Softbound

March 23, 2009

Contents

List of Figures

1 Type

Qualifier:

$$q ::= \quad \texttt{safe} \mid \quad \texttt{seq} \mid \quad \texttt{tame} \quad \textit{Qualifier}$$

Types:

a	::=		*atomic types*
		$\texttt{int}$	*int*
		$p * q$	*pointer type*
p	::=		*pointer types*
		a	*atomic type*
		s	*anonymous struct type*
		id	*named struct type*
		$\texttt{void}$	*void*
s	::=		*struct types*
		$\cdot$	*nil struct*
		$s \; ; \; id : a$	*cons struct*
tab	::=	$id \mapsto \texttt{option}\ s$	*named struct look-up table*

Size of Type:

$asize(a)$	::=		*where*
		1	$a = \texttt{int}$
		1	$a = p * q$
$psize(p)$	::=		*where*
		$asize(a)$	$p = a$
		$ssize(s)$	$p = s$
		$ssize(s)$	$p = n \;\; tab\, n = \texttt{some}\ s$
		1	$p = \texttt{void}$
$ssize(s)$	::=		*where*
		0	$s = \cdot$
		$ssize(s') + asize(a)$	$s = s' \; ; \; id : a$

Figure 1: Type.

$\boxed{a \equiv_a a}$

$$\frac{}{\texttt{int} \equiv_a \texttt{int}}\ \text{(E-Int)}$$

$$\frac{p_1 \equiv_p p_2 \qquad q_1 = q_2}{p_1 * q_1 \equiv_a p_2 * q_2}\ \text{(E-Pointer)}$$

$\boxed{p \equiv_p p}$

$$\frac{a_1 \equiv_a a_2}{a_1 \equiv_p a_2}\ \text{(E-Atomic)}$$

$$\frac{s_1 \equiv_s s_2}{s_1 \equiv_p s_2}\ \text{(E-AStruct)}$$

$$\frac{\text{tab}\ n_1 = \texttt{some}\ s_1 \qquad \text{tab}\ n_2 = \texttt{some}\ s_2 \qquad s_1 \equiv_s s_2}{n_1 \equiv_p n_2}\ \text{(E-NStruct)}$$

$$\frac{}{n \equiv_p n}\ \text{(E-EqName)}$$

$$\frac{}{\texttt{void} \equiv_p \texttt{void}}\ \text{(E-Void)}$$

$\boxed{s \equiv_s s}$

$$\frac{}{\cdot \equiv_s \cdot}\ \text{(E-Nil)}$$

$$\frac{s_1 \equiv_s s_2 \qquad a_1 \equiv_a a_2}{s_1\ ;\ id_1 : a_1 \equiv_s s_2\ ;\ id_2 : a_2}\ \text{(E-Cons)}$$

Figure 2: Type Equivalence.

$\boxed{a \Rightarrow a}$

$$\frac{}{\texttt{int} \Rightarrow \texttt{int}}\ \text{(C-Int)}$$

$$\frac{}{p * q \Rightarrow \texttt{int}}\ \text{(C-Ptr-Int)}$$

$$\frac{q \neq \texttt{safe}}{\texttt{int} \Rightarrow p * q}\ \text{(C-Int-Ptr)}$$

$$\frac{p_1 \equiv_p p_2}{p_1 * \texttt{safe} \Rightarrow p_2 * \texttt{safe}}\ \text{(C-SafePtr)}$$

$$\frac{p_1 \equiv_p p_2}{p_1 * \texttt{safe} \Rightarrow p_2 * \texttt{seq}}\ \text{(C-SafeSeqPtr)}$$

$$\frac{p_1 \equiv_p p_2}{p_1 * \texttt{seq} \Rightarrow p_2 * \texttt{safe}}\ \text{(C-SeqSafePtr)}$$

$$\frac{p_1 \equiv_p p_2}{p_1 * \texttt{seq} \Rightarrow p_2 * \texttt{seq}}\ \text{(C-SeqPtr)}$$

$$\frac{}{p_1 * \texttt{tame} \Rightarrow p_2 * \texttt{tame}}\ \text{(C-TamePtr)}$$

Figure 3: Type Conversion.

$\boxed{\vdash_a^{tame*} \ a}$

$$\frac{}{\vdash_a^{tame*} \ \texttt{int}} \ \text{(TC-Int)}$$

$$\frac{\vdash_p^{tame*} \ p}{\vdash_a^{tame*} \ p * \texttt{tame}} \ \text{(TC-Ptr)}$$

$\boxed{\vdash_p^{tame*} \ p}$

$$\frac{\vdash_a^{tame*} \ a}{\vdash_p^{tame*} \ a} \ \text{(TC-Atomic)}$$

$$\frac{\vdash_s^{tame*} \ s}{\vdash_p^{tame*} \ s} \ \text{(TC-AStruct)}$$

$$\frac{\text{tab} \ n = \texttt{some} \ s \qquad \vdash_s^{tame*} \ s}{\vdash_p^{tame*} \ n} \ \text{(TC-NStruct)}$$

$$\frac{}{\vdash_p^{tame*} \ \texttt{void}} \ \text{(TC-Void)}$$

$\boxed{\vdash_s^{tame*} \ s}$

$$\frac{}{\vdash_s^{tame*} \ .} \ \text{(TC-Nil)}$$

$$\frac{\vdash_a^{tame*} \ a \qquad \vdash_s^{tame*} \ s}{\vdash_s^{tame*} \ s \ ; \ id{:}a} \ \text{(TC-Cons)}$$

Figure 4: Tame Pointer Closure.

$\boxed{\vdash_a \ a}$

$$\frac{}{\vdash_a \ \texttt{int}} \text{ (WFT-Int)}$$

$$\frac{\vdash_p \ p \qquad q \neq \texttt{tame} \qquad p \neq \texttt{void}}{\vdash_a \ p * q} \text{ (WFT-NonTamePtr)}$$

$$\frac{\vdash_p^{tame*} \ p}{\vdash_a \ p * \texttt{tame}} \text{ (WFT-TamePtr)}$$

$\boxed{\vdash_p \ p}$

$$\frac{\vdash_a \ a}{\vdash_p \ a} \text{ (WFT-Atomic)}$$

$$\frac{\vdash_s \ s}{\vdash_p \ s} \text{ (WFT-AStruct)}$$

$$\frac{\text{tab}\ n = \texttt{some}\ s \qquad \vdash_s \ s}{\vdash_p \ n} \text{ (WFT-NStruct)}$$

$$\frac{}{\vdash_p \ \texttt{void}} \text{ (WFT-Void)}$$

$\boxed{\vdash_s \ s}$

$$\frac{}{\vdash_s \ .} \text{ (WFt-Nil)}$$

$$\frac{\vdash_a \ a \qquad \vdash_s \ s}{\vdash_s \ s \ ; \ id{:}a} \text{ (WFT-Cons)}$$

Figure 5: Well-formed Type.

2 Env

2.1 primitives

Table 1: primitives

Name	Function	
baseAddr	$\mathbb{N}$	lowest user-accessible addr
maxAddr	$\mathbb{N}$	max user-accessible addr
TOP	$\mathbb{N}$	stack top addr
Stack	$v \mapsto \texttt{option}\ (d * a)$	stack
Mem	$l \mapsto d_{(b,e)}$	memory
TypeInfo	$l \mapsto a$	type information
Env	$(Stack, Mem, TypeInfo)$	environment
Value	$\mathbb{N}$	Value of Memory
Base	$\mathbb{N}$	MetaData, Base of Memory
End	$\mathbb{N}$	MetaData, Bound of Memory
Loc	$\mathbb{N}$	Location of Memory
readMem	$Mem \to Loc \to \texttt{option}\ Value$	read data
readMemMeta	$Mem \to Loc \to \texttt{option}\ Value_{(Base,End)}$	read data with meta
writeMem	$Mem \to Loc \to Value \to \texttt{option}\ Mem$	write data
writeMemMeta	$Mem \to Loc \to Value_{(Base,End)} \to \texttt{option}\ Mem$	write data with meta
malloc	$Env \to \mathbb{N} \to \texttt{option}\ (Env * Loc)$	memory allocation
updateTI	$TypeInfo \to Loc \to PtrType \to \mathbb{N} \to TypeInfo$	updating type information

- readMem $M\ l$: read data from the location l if it is accessible
- readMemMeta $M\ l$: read data with meta from the location l if it is accessible
- writeMem $M\ l\ d$: write data to the location l if it is accessible
- writeMemMeta $M\ l\ d_{(b,e)}$: write data with meta to the location l if it is accessible
- malloc : memory allocation
- updateTI : updating type information

2.2 defined functions

- readMemBlock $M\ l\ size$
- readMemMetaBlock $M\ l\ size$
- writeMemBlock $M\ l\ d^*\ size$
- writeMemMetaBlock $M\ l\ d^*\ size$
- copyMemBlock $M\ l\ d^*\ size$
- copyMemMetaBlock $M\ l\ d^*\ size$
- validMem $M\ l \triangleq \exists d.$readMem $M\ l = \texttt{some}\ d \wedge \forall d.\ \exists M'.$writeMem $M\ ld = \texttt{some}\ M'$
- validMemBlock $M\ l\ size$

2.3 axioms

axiom 2.1 (validAddressRange) $0 < baseAddr \leq maxAddr$

axiom 2.2 (valid memory)

1. $\forall(M, l).\ (\exists d.readMem\ M\ l = \texttt{some}\ d) \Leftrightarrow (\forall d.\ \exists M'.writeMem\ M\ ld)$.
2. $\forall(M, l).\ (\exists d_{(b,e)}.readMemMeta\ M\ l = \texttt{some}\ d_{(b,e)}) \Leftrightarrow (\forall d_{(b,e)}.\ \exists M'.writeMemMeta\ M\ ld_{(b,e)})$.
3. $\forall(M, l).\ (\exists d.readMem\ M\ l = \texttt{some}\ d) \Leftrightarrow (\exists d_{(b,e)}.readMemMeta\ M\ l = \texttt{some}\ d_{(b,e)})$.
4. $\forall(M, l).\ (\forall d.\ \exists M'.writeMem\ M\ ld) \Leftrightarrow (\forall d_{(b,e)}.\ \exists M'.writeMemMeta\ M\ ld_{(b,e)})$.

axiom 2.3 (unique result)

1. $\forall(M, l).\ (\forall(d, d').\ readMem\ M\ l = \texttt{some}\ d \wedge readMem\ M\ l = \texttt{some}\ d' \Rightarrow d = d')$.
2. $\forall(M, l).\ (\forall(d_{(b,e)}, d'_{(b',e')}).\ readMemMeta\ M\ l = \texttt{some}\ d_{(b,e)} \wedge readMemMeta\ M\ l = \texttt{some}\ d'_{(b',e')} \Rightarrow d_{(b,e)} = d'_{(b',e')})$.
3. $\forall(M, l).\ (\forall(d, d').\ \exists M'.writeMem\ M\ ld \Leftrightarrow \exists M'.writeMem\ M\ ld')$.
4. $\forall(M, l).\ (\forall(d_{(b,e)}, d'_{(b',e')}).\ \exists M'.writeMemMeta\ M\ ld_{(b,e)} \Leftrightarrow \exists M'.writeMemMeta\ M\ ld'_{(b',e')})$.

axiom 2.4 (updateTypeInfo inversion)

1. *If* $\vdash_a\ \ p * q$, $q \neq \texttt{tame}$, $psize(p) > 0$ *and updateTypeInfo* $TI\ l\ p\ size = TI'$, *then* $\forall(l' \in [l, l + size)).\ TI'(l') = p[(l' - l) \bmod psize(p)]_t$ *and* $\forall(l' < l \vee l' \geq l + size)).\ TI(l') = TI'(l')$.
2. *If* $\vdash_a\ \ p * \texttt{tame}$, $psize(p) > 0$ *and updateTypeInfo* $TI\ l\ p\ size = TI'$, *then* $\forall(l' \in [l, l+size)).\ TI'(l') = \texttt{void} * \texttt{tame}$ *and* $\forall(l' < l \vee l' \geq l + size)).\ TI(l') = TI'(l')$.

axiom 2.5 (malloc inversion) *If* $malloc\ E\ size = \texttt{some}\ ((M', S', TI'), l)$, *then*

1. $\exists M, TI.E = (M, S', TI)$
2. $baseAddr \leq l \wedge l + size < maxAddr \wedge size > 0$
3. $\forall(readMemMeta\ M\ l' = \texttt{some}\ d_{(b,e)}).\ readMemMeta\ M'\ l' = \texttt{some}\ d_{(b,e)}$
4. $\forall(l' < l \vee l' \geq l + size).\ readMemMeta\ M\ l' = \texttt{none} \Rightarrow readMemMeta\ M'\ l' = \texttt{none}$
5. $\forall(l \leq l' < l + size).\ readMemMeta\ M\ l' = \texttt{none} \wedge readMemMeta\ M'\ l' = \texttt{some}\ 0_{(0,0)}$
6. $\forall(l' < l \vee l' \geq l + size).\ TI(l') = TI'(l')$
7. $\forall(l' < l \vee l' \geq l + size).\ TI'(l') = \texttt{int}$

axiom 2.6 (writeMem Inversion)

1. *If* $writeMemMeta\ M\ ld_{(b,e)} = \texttt{some}\ M'$, *then*
   (a) $readMemMeta\ M'\ l = \texttt{some}\ d_{(b,e)}$
   (b) *If* $\forall(l' \neq l).\ readMemMeta\ M\ l' = d$, *then* $readMemMeta\ M'\ l = d$
   (c) *If* $\forall l'.\ readMemMeta\ M\ l' = \texttt{none}$, *then* $readMemMeta\ M'\ l = \texttt{none}$
2. *If* $writeMem\ M\ ld = \texttt{some}\ M'$, *then*
   (a) *If* $readMemMeta\ M\ l = \texttt{some}\ d'_{(b,e)}$, *then* $readMemMeta\ M'\ l = \texttt{some}\ d_{(b,e)}$
   (b) *If* $\forall(l' \neq l).\ readMemMeta\ M\ l' = d$, *then* $readMemMeta\ M'\ l = d$
   (c) *If* $\forall l'.\ readMemMeta\ M\ l' = \texttt{none}$, *then* $readMemMeta\ M'\ l = \texttt{none}$

2.4 well-formed environment

$\boxed{M\ ;\ TI \vdash_S\ S}$

$$\frac{\begin{array}{l} baseAddr \le TOP \le maxAddr \\ \forall(v,l,a).\ (v \mapsto (l,a)) \in S \Rightarrow \\ \quad (TOP \le l) \wedge ((l + asize(a)) < maxAddr) \\ \forall(v,l,a).\ (v \mapsto (l,a)) \in S \Rightarrow \\ \quad \forall(v',l',a').\ (v' \mapsto (l',a')) \in S \Rightarrow \\ \quad\quad (l' \ge (l + asize(a))) \vee (l \ge (l' + asize(a'))) \end{array}}{M\ ;\ TI \vdash_S\ S}\ \text{(WF-Stack)}$$

$\boxed{M\ ;TI\ \vdash_D\ d_{(b,e)} : a}$

$$\begin{array}{rcll} M\ ;TI\ \vdash_D\ d_{(b,e)} : \texttt{int} & \triangleq & \texttt{true} & \text{(WFD-Int)} \\ M\ ;TI\ \vdash_D\ d_{(b,e)} : a * \texttt{safe} & \triangleq & (d = 0) \vee & \text{(WFD-ASafe)} \\ & & ((baseAddr \le d) \wedge (d + 1 < maxAddr) \wedge & \\ & & (\text{validMem}\ M\ d) \wedge (TI(d) = a)) & \\ M\ ;TI\ \vdash_D\ d_{(b,e)} : s * \texttt{safe} & \triangleq & (d = 0) \vee & \text{(WFD-SSafe)} \\ & & ((ssize(s) > 0) \wedge & \\ & & (baseAddr \le d) \wedge (d + ssize(s) < maxAddr) \wedge & \\ & & (\forall(i \in [0, ssize(s))). & \\ & & \quad (\text{validMem}\ M\ d + i \wedge (TI(d + i) = s[i]))) & \\ M\ ;TI\ \vdash_D\ d_{(b,e)} : n * \texttt{safe} & \triangleq & (d = 0) \vee & \text{(WFD-NSafe)} \\ & & (\exists s.\ \text{tab}\ n = \texttt{some}\ s \wedge & \\ & & (ssize(s) > 0) \wedge & \\ & & (baseAddr \le d) \wedge (d + ssize(s) < maxAddr) \wedge & \\ & & (\forall(i \in [0, ssize(s))). & \\ & & \quad (\text{validMem}\ M\ d + i \wedge (TI(d + i) = s[i]))) & \\ M\ ;TI\ \vdash_D\ d_{(b,e)} : \texttt{void} * \texttt{safe} & \triangleq & \texttt{false} & \text{(WFD-VSafe)} \\ M\ ;TI\ \vdash_D\ d_{(b,e)} : p * \texttt{seq} & \triangleq & (b = 0) \vee & \text{(WFD-Seq)} \\ & & ((b \ne 0) \wedge & \\ & & (baseAddr \le b \le e < maxAddr) \wedge & \\ & & (\forall(i \in [b, e)). & \\ & & \quad (\text{validMem}\ M\ i \wedge (TI(i) = p[(i - d) \bmod psize(p)]))) & \\ M\ ;TI\ \vdash_D\ d_{(b,e)} : p * \texttt{tame} & \triangleq & (b = 0) \vee & \text{(WFD-Tame)} \\ & & ((b \ne 0) \wedge & \\ & & (baseAddr \le b \le e < maxAddr) \wedge & \\ & & (\forall(i \in [b, e)). & \\ & & \quad (\text{validMem}\ M\ i \wedge \exists q.TI(i) = q * \texttt{tame})) & \end{array}$$

$\boxed{\vdash_M M\ ;\ TI}$

$$\frac{\forall(l,d,b,e).\ \text{readMemMeta}\ M\ l = d_{(b,e)} \Rightarrow M\ ;TI\ \vdash_D\ d_{(b,e)} : TI(l)}{\vdash_M M\ ;\ TI}\ \text{(WF-MemTI)}$$

$\boxed{\vdash_E E}$

$$\frac{\begin{array}{l} \vdash_M E.M\ ;\ E.TI \qquad E.M\ ;\ E.TI \vdash_S\ E.S \\ \forall(v,l,a).\ (v \mapsto (l,a)) \in E.S \Rightarrow \text{validMem}\ E.M\ l \wedge E.TI(l) = a \end{array}}{\vdash_E E}\ \text{(WF-Env)}$$

Figure 6: Well-formed Environment.

3 Syntax

Syntax:

$lhs ::=$		*lhs expressions*
	v	*variable*
	$*lhs$	*dereference*
	$lhs \rightarrow_s id$	*struct pos*
	$lhs \rightarrow_n id$	*name pos*
$rhs ::=$		*rhs expressions*
	i	*int constant*
	lhs	*lhs expression*
	$(a * q)\&lhs$	*reference*
	$rhs + rhs$	*addition*
	$(a)rhs$	*cast*
	$(sizeof)a$	*size*
	$(a * q)malloc\, rhs$	*alloc*
$c ::=$		*commands*
	$skip$	*skip*
	$c\ ;\ c$	*sequence*
	$lhs\ =\ rhs$	*assignment*

Figure 7: Syntax.

$\boxed{S \vdash_l lhs : a}$

$$\frac{(v \mapsto (l, a)) \in S \qquad \vdash_a a}{S \vdash_l v : a} \text{ (WFL-Var)}$$

$$\frac{S \vdash_l lhs : a * q}{S \vdash_l *lhs : a} \text{ (WFL-Def)}$$

$$\frac{S \vdash_l lhs : s * q \qquad s[id] = a}{S \vdash_l lhs \to_s id : a} \text{ (WFL-StructPos)}$$

$$\frac{S \vdash_l lhs : n * q \qquad \text{tab } n = \texttt{some } s \qquad s[id] = a}{S \vdash_l lhs \to_n id : a} \text{ (WFL-NamePos)}$$

$\boxed{S \vdash_l^{!tame} lhs : a}$

$$\frac{(v \mapsto (l, a)) \in S \qquad \vdash_a a}{S \vdash_l v : a} \text{ (WFLNT-Var)}$$

$$\frac{S \vdash_l^{!tame} lhs : s * q \qquad q \neq \texttt{tame} \qquad s[id] = a}{S \vdash_l^{!tame} lhs \to_s id : a} \text{ (WFLNT-StructPos)}$$

$$\frac{S \vdash_l^{!tame} lhs : n * q \qquad q \neq \texttt{tame} \qquad \text{tab } n = \texttt{some } s \qquad s[id] = a}{S \vdash_l^{!tame} lhs \to_n id : a} \text{ (WFLNT-NamePos)}$$

$\boxed{S \vdash_l^{tame} lhs : a}$

$$\frac{(v \mapsto (l, \texttt{void} * \texttt{tame})) \in S}{S \vdash_l^{tame} v : \texttt{void} * \texttt{tame}} \text{ (WFLT-Var)}$$

$$\frac{S \vdash_l^{tame} lhs : s * \texttt{tame} \qquad s[id] = \texttt{void} * \texttt{tame}}{S \vdash_l^{tame} lhs \to_s id : \texttt{void} * \texttt{tame}} \text{ (WFLT-StructPos)}$$

$$\frac{S \vdash_l^{tame} lhs : n * \texttt{tame} \qquad \text{tab } n = \texttt{some } s \qquad s[id] = \texttt{void} * \texttt{tame}}{S \vdash_l^{tame} lhs \to_n id : \texttt{void} * \texttt{tame}} \text{ (WFLT-NamePos)}$$

Figure 8: Well-formed LHS.

$\boxed{S \vdash_r rhs : a}$

$$\frac{}{S \vdash_r i : \texttt{int}} \text{ (WFR-Const)}$$

$$\frac{S \vdash_l lhs : a}{S \vdash_r lhs : a} \text{ (WFR-Lhs)}$$

$$\frac{S \vdash_l^{!tame} lhs : a \qquad \vdash_a a * \texttt{safe}}{S \vdash_r (a * \texttt{safe}) \& lhs : a * \texttt{safe}} \text{ (WFR-RefSafe)}$$

$$\frac{S \vdash_l^{!tame} lhs : a \qquad \vdash_a a * \texttt{seq}}{S \vdash_r (a * \texttt{seq}) \& lhs : a * \texttt{seq}} \text{ (WFR-RefSeq)}$$

$$\frac{S \vdash_l^{tame} lhs : a \qquad \vdash_a a * \texttt{tame}}{S \vdash_r (a * \texttt{tame}) \& lhs : a * \texttt{tame}} \text{ (WFR-RefTame)}$$

$$\frac{S \vdash_r rhs_1 : \texttt{int} \qquad S \vdash_r rhs_2 : \texttt{int}}{S \vdash_r rhs_1 + rhs_2 : \texttt{int}} \text{ (WFR-Add)}$$

$$\frac{S \vdash_r rhs_1 : p * q \qquad q \neq \texttt{safe} \qquad S \vdash_r rhs_2 : \texttt{int}}{S \vdash_r rhs_1 + rhs_2 : p * q} \text{ (WFR-AddPtr)}$$

$$\frac{S \vdash_r rhs : a \qquad a \Rightarrow a'}{S \vdash_r (a')rhs : a'} \text{ (WFR-Cast)}$$

$$\frac{\vdash_a a}{S \vdash_r sizeof(a) : \texttt{int}} \text{ (WFR-Size)}$$

$$\frac{S \vdash_r rhs : \texttt{int} \qquad \vdash_a p * q \qquad psize(p) > 0}{S \vdash_r (p * q) malloc\, rhs : p * q} \text{ (WFR-Alloc)}$$

Figure 9: Well-formed RHS.

$\boxed{S \vdash_c c}$

$$\frac{}{S \vdash_c skip} \text{ (WFC-Skip)}$$

$$\frac{S \vdash_c c_1 \qquad S \vdash_c c_2}{S \vdash_c c_1 \; ; \; c_2} \text{ (WFC-Seq)}$$

$$\frac{S \vdash_l lhs : a_l \qquad S \vdash_r rhs : a_r \qquad a_r \Rightarrow a_l}{S \vdash_c lhs \; = \; rhs} \text{ (WFC-Assign)}$$

Figure 10: Well-formed CMD.

4 Semantics

Annotation	USAGE
$d_{(b,e)}$	d with meta (b, e)
$d^{id}_{(b,e)}$	d with meta (b, e), id is the name of d' sub field
$s[id]_{off}$	the offset s' sub field id
$s[id]_t$	the type s' sub field id

Result:

$r ::=$		*results*
	ok	*Succ*
	l	*location*
	$(d_{(b,e)}, a)$	*data with meta*
	Abort	*Abort*
	$OutOfMem$	*OutOfMem*
$err ::=$		*errors*
	Abort	*Abort*
	$OutOfMem$	*OutOfMem*

Figure 11: Result.

dataCast:

from	*to*	*assertion*
$(d_{(b,e)}, p * q)$	$(d_{(b,e)}, \texttt{int})$	
$(d_{(b,e)}, \texttt{int})$	$(0_{(0,0)}, p * \texttt{safe})$	$d = 0$
$(d_{(b,e)}, \texttt{int})$	$(d_{(0,0)}, p * \texttt{seq})$	
$(d_{(b,e)}, \texttt{int})$	$(d_{(0,0)}, p * \texttt{tame})$	
$(d_{(b,e)}, p * \texttt{seq})$	$(d_{(b,e)}, p * \texttt{safe})$	
$(d_{(b,e)}, p * \texttt{safe})$	$(d_{(d,d+psize(p))}, p * \texttt{seq})$	$(v = 0) \vee (b \neq 0 \wedge b \leq d <= (e - psize(p)))$
$(d_{(b,e)}, \_)$	$(d_{(b,e)}, \_)$	

Figure 12: Data Cast.

Assertion:

$$
\begin{array}{rcl}
\text{assert } d_{(b,e)}\ a * \texttt{safe} & \triangleq & d \neq 0 \\
\text{assert } d_{(b,e)}\ a * \texttt{seq} & \triangleq & b \neq 0 \wedge b \leq d \wedge d + asize(a) <= e \\
\text{assert } d_{(b,e)}\ a * \texttt{tame} & \triangleq & b \neq 0 \wedge b \leq d \wedge d + asize(a) <= e \\
\text{assert } d^{id}_{(b,e)}\ s * \texttt{safe} & \triangleq & d \neq 0 \\
\text{assert } d^{id}_{(b,e)}\ s * \texttt{seq} & \triangleq & b \neq 0 \wedge b \leq d + s[id]_{off} + asize(s[id]_t) <= e \\
\text{assert } d^{id}_{(b,e)}\ s * \texttt{tame} & \triangleq & b \neq 0 \wedge b \leq d + s[id]_{off} + asize(s[id]_t) <= e
\end{array}
$$

Figure 13: Assertion.

$\boxed{E \vdash_l lhs \Rightarrow r : a}$

$$\frac{(v \mapsto (l, a)) \in E.S}{E \vdash_l v \Rightarrow l : a} \text{ (Ev-Var)}$$

$$\frac{E \vdash_l lhs \Rightarrow l : a * q \qquad \text{readMem } E.M\ l = \texttt{some}\ l'_{(b',e')} \qquad \text{assert } l'_{(b',e')}\ a * q}{E \vdash_l *lhs \Rightarrow l' : a} \text{ (Ev-Def)}$$

$$\frac{E \vdash_l lhs \Rightarrow e : a}{E \vdash_l *lhs \Rightarrow e : a'} \text{ (Ev-Def-ErrorProp)}$$

$$\frac{E \vdash_l lhs \Rightarrow l : a * q \qquad \text{readMem } E.M\ l = \texttt{some}\ l'_{(b',e')} \qquad \neg\text{assert } l'_{(b',e')}\ a * q}{E \vdash_l *lhs \Rightarrow Abort : a} \text{ (Ev-Def-Abort)}$$

$$\frac{E \vdash_l lhs \Rightarrow l : s * q \qquad \text{readMem } E.M\ l = \texttt{some}\ l'_{(b',e')} \qquad \text{assert } l'^{id}_{(b',e')}\ s * q}{E \vdash_l lhs \rightarrow_s id \Rightarrow l + s[id]_{off} : s[id]_t} \text{ (Ev-StructPos)}$$

$$\frac{E \vdash_l lhs \Rightarrow e : a}{E \vdash_l lhs \rightarrow_s id \Rightarrow e : a'} \text{ (Ev-StructPos-ErrorProp)}$$

$$\frac{E \vdash_l lhs \Rightarrow l : s * q \qquad \text{readMem } E.M\ l = \texttt{some}\ l'_{(b',e')} \qquad \neg\text{assert } l'^{id}_{(b',e')}\ s * q}{E \vdash_l lhs \rightarrow_s id \Rightarrow Abort : s[id]_t} \text{ (Ev-StructPos-Abort)}$$

$$\frac{\begin{array}{l}E \vdash_l lhs \Rightarrow l : n * q \qquad \text{tab } n = \texttt{some}\ s \\ \text{readMem } E.M\ l = \texttt{some}\ l'_{(b',e')} \qquad \text{assert } l'^{id}_{(b',e')}\ s * q\end{array}}{E \vdash_l lhs \rightarrow_n id \Rightarrow l + s[id]_{off} : s[id]_t} \text{ (Ev-NamePos)}$$

$$\frac{E \vdash_l lhs \Rightarrow e : a}{E \vdash_l lhs \rightarrow_n id \Rightarrow e : a'} \text{ (Ev-NamePos-ErrorProp)}$$

$$\frac{\begin{array}{l}E \vdash_l lhs \Rightarrow l : n * q \qquad \text{tab } n = \texttt{some}\ s \\ \text{readMem } E.M\ l = \texttt{some}\ l'_{(b',e')} \qquad \neg\text{assert } l'^{id}_{(b',e')}\ s * q\end{array}}{E \vdash_l lhs \rightarrow_n id \Rightarrow Abort : s[id]_t} \text{ (Ev-NamePos-Abort)}$$

Figure 14: Evaluation LHS.

$\boxed{E \vdash_r rhs \Rightarrow r : a \vdash_r E}$

$$\frac{}{E \vdash_r i \Rightarrow (i_{(0,0)}, \texttt{int}) : \texttt{int} \vdash_r E} \text{ (Ev-Const)}$$

$$\frac{E \vdash_l lhs \Rightarrow l : a \qquad \text{readMemMeta } E.M\ l = \texttt{some}\ d_{(b,e)}}{E \vdash_r lhs \Rightarrow d_{(b,e)}, E.TI(l) : a \vdash_r E} \text{ (Ev-Lhs)}$$

$$\frac{E \vdash_l lhs \Rightarrow e : a}{E \vdash_r lhs \Rightarrow e : a' \vdash_r E} \text{ (Ev-Lhs-ErrorProp)}$$

$$\frac{\vdash_a\ a * \texttt{safe} \quad E.S \vdash_l^{!tame} lhs : a \quad E \vdash_l lhs \Rightarrow l : a}{E \vdash_r (a * \texttt{safe}) \& lhs \Rightarrow (l_{(0,0)}, a * \texttt{safe}) : a * \texttt{safe} \vdash_r E} \text{ (Ev-RefSafe)}$$

$$\frac{\vdash_a\ a * \texttt{safe} \quad E.S \vdash_l^{!tame} lhs : a \quad E \vdash_l lhs \Rightarrow l : a}{E \vdash_r (a * \texttt{seq}) \& lhs \Rightarrow (l_{(l, l+asize(a))}, a * \texttt{seq}) : a * \texttt{seq} \vdash_r E} \text{ (Ev-RefSeq)}$$

$$\frac{\vdash_a\ a * \texttt{safe} \quad E.S \vdash_l^{tame} lhs : a \quad E \vdash_l lhs \Rightarrow l : a}{E \vdash_r (a * \texttt{tame}) \& lhs \Rightarrow (l_{(l, l+asize(a))}, a * \texttt{tame}) : a * \texttt{tame} \vdash_r E} \text{ (Ev-RefTame)}$$

$$\frac{E \vdash_l lhs \Rightarrow e : a}{E \vdash_r \& lhs \Rightarrow e : a' \vdash_r E} \text{ (Ev-Ref-ErrorProp)}$$

$$\frac{E \vdash_r rhs_1 \Rightarrow (d_{1(b_1,e_1)}, a_1) : \texttt{int} \vdash_r E' \qquad E' \vdash_r rhs_2 \Rightarrow (d_{2(b_2,e_2)}, a_2) : \texttt{int} \vdash_r E''}{E \vdash_r rhs_1 + rhs_2 \Rightarrow (d_1 + d_{2(0,0)}, \texttt{int}) : \texttt{int} \vdash_r E''} \text{ (Ev-Add)}$$

$$\frac{E \vdash_r rhs_1 \Rightarrow e : a \vdash_r E'}{E \vdash_r rhs_1 + rhs_2 \Rightarrow e : a' \vdash_r E'} \text{ (Ev-Add-ErrorProp1)}$$

$$\frac{E \vdash_r rhs_1 \Rightarrow (d_{1(b_1,e_1)}, a_1) : \texttt{int} \vdash_r E' \qquad E' \vdash_r rhs_2 \Rightarrow e : a \vdash_r E''}{E \vdash_r rhs_1 + rhs_2 \Rightarrow e : a' \vdash_r E''} \text{ (Ev-Add-ErrorProp2)}$$

$$\frac{E \vdash_r rhs_1 \Rightarrow (d_{1(b_1,e_1)}, a_1) : p * q \vdash_r E' \qquad E' \vdash_r rhs_2 \Rightarrow (d_{2(b_2,e_2)}, a_2) : \texttt{int} \vdash_r E''}{E \vdash_r rhs_1 + rhs_2 \Rightarrow (d_1 + d_2 * psize(p)_{(b_1,e_1)}, p * q) : p * q \vdash_r E''} \text{ (Ev-AddPtr)}$$

$$\frac{E \vdash_r rhs_1 \Rightarrow e : a \vdash_r E'}{E \vdash_r rhs_1 + rhs_2 \Rightarrow e : a' \vdash_r E'} \text{ (Ev-AddPtr-ErrorProp1)}$$

$$\frac{E \vdash_r rhs_1 \Rightarrow (d_{1(b_1,e_1)}, a_1) : p * q \vdash_r E' \qquad E' \vdash_r rhs_2 \Rightarrow e : a \vdash_r E''}{E \vdash_r rhs_1 + rhs_2 \Rightarrow e : e' \vdash_r E''} \text{ (Ev-AddPtr-ErrorProp2)}$$

Figure 15: Evaluation RHS.

$\boxed{E \vdash_r rhs \Rightarrow r : a \vdash_r E}$

$$\frac{E \vdash_r rhs \Rightarrow (d_{(b,e)}, a_0) : a \vdash_r E' \qquad \text{dataCast } d_{(b,e)}\ a\ a' = d'_{(b',e')}}{E \vdash_r (a')rhs \Rightarrow (d'_{(b',e')}, a_0) : a' \vdash_r E'} \text{ (Ev-Cast)}$$

$$\frac{E \vdash_r rhs \Rightarrow e : a \vdash_r E'}{E \vdash_r (a')rhs \Rightarrow e : a'' \vdash_r E'} \text{ (Ev-Cast-ErrorProp)}$$

$$\frac{E \vdash_r rhs \Rightarrow (d_{(b,e)}, a_0) : a \vdash_r E' \qquad \neg\text{dataCast } d_{(b,e)}\ a\ a' = d'_{(b',e')}}{E \vdash_r (a')rhs \Rightarrow Abort : a'' \vdash_r E'} \text{ (Ev-Cast-Abort)}$$

$$\frac{}{E \vdash_r sizeof(a) \Rightarrow (asize(a)_{(0,0)}, \texttt{int}) : \texttt{int} \vdash_r E} \text{ (Ev-Size)}$$

$$\frac{E \vdash_r rhs \Rightarrow (d_{(b,e)}, a) : \texttt{int} \vdash_r E' \qquad d \geq psize(p) \qquad \text{malloc } E'\ d = \texttt{some}\ (E'', l)}{E \vdash_r (p * \texttt{safe})malloc\ rhs \Rightarrow (l_{(0,0)}, p * \texttt{safe}) : p * \texttt{safe} \vdash_r (E''.M, E''.S, \text{updateTypeInfo } E''.TI\ l\ p\ d)} \text{ (Ev-AllocSafe)}$$

$$\frac{E \vdash_r rhs \Rightarrow (d_{(b,e)}, a) : \texttt{int} \vdash_r E' \qquad \text{malloc } E'\ d = \texttt{some}\ (E'', l)}{E \vdash_r (p * \texttt{seq})malloc\ rhs \Rightarrow (l_{(l,l+d)}, p * \texttt{seq}) : p * \texttt{seq} \vdash_r (E''.M, E''.S, \text{updateTypeInfo } E''.TI\ l\ p\ d)} \text{ (Ev-AllocSeq)}$$

$$\frac{E \vdash_r rhs \Rightarrow (d_{(b,e)}, a) : \texttt{int} \vdash_r E' \qquad \text{malloc } E'\ d = \texttt{some}\ (E'', l)}{E \vdash_r (p * \texttt{tame})malloc\ rhs \Rightarrow (l_{(l,l+d)}, p * \texttt{tame}) : p * \texttt{tame} \vdash_r (E''.M, E''.S, \text{updateTypeInfo } E''.TI\ l\ p\ d)} \text{ (Ev-AllocTame)}$$

$$\frac{E \vdash_r rhs \Rightarrow e : a \vdash_r E'}{E \vdash_r (p * q)malloc\ rhs \Rightarrow e : a' \vdash_r E'} \text{ (Ev-Alloc-ErrorProp)}$$

$$\frac{E \vdash_r rhs \Rightarrow (d_{(b,e)}, a) : \texttt{int} \vdash_r E' \qquad \text{malloc } E'\ d = \texttt{none}}{E \vdash_r (p * q)malloc\ rhs \Rightarrow OutofMem : a' \vdash_r (E'} \text{ (Ev-Alloc-OutofMem)}$$

$$\frac{E \vdash_r rhs \Rightarrow (d_{(b,e)}, a) : \texttt{int} \vdash_r E' \qquad d < psize(p)}{E \vdash_r (p * \texttt{safe})malloc\ rhs \Rightarrow Abort : a' \vdash_r (E')} \text{ (Ev-AllocSafe-Abort)}$$

Figure 16: Evaluation RHS - Con.

$\boxed{E \vdash_c c \Rightarrow r \vdash_c E}$

$$\frac{}{E \vdash_c \mathit{skip} \Rightarrow \mathit{ok} \vdash_c E} \text{ (Ev-Skip)}$$

$$\frac{E \vdash_c c_1 \Rightarrow \mathit{ok} \vdash_c E' \qquad E' \vdash_c c_2 \Rightarrow \mathit{ok} \vdash_c E''}{E' \vdash_c c_1 \ ; \ c_2 \Rightarrow \mathit{ok} \vdash_c E''} \text{ (Ev-Seq)}$$

$$\frac{E \vdash_c c_1 \Rightarrow e \vdash_c E'}{E' \vdash_c c_1 \ ; \ c_2 \Rightarrow e \vdash_c E'} \text{ (Ev-Seq-ErrorProp1)}$$

$$\frac{E \vdash_c c_1 \Rightarrow \mathit{ok} \vdash_c E' \qquad E' \vdash_c c_2 \Rightarrow e \vdash_c E''}{E' \vdash_c c_1 \ ; \ c_2 \Rightarrow e \vdash_c E''} \text{ (Ev-Seq-ErrorProp2)}$$

$$\frac{\begin{array}{l} E \vdash_l \mathit{lhs} \Rightarrow l : p * q \qquad E \vdash_r \mathit{rhs} \Rightarrow (d_{(b,e)}, \_) : a_r \vdash_r E' \\ \text{dataCast } d_{(b,e)}\ p * q\ a_r \qquad \text{writeMemMeta } E'.M\ \mathit{ld}_{(b,e)} = \texttt{some}\ M'' \end{array}}{E \vdash_c \mathit{lhs}\ =\ \mathit{rhs} \Rightarrow \mathit{ok} \vdash_c (M'', E'.S, E'.TI)} \text{ (Ev-Assign-Ptr)}$$

$$\frac{\begin{array}{l} E \vdash_l \mathit{lhs} \Rightarrow l : \texttt{int} \qquad E \vdash_r \mathit{rhs} \Rightarrow (d_{(b,e)}, \_) : a_r \vdash_r E' \\ \text{dataCast } d_{(b,e)}\ \texttt{int}\ a_r \qquad \text{writeMem } E'.M\ \mathit{ld} = \texttt{some}\ M'' \end{array}}{E \vdash_c \mathit{lhs}\ =\ \mathit{rhs} \Rightarrow \mathit{ok} \vdash_c (M'', E'.S, E'.TI)} \text{ (Ev-Assign-NPtr)}$$

$$\frac{E \vdash_l \mathit{lhs} \Rightarrow e : a}{E \vdash_c \mathit{lhs}\ =\ \mathit{rhs} \Rightarrow e \vdash_c E} \text{ (Ev-Assign-ErrorProp1)}$$

$$\frac{E \vdash_l \mathit{lhs} \Rightarrow l : a \qquad E \vdash_r \mathit{rhs} \Rightarrow e : a_r \vdash_r E'}{E \vdash_c \mathit{lhs}\ =\ \mathit{rhs} \Rightarrow e \vdash_c E'} \text{ (Ev-Assign-ErrorProp2)}$$

$$\frac{E \vdash_l \mathit{lhs} \Rightarrow l : p * q \qquad E \vdash_r \mathit{rhs} \Rightarrow (d_{(b,e)}, \_) : a_r \vdash_r E' \qquad \neg\text{dataCast } d_{(b,e)}\ p * q\ a_r}{E \vdash_c \mathit{lhs}\ =\ \mathit{rhs} \Rightarrow \mathit{Abort} \vdash_c E'} \text{ (Ev-Assign-Ptr-Abort)}$$

Figure 17: Evaluation CMD.

5 Properties

Lemma 5.1 (stack invariance)

1. *If* $\vdash_E E$, $E.S \vdash_r rhs : a$ *and* $E \vdash_r rhs \Rightarrow r : a \vdash_r E'$, *then* $E.S = E'.S$.
2. *If* $\vdash_E E$, $E.S \vdash_c c$ *and* $E \vdash_c c \Rightarrow r \vdash_c E'$, *then* $E.S = E'.S$.

Proof: Part 1 is by induction on $E \vdash_r rhs \Rightarrow r : a \vdash_r E'$, part 2 is by induction on $E \vdash_c c \Rightarrow r \vdash_c E'$. □

Lemma 5.2 (lhs inversion) *If* $\vdash_E E$ *and* $E \vdash_l lhs \Rightarrow l : a$, *then validMem* $E.M\ l$ *and* $l \neq 0 \wedge l \geq baseAddr \wedge l + asize(a) < maxAddr$.

Proof: By induction on $E \vdash_l lhs \Rightarrow l : a$. □

Lemma 5.3 (lhs ptr inversion) *If* $\vdash_E E$ *and* $E \vdash_l lhs \Rightarrow l : a$, *then*

1. *If* $a = p * \texttt{safe}$, *then* $E.TI(l) = p * \texttt{safe}$
2. *If* $a = p * \texttt{seq}$, *then* $E.TI(l) = p * \texttt{seq}$
3. *If* $a = p * \texttt{tame}$, *then* $\exists p'.E.TI(l) = p' * \texttt{tame}$

Lemma 5.4 (rhs ptr inversion) *If* $\vdash_E E$ *and* $E \vdash_r rhs \Rightarrow (d_{(b,e)}, a') : a \vdash_r E'$, *then*

1. *If* $a = p * \texttt{seq}$ *and* $a' = \texttt{int}$, *then* $b = e = 0$ *or* $d = b = 0$
2. *If* $a = p * \texttt{safe}$, *then* $a' \neq \texttt{int}$ *or* $a' = \texttt{int} \wedge d = b = 0$
3. *If* $a = p * \texttt{tame}$, $a' = p' * q'$ *and* $q' \neq \texttt{tame}$, *then* $b = e = 0$
4. *If* $a = p * \texttt{tame}$, $a' = \texttt{int}$, *then* $b = e = 0$
5. *If* $a = p * \texttt{safe}$, $a' = p' * \texttt{seq}$, *then* $d = 0$ *or* $b \neq 0 \wedge b \leq d < e - psize(p)$
6. *If* $a = p * \texttt{safe}$, $a' = p' * \texttt{tame}$, *then* $d = 0$.
7. *If* $a = p * \texttt{seq}$, $a' = p' * \texttt{tame}$, *then* $d = 0$ *or* $b = e = 0$.
8. *If* $a = p * \texttt{safe}$, $a' = p' * \texttt{seq}$, *then* $d = b = e = 0$ *or* $d = b \wedge e = b + psize(p)$.

Lemma 5.5 (rhs inversion) *If* $\vdash_E E$ *and* $E \vdash_r rhs \Rightarrow (d_{(b,e)}, a') : a \vdash_r E'$, *then* $E'.M\ ; E'.TI\ \vdash_D d_{(b,e)} : a'$.

Proof: By lhs ptr inversion 5.3, rhs ptr inversion 5.4. □

Theorem 5.1 (rhs well-formed environment invariance) *If* $\vdash_E E$, $E.S \vdash_r rhs : a$ *and* $E \vdash_r rhs \Rightarrow r : a \vdash_r E'$, *then* $\vdash_E E'$.

Proof: By induction on $E \vdash_r rhs \Rightarrow r : a \vdash_r E'$, other cases are trivial except Ev-AllocSafe, Ev-AllocSeq and Ev-AllocTame.

1. Ev-AllocSafe: $E \vdash_r rhs \Rightarrow (d_{(b,e)}, a) : \texttt{int} \vdash_r E'$ and malloc $E'\ d = \texttt{some}\ (E'', l)$.
   t.s. $WFEnv(E''.M, E''.S, \text{updateTypeInfo}\ E''.TI\ l\ p\ d)$. It is sufficient to show
   (a) $E''.M$; updateTypeInfo $E''.TI\ l\ p\ d \vdash_S E''.S$: By malloc inversion 2.5 and updateTypeInfo inversion 2.4.

(b) $\vdash_M E''.M$; updateTypeInfo $E''.TI\ l\ p\ d$: By definition, it is to show $\forall(l', d', b', e').$ readMemMeta $E''.M\ l' = d'_{(b',e')} \Rightarrow E''.M$;updateTypeInfo $E''.TI\ l\ p\ d\ \vdash_D d'_{(b',e')}$: (updateTypeInfo $E''.TI\ l\ p\ d)(l')$.

i. $l' \geq l+d \vee l' < l$: By updateTypeInfo inversion 2.4, $E''.TI(l') =$ (updateTypeInfo $E''.TI\ l\ p\ d)(l')$. By destruct $E''.TI(l')$, each case is by malloc inversion 2.5 and updateTypeInfo inversion 2.4.

ii. $l \leq l' < l + d$: By malloc inversion 2.5 and updateTypeInfo inversion 2.4.

2. Ev-AllocSeq,Ev-AllocTame: similar to Ev-AllocSafe.

□

Theorem 5.2 (cmd well-formed environment invariance) *If* $\vdash_E E$, $E.S \vdash_c c$ *and* $E \vdash_c c \Rightarrow r \vdash_c E'$, *then* $\vdash_E E'$.

Proof: By induction on $E \vdash_c c \Rightarrow r \vdash_c E'$,

1. Ev-Assign-Ptr: By rhs inversion 5.5.
2. others: immediate.

□

Theorem 5.3 (lhs progress) *If* $\vdash_E E$ *and* $E.S \vdash_l lhs : a$, *then* $\exists l.E \vdash_l lhs \Rightarrow l : a$ *or* $E \vdash_l lhs \Rightarrow Abort : a$.

Proof: By induction on $E.S \vdash_l lhs : a$,

1. WFL-Var: Immediate.
2. WFL-Def: $E.S \vdash_l lhs_0 : a_0 * q$ with $lhs = *lhs_0$ and $a = a_0$.
   t.s. $E.S \vdash_l *lhs_0 : a_0$.
   By IH,
   (a) $E \vdash_l lhs_0 \Rightarrow l : a_0$: By lhs inversion 5.2, we have readMemMeta $E.M\ l =$ `some` $l'_{(b',e')}$. The result follows by that assert $l'_{(b',e')}\ a_0 * q$ is decidable.
   (b) $E \vdash_l lhs_0 \Rightarrow Abort : a_0$: Immediate by Ev-Def-ErrorProp.
3. WFL-StructPos and WFL-NamePos: similar to Case WFL-Def.

□

Theorem 5.4 (rhs progress) *If* $\vdash_E E$ *and* $E.S \vdash_r rhs : a$, *then* $\exists(d_{(b,e)}, a'), E'.E \vdash_r rhs \Rightarrow (d_{(b,e)}, a') : a \vdash_r E'$ *or* $\exists E', a'.E \vdash_r rhs \Rightarrow OutofMem : a' \vdash_r E'$ *or* $\exists E, a'.E \vdash_r rhs \Rightarrow Abort : a' \vdash_r E'$.

Proof: By induction on $E.S \vdash_r rhs : a$,

1. WFR-Const: Immediate.
2. WFR-Lhs: $E.S \vdash_l lhs : a$ with $rhs = lhs$ and $a = a$. By lhs progress 5.3,
   (a) $E \vdash_l lhs \Rightarrow l : a$: By lhs inversion 5.2, we have readMemMeta $E.M\ l =$ `some` $l'_{(b',e')}$. The result follows by Ev-Lhs.
   (b) $E \vdash_l lhs \Rightarrow Abort : a$: Immediate by Ev-Lhs-ErrorProp.
3. WFR-Ref-Safe: $E \vdash_l^{!tame} lhs : a'$ with $rhs = \&lhs$ and $a = a' *$ `safe`. We have $E \vdash_l lhs : a'$ because $E \vdash_l^{!tame} lhs : a' \Rightarrow E \vdash_l lhs : a'$. By lhs progress 5.3.
4. WFR-Ref-Seq: similar to Ev-Ref-Safe.

5. WFR-Ref-Tame: similar to Ev-Ref-Safe, but by $E \vdash_l^{tame} lhs : a' \Rightarrow E \vdash_l lhs : a'$.
6. WFR-Add: $S \vdash_r rhs_1 : \texttt{int}$ and $S \vdash_r rhs_2 : \texttt{int}$ with $rhs = rhs_1 + rhs_2$ and $a = \texttt{int}$. By IH of rhs_1,
   (a) $E \vdash_r rhs_1 \Rightarrow (d_{(b,e)}, a') : a \vdash_r E'$: By rhs well-formed environment invariance 5.1, $\vdash_E E'$. By stack invariance 5.1, $E.S = E'.S$. The desired result is by IH of rhs_2.
   (b) else: By Ev-Add-ErrorProp1.
7. WFR-AddPtr: similar to Case WFR-Add.
8. WFR-Cast: $S \vdash_r rhs' : a'$ with $rhs = (a)rhs'$ and $a = a$. By IH,
   (a) $E \vdash_r rhs' \Rightarrow (d_{(b,e)}, a'') : a' \vdash_r E'$: If dataCast $d_{(b,e)}$ a' a holds, then the result is by Ev-Cast, else it is by Ev-Cast-Abort.
   (b) else: By Ev-Cast-ErrorProp.
9. WFR-Size: Immediate.
10. WFR-Alloc:$S \vdash_r rhs' : \texttt{int}$ with $rhs = (p * q)malloc\, rhs'$ and $a = p * q$. By IH,
    (a) $E \vdash_r rhs' \Rightarrow (d_{(b,e)}, a) : \texttt{int} \vdash_r E'$:
       i. malloc E' d = some (E'', l): If $q \neq \texttt{safe}$, then the result is by Ev-Alloc-Seq and Ev-Alloc-Tame. Otherwise, if d $\geq psize(p)$, the result is by Ev-Alloc-Safe, else by Ev-Alloc-SafeAbort.
       ii. malloc E' d = none:By Ev-Alloc-OutofMem.
    (b) else: By Ev-Alloc-ErrorProp.

□

Theorem 5.5 (cmd progress) *If $\vdash_E E$ and $E.S \vdash_c c$, then $\exists E'.E \vdash_c c \Rightarrow ok \vdash_c E'$ or $\exists E'.E \vdash_c c \Rightarrow OutofMem \vdash_c E'$ or $\exists E.E \vdash_c c \Rightarrow Abort \vdash_c E'$.*

Proof: By induction on $E.S \vdash_c c$,

1. WFC-Skip: Immediate.
2. WFC-Seq: $S \vdash_c c_1$ and $S \vdash_c c_2$ with $c = c_1 \ ; \ c_2$. By IH of c_1,
   (a) $E \vdash_c c_1 \Rightarrow ok \vdash_c E'$: By cmd well-formed environment invariance 5.2, $WFEnvE'$. By stack invariance 5.1, $E.S = E'.S$. The desired result is by IH of c_2.
   (b) else: By Ev-Seq-ErrorProp1.
3. WFC-Assign: $S \vdash_l lhs : a_l$, $S \vdash_r rhs : a_r$ and $a_r \Rightarrow a_l$ where $c = lhs \ = \ rhs$. By lhs progress 5.3,
   (a) $E \vdash_l lhs \Rightarrow l : a_l$: By rhs progress 5.4,
      i. $E \vdash_l rhs \Rightarrow (d_{(b,e)}, a'_r) : a_r E'$:
         A. $a = \texttt{int}$: By lhs inversion 5.2, writeMem $E'.M$ ld = some M''. In case, dataCast $d_{(b,e)}$ int a_r holds. The result is by Ev-Assign-NPtr.
         B. $a = p * q$: By lhs inversion 5.2, writeMemMeta $E'.M$ $ld(b, e)$ = some M''. In case, the result is by the decidability of dataCast $d_{(b,e)}$ $p * q$ a_r
      ii. else: By Ev-Assign-ErrorProp2.
   (b) else: By Ev-Assign-ErrorProp1.

□

A Notational Conventions

TEXT	USAGE
a	atomic type
b	base
c	command
d	value
e	end
err	error
E	Environment
i	int constant
id	identity
l	location
lhs	left hand side expression
rhs	right hand side expression
M	Memory
n	named struct
p	pointer type
q	qualifier
r	result
s	anonymous struct
S	Stack
t	type
TI	Type Information
v	variable