$$\frac{\text{is\_int\_prim\_type } pt = \text{true} \quad e1 \to v1 \quad \text{bool\_val } v1 \text{ (typeof } e1) \ m = \text{Some false} \quad e2 \to v2 \quad \text{bool\_val } v2 \text{ (typeof } e2) \ m = \text{Some } b}{\text{E\_cior } pt \ e1 \ e2 \to \text{Val.of\_bool } b}$$

18. select: 选择

$$\frac{\text{prim\_type\_of } e2 = pt \wedge \text{prim\_type\_of } e2 = pt \quad e1 \to v1 \quad \text{bool\_val } v1 \text{ (typeof } e1) \ m = \text{Some true} \quad e2 \to v2}{\text{E\_select } pt \ e1 \ e2 \ e3 \to v2}$$

$$\frac{\text{prim\_type\_of } e2 = pt \wedge \text{prim\_type\_of } e2 = pt \quad \text{bool\_val } v1 \text{ (typeof } e1) \ m = \text{Some false} \quad e3 \to v3}{\text{E\_select } pt \ e1 \ e2 \ e3 \to v3}$$

19. array: 从多维数组中根据下标取值

$$\frac{\begin{array}{c} e \to \text{Vptr } loc\ ofs \\ el \to_* vl \\ \text{eval\_array } b\ t\ ofs\ vl = \text{Some } delta \end{array}}{\text{E\_array } b\ pt\ t\ (e :: el) \to \text{Vptr } loc\ (\text{Ptrofs.add } ofs\ delta)}$$

20. 表达式列表求值

$$\frac{}{\text{nil} \to_* \text{nil}} \qquad \frac{e \to v \quad el \to_* vl}{e :: el \to_* v :: vl}$$

2.3 拓展表达式的语义

拓展表达式的求值过程中不会改变全局环境和局部环境，因此 ge 和 le 仍然是不变量，但会改变内存状态和对象环境，用 $(ee, m, oe) \to (v, m', oe')$ 的形式来表示拓展表达式 ee 在初始的内存状态 m 和对象环境 oe 下求值结果为 v，同时将内存状态改变为 m', 对象环境改变为 oe'.

1. pure: 普通表达式

$$\frac{e \xrightarrow{m} v}{(\text{EE\_pure } e, oe, m) \to (v, oe, m)}$$

2. malloc: 堆内存分配

$$\frac{\begin{array}{c} e \xrightarrow{m} v \\ \text{val\_to\_ptrofs } v = \text{Some } ofs \\ \text{Mem.alloc } m\ (-\text{size\_chunk Mptr})\ (\text{Ptrofs.unsigned } ofs) = (m', b) \\ \text{Mem.store Mptr } m'\ b\ (-\text{size\_chunk Mptr})\ v = \text{Some } m'' \\ \text{is\_pointer\_prim\_type } pt = \text{true} \end{array}}{(\text{EE\_malloc } pt\ e, oe, m) \to (\text{Vptr } b\ \text{Ptrofs.zero}, oe, m'')}$$

3. gcmalloc: 为对象分配内存（可回收）

$$\frac{\begin{array}{c} \text{Mem.alloc } m\ 0\ (\text{sizeof } (\text{genv\_cenv } ge)\ t\ \text{default\_type\_attr}) = (m', b) \\ \text{is\_pointer\_prim\_type } pt = \text{true} \end{array}}{(\text{EE\_gcmalloc } pt\ t, oe, m) \to (\text{Vptr } b\ \text{Ptrofs.zero}, oe\{b \rightsquigarrow (t, 0, \text{true})\}, m')}$$

4. gcmallocjarray: 为数组对象分配内存

$$\frac{\begin{array}{c} e \xrightarrow{m} v \\ (v = \text{Vint } n \wedge \text{Int.unsigned } n = z) \vee (v = \text{Vlong } n \wedge \text{Int64.unsigned } n = z) \\ \text{Mem.alloc } m\ 0\ (\text{Z.mul } z\ (\text{sizeof } (\text{genv\_cenv } ge)\ t\ \text{default\_type\_attr})) = (m', b) \\ \text{is\_pointer\_prim\_type } pt = \text{true} \end{array}}{(\text{EE\_gcmallocjarray } pt\ t\ e, oe, m) \to (\text{Vptr } b\ \text{Ptrofs.zero}, oe\{b \rightsquigarrow (\text{Tarray } t\ z, 0, \text{true})\}, m')}$$

5. gcpermalloc: 为对象分配内存（不可回收）

$$\frac{\begin{array}{c} \text{Mem.alloc } m\ 0\ (\text{sizeof } (\text{genv\_cenv } ge)\ t\ \text{default\_type\_attr}) = (m', b) \\ \text{is\_pointer\_prim\_type } pt = \text{true} \end{array}}{(\text{EE\_gcpermalloc } pt\ t, oe, m) \to (\text{Vptr } b\ \text{Ptrofs.zero}, oe\{b \rightsquigarrow (t, 0, \text{false})\}, m')}$$

2.4 指令的小步语义

指令在运行过程中不会改变全局环境，因此 ge 仍然是不变量，但局部环境、对象环境和内存状态都会发生变化。下面是表达式求值的语义中需要用到的辅助函数及其作用。

1. find_label $(lbl : ident)$ $(s : statement)$ $(k : cont)$: option $cont$

   作用：根据当前指令 s 和控制流信息 k 以及要跳转的 label 查找要执行的指令并产生新的控制流信息。

2. superclass $(ce : composite\_env)$ $(id : ident)$: list $ident$

   作用：id 是类名，查找该类的所有父类。

3. superinterface $(ce : composite\_env)$ $(id : ident)$: list $ident$

   作用：id 是类名/接口名，查找该类实现的所有接口/该接口的所有父接口。

4. diapatch_function $(cid : ident)$ $(fid : ident)$: list $ident$

   作用：cid 是类名，fid 是函数名，如果该类声明了函数 fid，则返回对应的全局函数的 id, 否则递归地查找该类的父类。

5. find_handler $(t : type)$ $(ll : \text{list } ident)$ $(s : statement)$ $(k : cont)$: option $cont$

   作用：根据抛出的异常值的类型 t, try block 中的 label 列表，当前指令 s 和控制流信息 k 查找需要执行的 exception handler 并产生新的控制流信息。

6. resolve_ref $(oe : oenv)$ $(v : val) : type$

   作用：从对象环境中根据对象的地址值 v 解析出该对象的动态类型。

7. resolve_type $(oe : oenv)$ $(v : val)$ $(t : type) : type$

   作用：从对象环境中根据值 v 及其静态类型 t 解析出其动态类型。

8. function_entry $(f : concrete\_functioin)$ $(m : mem)$ $(l :$ list $(val * type)) :$ option $(lenv * mem)$

   作用：函数调用的初始化过程，f 是被调函数，m 是当前内存状态，l 是实参列表，进行局部环境的初始化，给局部变量分配内存空间，对形参变量赋值为对应的实参值，返回初始化后的局部环境和更新后的内存状态。

9. set_preg $(le : lenv)$ $(id : ident)$ $(v : val)$ $(pt : prim\_type) : lenv$

   作用：le 是当前运行函数的局部环境，id 是伪寄存器名，在 le 中将该伪寄存器的值更新为 v, 类型更新为 pt, 返回更新后的局部环境。

10. set_thrownval $(le : lenv)$ $(v : val)$ $(pt : prim\_type) : lenv$

    作用：le 是当前运行函数的局部环境，在 le 中将特殊寄存器 Thrownval 的值更新为 v, 类型更新为 pt, 返回更新后的局部环境。

11. select_switch $(n :$ Z$)$ $(dl : label)$ $(ll : list$ (Z $* ident)) : ident$

    作用：在 ll 中查找和 n 相等的第一个整数并返回对应的 label，如果找不到则返回默认的 dl.

12. dassign $(le : lenv)$ $(m : mem)$ $(vid : var\_id)$ $(fi : field\_id)$ $(v : val)$ $(t : type) :$ option mem

    作用：le 是当前运行函数的局部环境，m 是当前内存状态，在内存中找到变量 vid 并将 fi 字段的值更新为类型为 t 的值 v, 返回更新后的内存状态。

13. dassign_list $(le : lenv)$ $(m : mem)$ $(l :$ list $(var\_id * field\_id)$ $(vl :$ list $(val * type)) :$ option mem

    作用：依次调用 dassign 更新多个变量的值。

14. blocks_of_lenv $(le : lenv) :$ list $(block *$ Z $*$ Z$)$

    作用：le 是当前运行函数的局部环境，根据 le 获得当前函数的局部变量在内存中占用的空间。

15. sem_cast_list $(vl$: list $val)$ $(tl1$: list $type)$ $(tl2$: list $type)$ $(m$: $mem)$: option (list $(val * type)$)

    作用：依次调用 sem_cast 对多个值进行类型转换。

16. typeof_list $(el$: list $expr)$: list $type$

    作用：依次调用 typeof 获取多个表达式的结果类型。

17. sem_switch_arg $(v : val)$ $(t : type)$: option Z

    作用：将值 v 转换为整数值。

18. typeof_ext_expr $(ee : ext\_expr)$: $type$

    作用：获取拓展表达式 ee 的结果类型。

19. typeof_params $(l$: list $(ident * type))$: list $type$

    作用：将函数的形参列表中的每个参数的类型抽取出来形成一个列表。

下面是指令的小步语义，用 $state \to state$ 的形式来表示指令的小步语义。

1. skip

$$\frac{}{\begin{array}{c}\text{NormalState } f \text{ S\_skip (Kwhile } e\ s\ k)\ le\ oe\ m \\ \to \text{NormalState } f \text{ (S\_while } e\ s)\ k\ le\ oe\ m\end{array}}$$

$$\frac{\begin{array}{c}\text{is\_call\_cont } k \\ \text{fun\_returns (fst } f) = \text{nil} \\ \text{Mem.free\_list } m \text{ (blocks\_of\_lenv } le) = \text{Some } m'\end{array}}{\text{NormalState } f \text{ S\_skip } k\ le\ oe\ m \to \text{ReturnState nil } k\ oe\ m'}$$

2. dassign

$$\frac{\begin{array}{c}(ee, oe, m) \xrightarrow{le} (v, oe', m') \\ \text{dassign } le\ m'\ vid\ fi\ v \text{ (typeof\_ext\_expr } ee) = \text{Some } m''\end{array}}{\begin{array}{c}\text{NormalState } f \text{ (S\_dassign } vid\ fi\ ee)\ k\ le\ oe\ m \\ \to \text{NormalState } f \text{ S\_skip } k\ le\ oe'\ m''\end{array}}$$

3. iassign

$$\frac{\begin{array}{c} e1 \overset{le,m}{\to} \text{Vptr } loc\ ofs \\ t = \text{Tpointer } t' \\ \text{fieldoffset (genv\_cenv } ge)\ t'\ fi = \text{Some } (t'', delta) \\ (ee, oe, m) \overset{le}{\to} (v, oe', m') \\ \text{sem\_cast } v\ (\text{typeof\_ext\_expr } ee)\ t''\ m' = \text{Some } v' \\ \text{Ptrofs.add } ofs\ (\text{Ptrofs.repr } delta) = ofs' \\ \text{assign\_loc (genv\_cenv } ge)\ t''\ \text{default\_type\_attr}\ m'\ loc\ ofs'\ v' = \text{Some } m'' \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_iassign } t\ fi\ e\ ee)\ k\ le\ oe\ m \\ \to \text{NormalState } f\ \text{S\_skip}\ k\ le\ oe'\ m'' \end{array}}$$

4. regassign

$$\frac{\begin{array}{c} (\text{lenv\_pregs } le)\ id = \text{Some } (pt, v0) \vee (\text{lenv\_pregs } le)\ id = \text{None} \\ (ee, oe, m) \overset{le}{\to} (v, oe', m') \\ \text{sem\_cast } v\ (\text{typeof\_ext\_expr } e)\ (\text{Tprim } pt)\ m' = \text{Some } v' \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_regassign } pt\ (\text{Preg } id)\ ee)\ k\ le\ oe\ m \\ \to \text{NormalState } f\ \text{S\_skip}\ k\ (\text{set\_preg } le\ id\ v'\ pt)\ oe'\ m' \end{array}}$$

$$\frac{\begin{array}{c} (ee, oe, m) \overset{le}{\to} (v, oe', m') \\ \text{sem\_cast } v\ (\text{typeof\_ext\_expr } e)\ (\text{Tprim } pt)\ m' = \text{Some } v' \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_regassign } pt\ \text{Thrownval}\ ee)\ k\ le\ oe\ m \\ \to \text{NormalState } f\ \text{S\_skip}\ k\ (\text{set\_thrownval } le\ v'\ pt)\ oe'\ m' \end{array}}$$

5. seq

$$\frac{}{\text{NormalState } f\ (\text{S\_seq } s1\ s2)\ k\ le\ oe\ m \to \text{NormalState } f\ s1\ (\text{Kseq } s2\ k)\ le\ oe\ m}$$

$$\frac{}{\text{NormalState } f\ \text{S\_skip}\ (\text{Kseq } s\ k)\ le\ oe\ m \to \text{NormalState } f\ s\ k\ le\ oe\ m}$$

6. label

$$\frac{}{\text{NormalState } f\ (\text{S\_label } lbl\ s)\ k\ le\ oe\ m \to \text{NormalState } f\ s\ k\ le\ oe\ m}$$

7. if

$$\frac{\begin{array}{c} e \overset{le,m}{\to} v \\ \text{bool\_val } v\ (\text{typeof } e)\ m = \text{Some } b \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_if } e\ s1\ s2)\ k\ le\ oe\ m \\ \to \text{NormalState } f\ (\text{if } b\ \text{then } s1\ \text{else } s2)\ k\ le\ oe\ m \end{array}}$$

8. while

$$\frac{}{\begin{array}{c}\text{NormalState } f\ (\text{S\_while } e\ s)\ k\ le\ oe\ m \\ \rightarrow \text{NormalState } f\ (\text{S\_if } e\ (\text{S\_seq } s\ (\text{S\_while } e\ s))\ \text{S\_skip})\ k\ le\ oe\ m\end{array}}$$

9. goto

$$\frac{\text{find\_label } lbl\ (\text{fun\_statement } (\text{snd} f))\ (\text{call\_cont } k) = \text{Some } (s', k')}{\text{NormalState } f\ (\text{S\_goto } lbl)\ k\ le\ oe\ m \rightarrow \text{NormalState } f\ s'\ k'\ le\ oe\ m}$$

10. return

$$\frac{\begin{array}{c}el \overset{le,m}{\rightarrow}_* vl \\ \text{typeof\_list } el = tl \\ \text{sem\_cast\_list } vl\ tl\ (\text{fun\_returns } (\text{fst} f))\ m = \text{Some } vl' \\ \text{Mem.free\_list } m\ (\text{blocks\_of\_lenv } le) = \text{Some } m'\end{array}}{\text{NormalState } f\ (\text{S\_return } el)\ k\ le\ oe\ m \rightarrow \text{ReturnState } vl'\ (\text{call\_cont } k)\ oe\ m'}$$

11. switch

$$\frac{\begin{array}{c}e \overset{le,m}{\rightarrow} v \\ \text{sem\_switch\_arg } v\ (\text{typeof } e) = \text{Some } n\end{array}}{\begin{array}{c}\text{NormalState } f\ (\text{S\_switch } e\ dl\ ll)\ k\ le\ oe\ m \\ \rightarrow \text{NormalState } f\ (\text{S\_goto } (\text{select\_switch } n\ dl\ ll))\ k\ le\ oe\ m\end{array}}$$

12. callassigned

$$\frac{\begin{array}{c}el \overset{le,m}{\rightarrow}_* vl \\ \text{typeof\_list } el = tl \\ (\text{genv\_funcs } ge)\ fid = \text{Some } loc \\ (\text{genv\_fundefs } ge)\ loc = \text{Some } f' \\ \text{sem\_cast\_list } vl\ tl\ (\text{typeof\_params } (\text{fun\_params } (\text{fst } f'))) \ m = \text{Some } vl'\end{array}}{\begin{array}{c}\text{NormalState } f\ (\text{S\_callassigned } fid\ el\ l)\ k\ le\ oe\ m \\ \rightarrow \text{CallState } f'\ vl'\ (\text{Kcall } l\ f\ le\ k)\ oe\ m\end{array}}$$

13. icallassigned

$$\frac{\begin{array}{c} e \overset{le,m}{\to} \text{Vptr } loc \text{ Ptrofs.zero} \\ el \overset{le,m}{\to}_* vl \\ \text{typeof\_list } el = tl \\ (\text{genv\_fundefs } ge)\ loc = \text{Some } f' \\ \text{sem\_cast\_list } vl\ tl\ (\text{typeof\_params } (\text{fun\_params } (\text{fst } f'))) \ m = \text{Some } vl' \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_icallassigned } e\ el\ l)\ k\ le\ oe\ m \\ \to \text{CallState } f'\ vl'\ (\text{Kcall } l\ f\ le\ k)\ oe\ m \end{array}}$$

14. virtualcallassigned

$$\frac{\begin{array}{c} e \overset{le,m}{\to} v \\ el \overset{le,m}{\to}_* vl \\ \text{resolve\_ref } oe\ v = \text{Some } (\text{Tcomposite } cid') \\ \text{In } cid\ (\text{superclasses\_id } (\text{genv\_cenv } ge)\ cid') \\ \text{dispatch\_function } cid'\ fid = \text{Some } fid' \\ \text{typeof\_list } (e :: el) = tl \\ (\text{genv\_funcs } ge)\ fid' = \text{Some } loc \\ (\text{genv\_fundefs } ge)\ loc = \text{Some } f' \\ \text{sem\_cast\_list } (v :: vl)\ tl\ (\text{typeof\_params } (\text{fun\_params } (\text{fst } f'))) \ m = \text{Some } vl' \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_virtualcallassigned } cid\ fid\ el\ l)\ k\ le\ oe\ m \\ \to \text{CallState } f'\ vl'\ (\text{Kcall } l\ f\ le\ k)\ oe\ m \end{array}}$$

15. interfacecallassigned

$$\frac{\begin{array}{c} e \overset{le,m}{\to} v \\ el \overset{le,m}{\to}_* vl \\ \text{resolve\_ref } oe\ v = \text{Some } (\text{Tcomposite } cid') \\ \text{In } cid\ (\text{superinterfaces\_id } (\text{genv\_cenv } ge)\ cid') \\ \text{dispatch\_function } cid'\ fid = \text{Some } fid' \\ \text{typeof\_list } (e :: el) = tl \\ (\text{genv\_funcs } ge)\ fid' = \text{Some } loc \\ (\text{genv\_fundefs } ge)\ loc = \text{Some } f' \\ \text{sem\_cast\_list } (v :: vl)\ tl\ (\text{typeof\_params } (\text{fun\_params } (\text{fst } f'))) \ m = \text{Some } vl' \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_interfacecallassigned } cid\ fid\ el\ l)\ k\ le\ oe\ m \\ \to \text{CallState } f'\ vl'\ (\text{Kcall } l\ f\ le\ k)\ oe\ m \end{array}}$$

16. javatry

$$\frac{}{\text{NormalState } f\ (\text{S\_javatry } ll\ s)\ k\ le\ oe\ m \to \text{NormalState } f\ s\ (\text{Kjavatry } ll\ k)\ le\ oe\ m}$$

17. throw

$$\frac{e \overset{le,m}{\to} v}{\text{NormalState } f\ (\text{S\_throw } e)\ k\ le\ oe\ m \to \text{ExceptionState } f\ (\text{set\_thrownval } le\ v\ (\text{prim\_type\_of } e))\ k\ le\ oe\ m}$$

18. javacatch

$$\frac{\begin{array}{c}\text{lenv\_thrownval } le = \text{Some } (v,t) \\ \text{catch\_cont } k = \text{Kjavatry } ll\ k1 \\ \text{call\_cont } k1 = k2 \\ \text{resolve\_type } oe\ v\ t = t' \\ \text{find\_handler } t'\ ll\ (\text{fun\_statement } (\text{snd } f))\ k2 = \text{Some } k3\end{array}}{\text{ExceptionState } f\ k\ le\ oe\ m \to \text{NormalState } f\ \text{S\_skip } k3\ le\ oe\ m}$$

$$\frac{\begin{array}{c}\text{lenv\_thrownval } le = \text{Some } (v,t) \\ \text{catch\_cont } k = \text{Kjavatry } ll\ k1 \\ \text{call\_cont } k1 = k2 \\ \text{resolve\_type } oe\ v\ t = t' \\ \text{find\_handler } t'\ ll\ (\text{fun\_statement } (\text{snd } f))\ k2 = \text{None}\end{array}}{\text{ExceptionState } f\ k\ le\ oe\ m \to \text{ExceptionState } f\ k1\ le\ oe\ m}$$

$$\frac{\begin{array}{c}\text{catch\_cont } k = \text{Kcall } l\ f'\ le'\ k1 \\ \text{Mem.free\_list } m\ (\text{blocks\_of\_lenv } le) = \text{Some } m'\end{array}}{\text{ExceptionState } f\ k\ le\ oe\ m \to \text{ExceptionState } f'\ k1\ le'\ oe\ m'}$$

19. free

$$\frac{\begin{array}{c} e \overset{le,m}{\to} \text{Vptr } loc\ ofs \\ \text{Ptrofs.unsigned } ofs - \text{size\_chunk Mptr} = low \\ \text{Ptrofs.unsigned } lo + \text{Ptrofs.unsigned } sz = high \\ \text{Mem.load Mptr } m\ loc\ low = \text{Some } v \\ \text{val\_to\_ptrofs } v = \text{Some } sz \\ \text{Ptrofs.unsigned } sz > 0 \\ \text{Mem.free } m\ b\ low\ high = \text{Some } m' \end{array}}{\text{NormalState } f\ (\text{S\_free } e)\ k\ le\ oe\ m \to \text{NormalState } f\ \text{S\_skip}\ k\ le\ oe\ m'}$$

$$\frac{e \overset{le,m}{\to} \text{Vnullptr}}{\text{NormalState } f\ (\text{S\_free } e)\ k\ le\ oe\ m \to \text{NormalState } f\ \text{S\_skip}\ k\ le\ oe\ m}$$

20. incref

$$\frac{\begin{array}{c} \text{prim\_type\_of } e = \text{PTref} \\ e \overset{le,m}{\to} \text{Vptr } loc\ (\text{Ptrofs.repr } 0) \\ oe\ loc = \text{Some } (t, n, b) \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_incref } e)\ k\ le\ oe\ m \\ \to \text{NormalState } f\ \text{S\_skip}\ k\ le\ oe\{loc \rightsquigarrow (t, \text{succ } n, b)\}\ m \end{array}}$$

21. decref

$$\frac{\begin{array}{c} \text{prim\_type\_of } e = \text{PTref} \\ e \overset{le,m}{\to} \text{Vptr } loc\ (\text{Ptrofs.repr } 0) \\ oe\ loc = \text{Some } (t, n, b) \end{array}}{\begin{array}{c} \text{NormalState } f\ (\text{S\_decref } e)\ k\ le\ oe\ m \\ \to \text{NormalState } f\ \text{S\_skip}\ k\ le\ oe\{loc \rightsquigarrow (t, \text{pred } n, b)\}\ m \end{array}}$$

22. eval: 求值

$$\frac{e \overset{le,m}{\to} v}{\text{NormalState } f\ (\text{S\_eval } e)\ k\ le\ oe\ m \to \text{NormalState } f\ \text{S\_skip}\ k\ le\ oe\ m}$$

23. CallState

$$\frac{\begin{array}{c} f = (fp, \text{Some } fb) \\ \text{function\_entry } (fp, fb)\ m\ vl = \text{Some } (le, m') \end{array}}{\begin{array}{c} \text{CallState } f\ vl\ k\ oe\ m \\ \to \text{NormalState } (fp, fb)\ (\text{fun\_statement } fb)\ (\text{Vundef}, \text{Tprim PTvoid})\ k\ le\ oe\ m' \end{array}}$$

24. ReturnState

$$\frac{\text{dassign\_list } le\ m\ l\ vl = \text{Some } m'}{\text{ReturnState } vl\ (\text{Kcall } l\ f\ le\ k)\ oe\ m \rightarrow \text{NormalState } f\ \text{S\_skip } k\ le\ oe\ m'}$$