r	repository unique identifier
m	module name
pn	package name
dcl	name of derived class
$Method$, $meth$	method name
$Field$, f	field name
Var, var	term variable
mi	module instance identifier
$Pointer$, oid	object identifier
j, k, l	index

mf	::=			module file
	|	$\textbf{superpackage}\ mn\{\ \overline{\textbf{member}\ pn_j;}^{\,j}\ \overline{\textbf{import}\ mn_k;}^{\,k}\ \overline{\textbf{export}\ fqn_l;}^{\,l}\ \}$		def.
mn	::=			module name
	|	$core\_m$		core module
	|	m		standard module
fqn	::=			fully-qualified name
	|	$pn.dcl$		def.
SRC	::=			source files ($\overline{cld^{\mathsf{c}}}$)
	|	$\overline{cld^{\mathsf{c}}}$	M	def.
$\overline{cld^{\mathsf{c}}}$	::=			class def.'s, compile-time code
	|	$cld_1^{\mathsf{c}} .. cld_k^{\mathsf{c}}$	M	def.
	|	$cld^{\mathsf{c}} \# \overline{cld^{\mathsf{c}}}$	M	cons
cld^{c}	::=			class, compile-time code
	|	$pd\ am\ \textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def^{\mathsf{c}}}\}$		def.
pd	::=			package declaration (pn)
	|	$\textbf{package}\ pn;$	M	def.
am	::=			access modifier
	|			default
	|	**public**		public
$C,\ cl$	::=			class name
	|	`Object`		top class
	|	fqn		fully qualified name
$\overline{fd}$	::=			field declarations (fd list)
	|	$[\,]$	M	empty
	|	$fd_1 .. fd_k$	M	def.
fd	::=			field declaration
	|	$cl\,f;$		def.
$\overline{meth\_def^{\mathsf{c}}}$	::=			method def.'s, compile-time c
	|	$meth\_def_1^{\mathsf{c}} .. meth\_def_k^{\mathsf{c}}$	M	def.

$meth\_def^{\mathsf{c}}$	::=			method def., compile-time code
	|	$meth\_sig\{meth\_body^{\mathsf{c}}\}$		def.
$meth\_sig$	::=			method signature
	|	$cl\ meth(\overline{vd})$		def.
$\overline{vd}$	::=			variable declarations (vd list)
	|	$vd_1 .. vd_k$	M	def.
vd	::=			variable declaration
	|	$cl\ var$		def.
$meth\_body^{\mathsf{c}}$	::=			method body, compile-time code
	|	$s_1^{\mathsf{c}} .. s_k^{\mathsf{c}}$ **return** y;		def.
s^{c}	::=			statement, compile-time code
	|	$\{\ \overline{s_k^{\mathsf{c}}}^{\,k}\ \}$		block
	|	$var = x;$		variable assignment
	|	$var = x.f;$		field read
	|	$x.f = y;$		field write
	|	**if** $(x == y) s_1^{\mathsf{c}}$ **else** s_2^{c}		conditional branch
	|	$var =$ **new** $cl();$		object construction
	|	$var = x.meth(\overline{y});$		method call
$TVar,\ x,\ y$	::=			term variable
	|	var		normal variable
	|	**this**		ref. to current object
$\overline{x},\ \overline{y}$	::=			term variables (x list)
	|	$x_1 .. x_k$	M	def
P	::=			program
	|	(RC, MH)		def.
RC	::=			repository context ($rn \rightharpoonup R$)
	|	$[\,]$	M	empty repository context
	|	$RC[rn \mapsto R]$	M	rn maps to R in RC
rn	::=			repository name
	|	$bootstrap\_r$		bootstrap
	|	r		standard
R	::=			repository
	|	**bootstrap repository** $\{\overline{md^{\mathsf{c}}}; \phi\}$		bootstrap
	|	**repository** r **child of** $rn\{\overline{md^{\mathsf{c}}}; \phi\}$		standard
$\overline{md^{\mathsf{c}}}$	::=			module def.'s, compile-time code (md^{c} list)
	|	$md_1^{\mathsf{c}} .. md_k^{\mathsf{c}}$	M	def.
	|	$md^{\mathsf{c}} \# \overline{md^{\mathsf{c}}}$	M	cons
md^{c}	::=			module definition

	|	**module** $mn\{\overline{cld^{\mathsf{c}}}\ \overline{m}\ \overline{fqn}\}$		def.
$\overline{m}$	::=			module names (m list)
	|	$m_1 \mathinner{..} m_k$	M	def.
$\overline{fqn}$	::=			fully-qualified names (fqn list)
	|	$fqn_1 \mathinner{..} fqn_k$	M	def.
ϕ	::=			repository's cache ($md^{\mathsf{c}} \rightharpoonup mi$)
	|	$[\,]$	M	empty repository's cache
	|	$\phi[md^{\mathsf{c}} \mapsto mi]$	M	map md^{c} to mi in ϕ
	|	$\phi \setminus md^{\mathsf{c}}$	M	remove mapping for md^{c}
MH	::=			module hierarchy ($mi \rightharpoonup md \times \overline{mi}$)
	|	$[\,]$	M	empty module hierarchy
	|	$[mi \mapsto (md, \overline{mi})]$	M	maps mi to the given def. and imports
	|	$MH_1 \mathinner{..} MH_k$	M	composes many
md	::=			module instance
	|	**module** $mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}$		def.
$\overline{cld}$	::=			class def.'s (cld list)
	|	$[\,]$	M	empty
	|	$cld_1 \mathinner{..} cld_k$	M	def.
cld	::=			class def.
	|	$pd\ am$ **class** dcl **extends** $cl\{\overline{fd}\ \overline{meth\_def}\}$		def.
$\overline{meth\_def}$	::=			method def.'s ($meth\_def$ list)
	|	$[\,]$	M	empty
	|	$meth\_def_1 \mathinner{..} meth\_def_k$	M	def.
$meth\_def$	::=			method definition
	|	$meth\_sig\{meth\_body\}$		def.
$meth\_body$	::=			method body
	|	$s_1 \mathinner{..} s_k$ **return** y;		def.
s	::=			statement
	|	$\{\ \overline{s_k}^{\,k}\ \}$		block
	|	$var = x$;		variable assignment
	|	$var = x.f$;		field read
	|	$x.f = y$;		field write
	|	**if** $(x == y) s$ **else** s'		conditional branch
	|	$var =$ **new**$_{ctx}\, cl()$;		object creation
	|	$var = x.meth(\overline{y})$;		method call
ctx	::=			context
	|	$mi.pn$		def.
$\overline{mi}$	::=			module instance identifiers (mi list)

	| $[\,]$	M	empty
	| $mi_1 \,..\, mi_k$	M	def.
md^{c}_{opt}	::=		module def., compile-time code option (md^{c} option)
	| **null**	M	none
	| md^{c}	M	some
$\overline{f}$	::=		fields (f list)
	| $[\,]$	M	empty
	| $f_1 \,..\, f_k$	M	def.
	| $\overline{f}; \overline{f}'$	M	append
$\overline{f}_{\,opt}$	::=		fields option ($\overline{f}$ option)
	| **null**	M	none
	| $\overline{f}$	M	some
$\overline{meth}$	::=		method names ($meth$ list)
	| $[\,]$	M	empty
	| $meth_1 \,..\, meth_k$	M	def.
	| $\overline{meth}; \overline{meth}'$	M	append
$meth\_def_{opt}$	::=		method def. option ($meth\_def$ option)
	| **null**	M	none
	| $meth\_def$	M	some
$ctxmeth\_def_{opt}$	::=		method def. in context option (($ctx \times meth\_def$) option)
	| **null**	M	none
	| $(ctx, meth\_def)$	M	some
cld_{opt}	::=		class def. option (cld list)
	| **null**	M	none
	| cld	M	some
$ctxcld$	::=		class def. in context ($ctx \times cld$)
	| (ctx, cld)	M	def.
$\overline{ctxcld}$	::=		class def.'s in context ($ctxcld$ list)
	| $[\,]$	M	empty
	| $ctxcld_1 \,..\, ctxcld_k$	M	def.
	| $\overline{ctxcld}@[ctxcld]$	M	rev cons
$ctxcld_{opt}$	::=		class def. lookup result ($ctxcld$ option)
	| **null**	M	none
	| $ctxcld$	M	some
$\overline{ctxcld}_{opt}$	::=		class def.'s lookup result ($\overline{ctxcld}$ option)
	| **null**	M	none
	| $\overline{ctxcld}$	M	some
$\overline{pn}$	::=		package names (pn list)
	| $pn_1 \,..\, pn_k$	M	def.

mi_{opt}	::=			module instance option (mi option)
	|	**null**	M	none
	|	mi	M	some
	|	$\phi(md^{\mathsf{c}})$	M	module instance lookup
R_{opt}	::=			repository option (R option)
	|	**null**	M	none
	|	R	M	some
	|	$RC(rn)$	M	repository lookup
mhv	::=			module hierarchy value ($md \times \overline{mi}$)
	|	$(md, \overline{mi})$	M	def.
mhv_{opt}	::=			module hierarchy value option (mhv option)
	|	**null**	M	none
	|	mhv	M	some
	|	$MH(mi)$	M	lookup
$rnmd^{\mathsf{c}}_{opt}$	::=			module def., compile-time code lookup value ($rnmd^{\mathsf{c}}$ option)
	|	**null**	M	none
	|	(rn, md^{c})	M	some
$Type,\ \tau$	::=			type
	|	`Object`		supertype of all types
	|	$ctx.dcl$		class identifier
τ_{opt}	::=			result of type lookup (τ option)
	|	**null**	M	none
	|	τ	M	some
	|	$\Gamma(x)$	M	static type lookup
	|	$H(oid)$	M	dynamic type lookup
$\tau^{\perp}_{opt}$	::=			result of type lookup that can abort
	|	τ_{opt}		result of type lookup
	|	$\perp$		failed to find a type
$\overline{\tau}$	::=			types (τ list)
	|	$\tau_1 .. \tau_k$	M	def.
π	::=			method type
	|	$\overline{\tau} \rightarrow \tau$		def.
Γ	::=			type environment ($x \rightharpoonup \tau$)
	|	$[x_1 \mapsto \tau_1 .. x_k \mapsto \tau_k]$	M	type mappings
	|	$\Gamma[x \mapsto \tau]$	M	Γ with $x \mapsto \tau$
θ	::=			variable mapping ($x \rightharpoonup x$)
	|	$[x_1 \mapsto y_1 .. x_k \mapsto y_k]$	M	variable mappings
	|	$\theta[x \mapsto y]$	M	θ with $x \mapsto y$
$Val,\ v,\ w$	::=			value

	| **null**		null value
	| *oid*		object identifier
v_{opt}	::=		result of value lookup (v option)
	| v	M	some
	| $L(x)$	M	lookup value of local variable
	| $H(oid, f)$	M	lookup value of field
L	::=		variable state ($x \rightharpoonup v$)
	| $[\,]$	M	empty variable state
	| $L[x \mapsto v]$	M	L with $x \mapsto v$
	| $L[x_1 \mapsto v_1 \,..\, x_k \mapsto v_k]$	M	L with many mappings
H	::=		heap ($oid \rightharpoonup (\tau \times (f \rightharpoonup v))$)
	| $[\,]$	M	empty heap
	| $H[oid \mapsto (\tau, f_1 \mapsto v_1 \,..\, f_k \mapsto v_k)]$	M	H with new oid of type τ
	| $H[(oid, f) \mapsto v]$	M	H with $(oid,f) \mapsto v$
config	::=		configuration
	| $(P, L, H, \overline{s_k}^{\,k})$		normal configuration
	| $(P, L, H, Exception)$		exception occurred
Exception	::=		exception
	| **NPE**		null-pointer exception
a	::=		administrator action
	| $rn.\,\mathbf{install}\,(md^{c})$;		install
	| $rn.\,\mathbf{uninstall}\,(m)$;		uninstall
	| $rn.\,\mathbf{initialise}\,(m)$;		initialise
ia	::=		internal action
	| $rn.\,\mathbf{install}\,(md^{c})$		install
	| $rn.\,\mathbf{uninstall}\,(m)$		uninstall
	| $mi = rn.get\_instance(m)$		initialise
$\overline{ia}$	::=		internal actions
	| $ia_1 \,..\, ia_k$	M	def.
nn	::=		natural number (nat)
	| 0	M	zero
	| 1	M	one
	| $nn + nn'$	M	plus
	| $nn - nn'$	M	minus
	| $\mathbf{size}\,(\mathbf{dom}\,RC)$	M	size of domain of RC

$\boxed{R\_name(R) = rn}$ – extract repository's name

R_NAME_BOOTSTRAP

$$\overline{R\_name(\mathbf{bootstrap\ repository}\ \{\overline{md^{c}}; \phi\}) = bootstrap\_r}$$

R_NAME_STANDARD

$$\overline{R\_name(\mathbf{repository}\ r\ \mathbf{child\ of}\ rn\{\overline{md^{c}}; \phi\}) = r}$$

$\boxed{R\_body(R) = (\overline{md^{c}}, \phi)}$ – extract repository's contents

R_BODY_BOOTSTRAP

$$\frac{}{R\_body(\textbf{bootstrap repository}\ \{\overline{md^c};\phi\}) = (\overline{md^c},\phi)}$$

R_BODY_STANDARD

$$\frac{}{R\_body(\textbf{repository}\ r\ \textbf{child of}\ rn\{\overline{md^c};\phi\}) = (\overline{md^c},\phi)}$$

$\boxed{R\_update(R,\overline{md^c},\phi) = R'}$ – update a repository with given contents

R_UPDATE_BOOTSTRAP

$$\frac{}{R\_update(\textbf{bootstrap repository}\ \{\overline{md^c}_1;\phi_1\},\overline{md^c}_2,\phi_2) = \textbf{bootstrap repository}\ \{\overline{md^c}_2;\phi_2\}}$$

R_UPDATE_STANDARD

$$\frac{}{R\_update(\textbf{repository}\ r\ \textbf{child of}\ rn\{\overline{md^c}_1;\phi_1\},\overline{md^c}_2,\phi_2) = \textbf{repository}\ r\ \textbf{child of}\ rn\{\overline{md^c}_2;\phi_2\}}$$

$\boxed{mds\_rm(\overline{md^c}_1, md^c) = \overline{md^c}_2}$ – remove a module def. from a list

MDS_RM_EMPTY

$$\frac{}{mds\_rm(,md^c) =}$$

MDS_RM_CONS_TRUE

$$\frac{1.\ mds\_rm(\overline{md^c}_1, md^c) = \overline{md^c}_2}{mds\_rm(md^c\#\overline{md^c}_1, md^c) = \overline{md^c}_2}$$

MDS_RM_CONS_FALSE

$$\frac{\begin{array}{l}1.\ md_1^c \neq md^c\\ 2.\ mds\_rm(\overline{md^c}_1, md^c) = \overline{md^c}_2\end{array}}{mds\_rm(md_1^c\#\overline{md^c}_1, md^c) = md_1^c\#\overline{md^c}_2}$$

$\boxed{md\_name(md^c) = mn}$ – get name of module def.

MD_NAME

$$\frac{1.\ md^c = \textbf{module}\ mn\{\overline{cld^c}\ \overline{m}\ \overline{fqn}\}}{md\_name(md^c) = mn}$$

$\boxed{full\_name(cld) = fqn}$ – extract the full name from a class

FULL_NAME

$$\frac{}{full\_name(\textbf{package}\ pn;\ am\ \textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}) = pn.dcl}$$

$\boxed{package\_name(cld) = pn}$ – extract the package name of a class

PACKAGE_NAME

$$\frac{}{package\_name(\textbf{package}\ pn;\ am\ \textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}) = pn}$$

$\boxed{class\_name(cld) = dcl}$ – extract the class name from a class

CLASS_NAME

$$\frac{}{class\_name(pd\ am\ \textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}) = dcl}$$

$\boxed{superclass\_name(cld) = cl}$ – extract the superclass name from a class

SUPERCLASS_NAME

$$\frac{}{superclass\_name(pd\ am\ \textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}) = cl}$$

$\boxed{class\_fields(cld) = \overline{fd}}$ – extract class fields from a class

CLASS_FIELDS

$$\frac{}{class\_fields(pd\ am\ \textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}) = \overline{fd}}$$

$\boxed{class\_methods(cld) = \overline{meth\_def}}$ – extract class methods from a class

CLASS_METHODS

$$\frac{}{class\_methods(pd\ am\ \textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}) = \overline{meth\_def}}$$

$\boxed{method\_name(meth\_def) = meth}$ – extract the method name from a method definition

METHOD_NAME

$$\frac{}{method\_name(cl\ meth(\overline{vd})\{meth\_body\}) = meth}$$

$\boxed{distinct\_fqns(\overline{cld})}$ – fully-qualified names are distinct

DF_DEF

$$\frac{\begin{array}{l}1.\ \overline{full\_name(cld_k) = fqn_k}^{\,k} \\ 2.\ \textbf{distinct}\ (\overline{fqn_k}^{\,k})\end{array}}{distinct\_fqns(\overline{cld_k}^{\,k})}$$

$\boxed{find\_md\_in\_mds(\overline{md^c}, mn) = md^c_{opt}}$ – module definition lookup in a list

FMIM_EMPTY

$$\frac{}{find\_md\_in\_mds(, mn) = \textbf{null}}$$

FMIM_CONS_TRUE

$$\frac{1.\ md^c = \textbf{module}\ mn\{\overline{cld^c}\ \overline{m}\ \overline{fqn}\}}{find\_md\_in\_mds(md^c\ md^c_2\ ..\ md^c_k, mn) = md^c}$$

FMIM_CONS_FALSE

$$\frac{\begin{array}{l}1.\ md^c = \textbf{module}\ mn'\{\overline{cld^c}\ \overline{m}\ \overline{fqn}\} \\ 2.\ mn \neq mn' \\ 3.\ find\_md\_in\_mds(md^c_2\ ..\ md^c_k, mn) = md^c_{opt}\end{array}}{find\_md\_in\_mds(md^c\ md^c_2\ ..\ md^c_k, mn) = md^c_{opt}}$$

$\boxed{find\_md\_rec(RC, rn_1, mn, nn) = rnmd^c_{opt}}$ – module def. lookup (recursive part)

FMR_NULL

$$\frac{1.\ RC(rn) = \textbf{null}}{find\_md\_rec(RC, rn, mn, nn) = \textbf{null}}$$

FMR_BOOTSTRAP_NULL

$$\frac{\begin{array}{l}1.\ RC(rn) = \textbf{bootstrap repository}\ \{\overline{md^c}; \phi\} \\ 2.\ find\_md\_in\_mds(\overline{md^c}, mn) = \textbf{null}\end{array}}{find\_md\_rec(RC, rn, mn, nn) = \textbf{null}}$$

FMR_BOOTSTRAP

$$\frac{\begin{array}{l}1.\ RC(rn) = \textbf{bootstrap repository}\ \{\overline{md^c}; \phi\} \\ 2.\ find\_md\_in\_mds(\overline{md^c}, mn) = md^c\end{array}}{find\_md\_rec(RC, rn, mn, nn) = (rn, md^c)}$$

FMR_STANDARD_FAIL

$$\frac{\begin{array}{l}1.\ RC(rn_1) = \textbf{repository}\ r\ \textbf{child of}\ rn_2\{\overline{md^c}; \phi\} \\ 2.\ \textbf{size}\ (\textbf{dom}\ RC) \leq nn\end{array}}{find\_md\_rec(RC, rn_1, mn, nn) = \textbf{null}}$$

FMR_STANDARD_REC

$$\frac{\begin{array}{l}1.\ RC(rn_1) = \textbf{repository}\ r\ \textbf{child of}\ rn_2\{\overline{md^c}; \phi\} \\ 2.\ \textbf{size}\ (\textbf{dom}\ RC) > nn \\ 3.\ find\_md\_rec(RC, rn_2, mn, nn+1) = (rn_3, md^c)\end{array}}{find\_md\_rec(RC, rn_1, mn, nn) = (rn_3, md^c)}$$

FMR_STANDARD_SELF

$$\frac{\begin{array}{l}1.\ RC(rn_1) = \textbf{repository}\ r\ \textbf{child of}\ rn_2\{\overline{md^c}; \phi\} \\ 2.\ \textbf{size}\ (\textbf{dom}\ RC) > nn \\ 3.\ find\_md\_rec(RC, rn_2, mn, nn+1) = \textbf{null} \\ 4.\ find\_md\_in\_mds(\overline{md^c}, mn) = md^c\end{array}}{find\_md\_rec(RC, rn_1, mn, nn) = (rn_1, md^c)}$$

FMR_STANDARD_NULL

$$\frac{\begin{array}{l}1.\ RC(rn_1) = \textbf{repository}\ r\ \textbf{child of}\ rn_2\{\overline{md^c}; \phi\} \\ 2.\ \textbf{size}\ (\textbf{dom}\ RC) > nn \\ 3.\ find\_md\_rec(RC, rn_2, mn, nn+1) = \textbf{null} \\ 4.\ find\_md\_in\_mds(\overline{md^c}, mn) = \textbf{null}\end{array}}{find\_md\_rec(RC, rn_1, mn, nn) = \textbf{null}}$$

$\boxed{find\_md(RC, rn, mn) = rnmd^{c}_{opt}}$ – module def. lookup

FM_DEF

$$\frac{1.\ find\_md\_rec(RC, rn, mn, 0) = rnmd^{c}_{opt}}{find\_md(RC, rn, mn) = rnmd^{c}_{opt}}$$

$\boxed{find\_cld\_in\_module(\overline{cld}, fqn) = cld_{opt}}$ – class lookup in an import

FCIM_EMPTY

$$\frac{}{find\_cld\_in\_module([\,], fqn) = \textbf{null}}$$

FCIM_NULL

$$\frac{1.\ \neg distinct\_fqns(cld\ cld_2\ ..\ cld_k)}{find\_cld\_in\_module(cld\ cld_2\ ..\ cld_k, fqn) = \textbf{null}}$$

FCIM_CONS_TRUE

$$\frac{\begin{array}{l}1.\ distinct\_fqns(cld\ cld_2\ ..\ cld_k)\\ 2.\ cld = \textbf{package}\ pn;\ \textbf{public class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}\end{array}}{find\_cld\_in\_module(cld\ cld_2\ ..\ cld_k, pn.dcl) = cld}$$

FCIM_CONS_FALSE

$$\frac{\begin{array}{l}1.\ distinct\_fqns(cld\ cld_2\ ..\ cld_k)\\ 2.\ cld = \textbf{package}\ pn';\ am\ \textbf{class}\ dcl'\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}\\ 3.\ pn \neq pn' \ \vee\ am \neq \textbf{public} \ \vee\ dcl \neq dcl'\\ 4.\ find\_cld\_in\_module(cld_2\ ..\ cld_k, pn.dcl) = cld_{opt}\end{array}}{find\_cld\_in\_module(cld\ cld_2\ ..\ cld_k, pn.dcl) = cld_{opt}}$$

$\boxed{find\_cld\_in\_core(P, fqn) = ctxcld_{opt}}$ – class lookup in the core library module

FCIC_NO_REP_EX

$$\frac{1.\ RC(bootstrap\_r) = \textbf{null}}{find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}}$$

FCIC_NOT_BOOTSTRAP_EX

$$\frac{1.\ RC(bootstrap\_r) = \textbf{repository}\ r\ \textbf{child of}\ rn\{\overline{md^{c}}; \phi\}}{find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}}$$

FCIC_NO_CORE_EX

$$\frac{\begin{array}{l}1.\ RC(bootstrap\_r) = \textbf{bootstrap repository}\ \{\overline{md^{c}}; \phi\}\\ 2.\ find\_md\_in\_mds(\overline{md^{c}}, core\_m) = \textbf{null}\end{array}}{find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}}$$

FCIC_NO_CORE_MI_EX

$$\frac{\begin{array}{l}1.\ RC(bootstrap\_r) = \textbf{bootstrap repository}\ \{\overline{md^{c}}; \phi\}\\ 2.\ find\_md\_in\_mds(\overline{md^{c}}, core\_m) = md^{c}\\ 3.\ \phi(md^{c}) = \textbf{null}\end{array}}{find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}}$$

FCIC_NO_MDMIS_EX

$$\frac{\begin{array}{l}1.\ RC(bootstrap\_r) = \textbf{bootstrap repository}\ \{\overline{md^{c}}; \phi\}\\ 2.\ find\_md\_in\_mds(\overline{md^{c}}, core\_m) = md^{c}\\ 3.\ \phi(md^{c}) = mi \quad 4.\ MH(mi) = \textbf{null}\end{array}}{find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}}$$

FCIC_FALSE

$$\frac{\begin{array}{l}1.\ RC(bootstrap\_r) = \textbf{bootstrap repository}\ \{\overline{md^{c}}; \phi\}\\ 2.\ find\_md\_in\_mds(\overline{md^{c}}, core\_m) = md^{c}\\ 3.\ \phi(md^{c}) = mi\\ 4.\ MH(mi) = (\textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}, \overline{mi})\\ 5.\ find\_cld\_in\_module(\overline{cld}, fqn) = \textbf{null}\end{array}}{find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}}$$

FCIC_TRUE

$$\frac{\begin{array}{l}1.\ RC(bootstrap\_r) = \textbf{bootstrap repository}\ \{\overline{md^c}; \phi\} \\ 2.\ find\_md\_in\_mds(\overline{md^c}, core\_m) = md^c \\ 3.\ \phi(md^c) = mi \\ 4.\ MH(mi) = (\textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}, \overline{mi}) \\ 5.\ find\_cld\_in\_module(\overline{cld}, fqn) = cld \\ 6.\ package\_name(cld) = pn\end{array}}{find\_cld\_in\_core((RC, MH), fqn) = (mi.pn, cld)}$$

$\boxed{(MH, \overline{mi}, nn) \in \textbf{reachable}}$ – there are nn module instances reachable from $\overline{mi}$ in MH

REACHABLE_EMPTY

$$\frac{}{(MH, [\,], 0) \in \textbf{reachable}}$$

REACHABLE_CONS

$$\frac{\begin{array}{l}1.\ MH(mi) = (md, \overline{mi}) \\ 2.\ (MH, \overline{mi}, nn') \in \textbf{reachable} \\ 3.\ (MH, mi_2\ ..\ mi_k, nn) \in \textbf{reachable}\end{array}}{(MH, mi\ mi_2\ ..\ mi_k, nn' + nn + 1) \in \textbf{reachable}}$$

$\boxed{acyclic\_mhMH}$ – a module hierarchy is acyclic

AM_DEF

$$\frac{\begin{array}{l}1.\ \textbf{finite}\,(\textbf{dom}\,(MH)) \\ 2.\ \forall \overline{mi}.\overline{mi}\ \subseteq\ \textbf{dom}\,(MH) \longrightarrow \big(\exists nn.(MH, \overline{mi}, nn) \in \textbf{reachable}\big) \\ 3.\ \forall mi \in \textbf{dom}\,(MH).\exists md\ \overline{mi}.MH(mi) = (md, \overline{mi})\ \wedge\ \overline{mi}\ \subseteq\ \textbf{dom}\,(MH)\end{array}}{acyclic\_mhMH}$$

$\boxed{find\_cld\_in\_imports(MH, \overline{mi}, fqn) = ctxcld_{opt}}$ – class lookup in imports

FCII_EMPTY

$$\frac{}{find\_cld\_in\_imports(MH, [\,], fqn) = \textbf{null}}$$

FCII_NULL

$$\frac{1.\ \neg\,(acyclic\_mhMH\ \wedge\ mi \in \textbf{dom}\,(MH)\ \wedge\ mi_2\ ..\ mi_k\ \subseteq\ \textbf{dom}\,(MH))}{find\_cld\_in\_imports(MH, mi\ mi_2\ ..\ mi_k, fqn) = \textbf{null}}$$

FCII_SKIP

$$\frac{\begin{array}{l}1.\ acyclic\_mhMH\ \wedge\ MH(mi) = (md, \overline{mi})\ \wedge\ mi_2\ ..\ mi_k\ \subseteq\ \textbf{dom}\,(MH) \\ 2.\ md = \textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}\ \wedge\ fqn \notin \overline{fqn} \\ 3.\ find\_cld\_in\_imports(MH, mi_2\ ..\ mi_k, fqn) = ctxcld_{opt}\end{array}}{find\_cld\_in\_imports(MH, mi\ mi_2\ ..\ mi_k, fqn) = ctxcld_{opt}}$$

FCII_REC

$$\frac{\begin{array}{l}1.\ acyclic\_mhMH\ \wedge\ MH(mi) = (md, \overline{mi})\ \wedge\ mi_2\ ..\ mi_k\ \subseteq\ \textbf{dom}\,(MH) \\ 2.\ md = \textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}\ \wedge\ fqn \in \overline{fqn} \\ 3.\ find\_cld\_in\_imports(MH, \overline{mi}, fqn) = ctxcld\end{array}}{find\_cld\_in\_imports(MH, mi\ mi_2\ ..\ mi_k, fqn) = ctxcld}$$

FCII_SELF

$$\frac{\begin{array}{l}1.\ acyclic\_mhMH\ \wedge\ MH(mi) = (md, \overline{mi})\ \wedge\ mi_2\ ..\ mi_k\ \subseteq\ \textbf{dom}\,(MH) \\ 2.\ md = \textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}\ \wedge\ fqn \in \overline{fqn} \\ 3.\ find\_cld\_in\_imports(MH, \overline{mi}, fqn) = \textbf{null} \\ 4.\ find\_cld\_in\_module(\overline{cld}, fqn) = cld\ \wedge\ package\_name(cld) = pn\end{array}}{find\_cld\_in\_imports(MH, mi\ mi_2\ ..\ mi_k, fqn) = (mi.pn, cld)}$$

FCII_NEXT

$$\frac{\begin{array}{l}1.\ acyclic\_mhMH\ \wedge\ MH(mi) = (md, \overline{mi})\ \wedge\ mi_2\ ..\ mi_k\ \subseteq\ \textbf{dom}\,(MH) \\ 2.\ md = \textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}\ \wedge\ fqn \in \overline{fqn} \\ 3.\ find\_cld\_in\_imports(MH, \overline{mi}, fqn) = \textbf{null} \\ 4.\ find\_cld\_in\_module(\overline{cld}, fqn) = \textbf{null} \\ 5.\ find\_cld\_in\_imports(MH, mi_2\ ..\ mi_k, fqn) = ctxcld_{opt}\end{array}}{find\_cld\_in\_imports(MH, mi\ mi_2\ ..\ mi_k, fqn) = ctxcld_{opt}}$$

$\boxed{find\_cld\_in\_self(\overline{cld}, pn, fqn) = cld_{opt}}$ – class lookup in the same module

FCIS_EMPTY

$$\frac{}{find\_cld\_in\_self([\,], pn, fqn) = \textbf{null}}$$

FCIS_NULL

$$\frac{1.\ \neg distinct\_fqns(cld\ cld_2\ ..\ cld_k)}{find\_cld\_in\_self(cld\ cld_2\ ..\ cld_k, pn, fqn) = \textbf{null}}$$

FCIS_CONS_TRUE

$$\frac{\begin{array}{l}1.\ distinct\_fqns(cld\ cld_2\ ..\ cld_k)\\ 2.\ cld = \textbf{package}\ pn';\ am\ \textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}\\ 3.\ pn = pn'\ \vee\ am = \textbf{public}\end{array}}{find\_cld\_in\_self(cld\ cld_2\ ..\ cld_k, pn, pn'.dcl) = cld}$$

FCIS_CONS_FALSE

$$\frac{\begin{array}{l}1.\ distinct\_fqns(cld\ cld_2\ ..\ cld_k)\\ 2.\ cld = \textbf{package}\ pn'';\ am\ \textbf{class}\ dcl'\ \textbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}\\ 3.\ (pn \neq pn'\ \wedge\ am \neq \textbf{public})\ \vee\ pn' \neq pn''\ \vee\ dcl \neq dcl'\\ 4.\ find\_cld\_in\_self(cld_2\ ..\ cld_k, pn, pn'.dcl) = cld_{opt}\end{array}}{find\_cld\_in\_self(cld\ cld_2\ ..\ cld_k, pn, pn'.dcl) = cld_{opt}}$$

$\boxed{find\_cld(P, ctx, fqn) = ctxcld_{opt}}$ – class lookup

FC_CORE

$$\frac{1.\ find\_cld\_in\_core(P, fqn) = ctxcld}{find\_cld(P, ctx, fqn) = ctxcld}$$

FC_NULL

$$\frac{\begin{array}{l}1.\ find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}\\ 2.\ MH(mi) = \textbf{null}\end{array}}{find\_cld((RC, MH), mi.pn, fqn) = \textbf{null}}$$

FC_IMPORTS

$$\frac{\begin{array}{l}1.\ find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}\\ 2.\ MH(mi) = (md, \overline{mi})\\ 3.\ md = \textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}\\ 4.\ find\_cld\_in\_imports(MH, \overline{mi}, fqn) = ctxcld\end{array}}{find\_cld((RC, MH), mi.pn, fqn) = ctxcld}$$

FC_SELF

$$\frac{\begin{array}{l}1.\ find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}\\ 2.\ MH(mi) = (md, \overline{mi})\\ 3.\ md = \textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}\\ 4.\ find\_cld\_in\_imports(MH, \overline{mi}, fqn) = \textbf{null}\\ 5.\ find\_cld\_in\_self(\overline{cld}, pn, fqn) = cld\\ 6.\ package\_name(cld) = pn'\end{array}}{find\_cld((RC, MH), mi.pn, fqn) = (mi.pn', cld)}$$

FC_FAIL

$$\frac{\begin{array}{l}1.\ find\_cld\_in\_core((RC, MH), fqn) = \textbf{null}\\ 2.\ MH(mi) = (md, \overline{mi})\\ 3.\ md = \textbf{module}\ mn\{\overline{cld}\ \overline{m}\ \overline{fqn}\}\\ 4.\ find\_cld\_in\_imports(MH, \overline{mi}, fqn) = \textbf{null}\\ 5.\ find\_cld\_in\_self(\overline{cld}, pn, fqn) = \textbf{null}\end{array}}{find\_cld((RC, MH), mi.pn, fqn) = \textbf{null}}$$

$\boxed{find\_type(P, ctx, cl) = \tau_{opt}}$ – type lookup

FT_OBJ

$$\frac{}{find\_type(P, ctx, \texttt{Object}) = \texttt{Object}}$$

FT_NULL

$$\frac{1.\ find\_cld(P, ctx, fqn) = \textbf{null}}{find\_type(P, ctx, fqn) = \textbf{null}}$$

FT_DCL

$$\frac{1.\ find\_cld(P, ctx, pn.dcl) = (ctx', cld)}{find\_type(P, ctx, pn.dcl) = ctx'.dcl}$$

$\boxed{(P, ctx, cl, nn) \in path\_length}$ – get the length of the inheritance path for cl

PL_OBJ

$$\frac{}{(P, ctx, \texttt{Object}, 0) \in path\_length}$$

PL_FQN

$$\frac{\begin{array}{l}1.\ find\_cld(P, ctx, fqn) = (ctx', cld)\\ 2.\ superclass\_name(cld) = cl\\ 3.\ (P, ctx', cl, nn) \in path\_length\end{array}}{(P, ctx, fqn, nn+1) \in path\_length}$$

$\boxed{acyclic\_clds_{mi}\,P}$ – the class inheritance hierarchy in P is acyclic (starting at mi)

ACM_DEF

$$\frac{1.\ \forall pn\ fqn.\left(\begin{array}{c}\exists ctx'\ cld.\ find\_cld(P, mi.pn, fqn) = (ctx', cld) \longrightarrow \\ \exists nn.(P, mi.pn, fqn, nn) \in path\_length\end{array}\right)}{acyclic\_clds_{mi}\,P}$$

$\boxed{acyclic\_cldsP}$ – the class inheritance hierarchy in P is acyclic

AC_DEF

$$\frac{1.\ \forall mi.acyclic\_clds_{mi}\,P}{acyclic\_cldsP}$$

$\boxed{find\_path\_rec(P, ctx, cl, \overline{ctxcld}) = \overline{ctxcld}_{opt}}$ – class path lookup (recursive part)

FPR_OBJ

$$\frac{}{find\_path\_rec(P, ctx, \texttt{Object}, \overline{ctxcld}) = \overline{ctxcld}}$$

FPR_NULL

$$\frac{1.\ (\neg acyclic\_cldsP)\ \vee\ find\_cld(P, ctx, fqn) = \mathbf{null}}{find\_path\_rec(P, ctx, fqn, \overline{ctxcld}) = \mathbf{null}}$$

FPR_FQN

$$\frac{\begin{array}{l}1.\ acyclic\_cldsP\ \wedge\ find\_cld(P, ctx, fqn) = (ctx', cld) \\ 2.\ superclass\_name(cld) = cl \\ 3.\ find\_path\_rec(P, ctx', cl, \overline{ctxcld}@[(ctx', cld)]) = \overline{ctxcld}_{opt}\end{array}}{find\_path\_rec(P, ctx, fqn, \overline{ctxcld}) = \overline{ctxcld}_{opt}}$$

$\boxed{find\_path(P, ctx, cl) = \overline{ctxcld}_{opt}}$ – class path lookup with a class name

FP_DEF

$$\frac{1.\ find\_path\_rec(P, ctx, cl, [\,]) = \overline{ctxcld}_{opt}}{find\_path(P, ctx, cl) = \overline{ctxcld}_{opt}}$$

$\boxed{find\_path(P, \tau) = \overline{ctxcld}_{opt}}$ – class path lookup with a type

FPTY_OBJ

$$\frac{}{find\_path(P, \texttt{Object}) = [\,]}$$

FPTY_DCL

$$\frac{1.\ find\_path(P, mi.pn, pn.dcl) = \overline{ctxcld}_{opt}}{find\_path(P, mi.pn.dcl) = \overline{ctxcld}_{opt}}$$

$\boxed{fields\_in\_path(\overline{ctxcld}) = \overline{f}}$ – fields lookup in a class path

FIP_EMPTY

$$\frac{}{fields\_in\_path([\,]) = [\,]}$$

FIP_CONS

$$\frac{\begin{array}{l}1.\ class\_fields(cld) = \overline{cl_j\ f_j}^{\,j} \\ 2.\ fields\_in\_path(ctxcld_2\ ..\ ctxcld_k) = \overline{f} \\ 3.\ \overline{f}' = \overline{f_j}^{\,j}; \overline{f}\end{array}}{fields\_in\_path((ctx, cld)\ ctxcld_2\ ..\ ctxcld_k) = \overline{f}'}$$

$\boxed{\mathbf{fields}\,(P, \tau) = \overline{f}_{\,opt}}$ – fields lookup in type τ

FIELDS_NONE

$$\frac{1.\ find\_path(P, \tau) = \mathbf{null}}{\mathbf{fields}\,(P, \tau) = \mathbf{null}}$$

FIELDS_SOME

$$\frac{\begin{array}{l}1.\ find\_path(P, \tau) = \overline{ctxcld} \\ 2.\ fields\_in\_path(\overline{ctxcld}) = \overline{f}\end{array}}{\mathbf{fields}\,(P, \tau) = \overline{f}}$$

$\boxed{methods\_in\_path(\overline{cld}) = \overline{meth}}$ – method names lookup in a path

MIP_EMPTY

$$\frac{}{methods\_in\_path([\,]) = [\,]}$$

MIP_CONS

$$\frac{\begin{array}{l}1.\ class\_methods(cld) = \overline{meth\_def_l}^{\,l} \\ 2.\ meth\_def_l = cl_l\ meth_l(\overline{vd}_l)\{meth\_body_l\}^{\,l} \\ 3.\ methods\_in\_path(cld_2\ ..\ cld_k) = \overline{meth}' \\ 4.\ \overline{meth} = \overline{meth_l}^{\,l}; \overline{meth}'\end{array}}{methods\_in\_path(cld\ cld_2\ ..\ cld_k) = \overline{meth}}$$

$\boxed{\textbf{methods}\ (P, \tau) = \overline{meth}}$ – method names lookup in a type

METHODS_METHODS

$$\frac{\begin{array}{l}1.\ find\_path(P, \tau) = \overline{(ctx_k, cld_k)}^{\,k} \\ 2.\ methods\_in\_path(\overline{cld_k}^{\,k}) = \overline{meth}\end{array}}{\textbf{methods}\ (P, \tau) = \overline{meth}}$$

$\boxed{ftype\_in\_fds(P, ctx, \overline{fd}, f) = \tau^{\perp}_{opt}}$ – field type lookup in a list

FTIF_EMPTY

$$\frac{}{ftype\_in\_fds(P, ctx, [\,], f) = \textbf{null}}$$

FTIF_CONS_BOT

$$\frac{1.\ find\_type(P, ctx, cl) = \textbf{null}}{ftype\_in\_fds(P, ctx, cl\,f;\ fd_2\ ..\ fd_k, f) = \perp}$$

FTIF_CONS_TRUE

$$\frac{1.\ find\_type(P, ctx, cl) = \tau}{ftype\_in\_fds(P, ctx, cl\,f;\ fd_2\ ..\ fd_k, f) = \tau}$$

FTIF_CONS_FALSE

$$\frac{\begin{array}{l}1.\ f \neq f' \\ 2.\ ftype\_in\_fds(P, ctx, fd_2\ ..\ fd_k, f') = \tau^{\perp}_{opt}\end{array}}{ftype\_in\_fds(P, ctx, cl\,f;\ fd_2\ ..\ fd_k, f') = \tau^{\perp}_{opt}}$$

$\boxed{ftype\_in\_path(P, \overline{ctxcld}, f) = \tau_{opt}}$ – field type lookup in a path

FTIP_EMPTY

$$\frac{}{ftype\_in\_path(P, [\,], f) = \textbf{null}}$$

FTIP_CONS_BOT

$$\frac{\begin{array}{l}1.\ class\_fields(cld) = \overline{fd} \\ 2.\ ftype\_in\_fds(P, ctx, \overline{fd}, f) = \perp\end{array}}{ftype\_in\_path(P, (ctx, cld)\ ctxcld_2\ ..\ ctxcld_k, f) = \textbf{null}}$$

FTIP_CONS_TRUE

$$\frac{\begin{array}{l}1.\ class\_fields(cld) = \overline{fd} \\ 2.\ ftype\_in\_fds(P, ctx, \overline{fd}, f) = \tau\end{array}}{ftype\_in\_path(P, (ctx, cld)\ ctxcld_2\ ..\ ctxcld_k, f) = \tau}$$

FTIP_CONS_FALSE

$$\frac{\begin{array}{l}1.\ class\_fields(cld) = \overline{fd} \\ 2.\ ftype\_in\_fds(P, ctx, \overline{fd}, f) = \textbf{null} \\ 3.\ ftype\_in\_path(P, ctxcld_2\ ..\ ctxcld_k, f) = \tau_{opt}\end{array}}{ftype\_in\_path(P, (ctx, cld)\ ctxcld_2\ ..\ ctxcld_k, f) = \tau_{opt}}$$

$\boxed{\textbf{ftype}\ (P, \tau, f) = \tau'}$ – field type lookup

FTYPE

$$\frac{\begin{array}{l}1.\ find\_path(P, \tau) = \overline{ctxcld} \\ 2.\ ftype\_in\_path(P, \overline{ctxcld}, f) = \tau'\end{array}}{\textbf{ftype}\ (P, \tau, f) = \tau'}$$

$\boxed{find\_meth\_def\_in\_list(\overline{meth\_def}, meth) = meth\_def_{opt}}$ – meth. def. lookup (list)

FMDIL_EMPTY

$$\frac{}{find\_meth\_def\_in\_list([\,], meth) = \textbf{null}}$$

FMDIL_CONS_TRUE

$$\frac{1.\ meth\_def = cl\ meth(\overline{vd})\{meth\_body\}}{find\_meth\_def\_in\_list(meth\_def\ meth\_def_2\ ..\ meth\_def_k, meth) = meth\_def}$$

FMDIL_CONS_FALSE

$$\frac{1.\ meth\_def = cl\ meth'(\overline{vd})\{meth\_body\} \qquad 2.\ meth \neq meth' \qquad 3.\ find\_meth\_def\_in\_list(meth\_def_2\ ..\ meth\_def_k, meth) = meth\_def_{opt}}{find\_meth\_def\_in\_list(meth\_def\ meth\_def_2\ ..\ meth\_def_k, meth) = meth\_def_{opt}}$$

$\boxed{find\_meth\_def\_in\_path(\overline{ctxcld}, meth) = ctxmeth\_def_{opt}}$ – meth. def. lookup (path)

FMDIP_EMPTY

$$\frac{}{find\_meth\_def\_in\_path([\,], meth) = \textbf{null}}$$

FMDIP_CONS_TRUE

$$\frac{1.\ class\_methods(cld) = \overline{meth\_def} \qquad 2.\ find\_meth\_def\_in\_list(\overline{meth\_def}, meth) = meth\_def}{find\_meth\_def\_in\_path((ctx, cld)\ ctxcld_2\ ..\ ctxcld_k, meth) = (ctx, meth\_def)}$$

FMDIP_CONS_FALSE

$$\frac{1.\ class\_methods(cld) = \overline{meth\_def} \qquad 2.\ find\_meth\_def\_in\_list(\overline{meth\_def}, meth) = \textbf{null} \qquad 3.\ find\_meth\_def\_in\_path(ctxcld_2\ ..\ ctxcld_k, meth) = ctxmeth\_def_{opt}}{find\_meth\_def\_in\_path((ctx, cld)\ ctxcld_2\ ..\ ctxcld_k, meth) = ctxmeth\_def_{opt}}$$

$\boxed{find\_meth\_def(P, \tau, meth) = ctxmeth\_def_{opt}}$ – method def. lookup in a type

FMD_NULL

$$\frac{1.\ find\_path(P, \tau) = \textbf{null}}{find\_meth\_def(P, \tau, meth) = \textbf{null}}$$

FMD_OPT

$$\frac{1.\ find\_path(P, \tau) = \overline{ctxcld} \qquad 2.\ find\_meth\_def\_in\_path(\overline{ctxcld}, meth) = ctxmeth\_def_{opt}}{find\_meth\_def(P, \tau, meth) = ctxmeth\_def_{opt}}$$

$\boxed{\textbf{mtype}\ (P, \tau, meth) = \pi}$ – method type lookup

MTYPE

$$\frac{\begin{array}{l}1.\ find\_meth\_def(P, \tau, meth) = (ctx, meth\_def) \\ 2.\ meth\_def = cl\ meth(\overline{cl_k\ var_k}^{\,k})\{meth\_body\} \\ 3.\ find\_type(P, ctx, cl) = \tau' \\ 4.\ \overline{find\_type(P, ctx, cl_k) = \tau_k}^{\,k} \\ 5.\ \pi = \overline{\tau_k}^{\,k} \rightarrow \tau'\end{array}}{\textbf{mtype}\ (P, \tau, meth) = \pi}$$

$\boxed{P \vdash \tau \prec \tau'}$ – subtyping

STY_OBJ

$$\frac{1.\ find\_path(P, \tau) = \overline{ctxcld}}{P \vdash \tau \prec \texttt{Object}}$$

STY_DCL

$$\frac{1.\ find\_path(P, \tau) = \overline{ctxcld} \qquad 2.\ find\_cld(P, mi'.pn', pn'.dcl') = ctxcld \qquad 3.\ ctxcld \in \overline{ctxcld}}{P \vdash \tau \prec mi'.pn'.dcl'}$$

$\boxed{P \vdash \overline{\tau} \prec \overline{\tau'}}$ – normal, multiple subtyping

STY_MANY

$$\frac{1.\ \overline{\tau} = \overline{\tau_k}^{\,k} \qquad 2.\ \overline{\tau'} = \overline{\tau'_k}^{\,k} \qquad 3.\ \overline{P \vdash \tau_k \prec \tau'_k}^{\,k}}{P \vdash \overline{\tau} \prec \overline{\tau'}}$$

$\boxed{P \vdash \tau_{opt} \prec \tau_{opt}'}$ – option subtyping

STY_OPTION

$$\frac{1.\ \tau_{opt} = \tau \quad 2.\ \tau_{opt}' = \tau' \quad 3.\ P \vdash \tau \prec \tau'}{P \vdash \tau_{opt} \prec \tau_{opt}'}$$

$\boxed{P, H \vdash v_{opt} \prec \tau_{opt}}$ – well-formed value

WF_NULL

$$\frac{1.\ \tau_{opt} = \tau}{P, H \vdash \mathbf{null} \prec \tau_{opt}}$$

WF_OBJECT

$$\frac{1.\ P \vdash H(oid) \prec \tau_{opt}}{P, H \vdash oid \prec \tau_{opt}}$$

$\boxed{P, \Gamma, H \vdash L}$ – well-formed variable state

WF_VARSTATE

$$\frac{1.\ \mathbf{finite}\,(\mathbf{dom}\,(L)) \quad 2.\ \forall x \in \mathbf{dom}\,(\Gamma).P, H \vdash L(x) \prec \Gamma(x)}{P, \Gamma, H \vdash L}$$

$\boxed{P \vdash H}$ – well-formed heap

WF_HEAP

$$\frac{1.\ \mathbf{finite}\,(\mathbf{dom}\,(H)) \quad 2.\ \forall oid \in \mathbf{dom}\,(H).\left(\begin{array}{c}\exists \tau.H(oid) = \tau\ \wedge\ \exists \overline{f}.\mathbf{fields}\,(P,\tau) = \overline{f}\ \wedge \\ \forall f \in \overline{f}.\exists \tau'.\left(\begin{array}{c}\mathbf{ftype}\,(P,\tau,f) = \tau'\ \wedge \\ P, H \vdash H(oid, f) \prec \tau'\end{array}\right)\end{array}\right)}{P \vdash H}$$

$\boxed{\Gamma \vdash config}$ – well-formed configuration

WF_ALL_EX

$$\frac{1.\ \vdash P \quad 2.\ P \vdash H \quad 3.\ P, \Gamma, H \vdash L}{\Gamma \vdash (P, L, H, Exception)}$$

WF_ALL

$$\frac{1.\ \vdash P \quad 2.\ P \vdash H \quad 3.\ P, \Gamma, H \vdash L \quad 4.\ \overline{P, \Gamma \vdash s_k}^{\,k}}{\Gamma \vdash (P, L, H, \overline{s_k}^{\,k})}$$

$\boxed{P, \Gamma \vdash s}$ – well-formed statement

WF_BLOCK

$$\frac{1.\ \overline{P, \Gamma \vdash s_k}^{\,k}}{P, \Gamma \vdash \{\ \overline{s_k}^{\,k}\ \}}$$

WF_VAR_ASSIGN

$$\frac{1.\ P \vdash \Gamma(x) \prec \Gamma(var)}{P, \Gamma \vdash var = x;}$$

WF_FIELD_READ

$$\frac{1.\ \Gamma(x) = \tau \quad 2.\ \mathbf{ftype}\,(P,\tau,f) = \tau' \quad 3.\ P \vdash \tau' \prec \Gamma(var)}{P, \Gamma \vdash var = x.f;}$$

WF_FIELD_WRITE

$$\frac{1.\ \Gamma(x) = \tau \quad 2.\ \mathbf{ftype}\,(P,\tau,f) = \tau' \quad 3.\ P \vdash \Gamma(y) \prec \tau'}{P, \Gamma \vdash x.f = y;}$$

WF_IF

$$\frac{1.\ P \vdash \Gamma(x) \prec \Gamma(y)\ \vee\ P \vdash \Gamma(y) \prec \Gamma(x) \quad 2.\ P, \Gamma \vdash s_1 \quad 3.\ P, \Gamma \vdash s_2}{P, \Gamma \vdash \mathbf{if}\ (x == y) s_1\ \mathbf{else}\ s_2}$$

WF_NEW

$$\frac{1.\ find\_type(P, ctx, cl) = \tau \quad 2.\ P \vdash \tau \prec \Gamma(var)}{P, \Gamma \vdash var = \mathbf{new}_{ctx}\, cl();}$$

WF_MCALL

$$\frac{1.\ \overline{y} = \overline{y_k}^{\,k} \quad 2.\ \Gamma(x) = \tau \quad 3.\ \mathbf{mtype}\,(P,\tau,meth) = \overline{\tau_k}^{\,k} \to \tau' \quad 4.\ \overline{P \vdash \Gamma(y_k) \prec \tau_k}^{\,k} \quad 5.\ P \vdash \tau' \prec \Gamma(var)}{P, \Gamma \vdash var = x.meth(\overline{y});}$$

$\boxed{P \vdash_\tau meth\_def}$ – well-formed method in τ

WF_METHOD

$$\frac{\begin{array}{l}1.\ \mathbf{distinct}\ (\overline{var_k}^{\,k}) \\ 2.\ \overline{find\_type(P, ctx, cl_k) = \tau_k}^{\,k} \\ 3.\ \Gamma = [\overline{var_k \mapsto \tau_k}^{\,k}][\mathbf{this} \mapsto ctx.dcl] \\ 4.\ \overline{P, \Gamma \vdash s_l}^{\,l} \quad 5.\ find\_type(P, ctx, cl) = \tau \\ 6.\ P \vdash \Gamma(y) \prec \tau\end{array}}{P \vdash_{ctx.dcl} cl\ meth(\overline{cl_k\ var_k}^{\,k})\{\overline{s_l}^{\,l}\ \mathbf{return}\ y;\}}$$

$\boxed{P \vdash_{ctx} (dcl, cl, \overline{fd}, \overline{meth\_def})}$ – well-formed class in ctx (generic rule)

WF_CLASS_COMMON

$$\frac{\begin{array}{l}1.\ find\_type(P, ctx, cl) = \tau \\ 2.\ ctx.dcl \neq \tau \quad 3.\ \mathbf{distinct}\ (\overline{f_j}^{\,j}) \\ 4.\ \mathbf{fields}\ (P, \tau) = \overline{f} \quad 5.\ \overline{f_j}^{\,j} \perp \overline{f} \\ 6.\ \overline{find\_type(P, ctx, cl_j) = \tau_j}^{\,j} \\ 7.\ \overline{P \vdash_{ctx.dcl} meth\_def_k}^{\,k} \\ 8.\ \overline{method\_name(meth\_def_k) = meth_k}^{\,k} \\ 9.\ \mathbf{distinct}\ (\overline{meth_k}^{\,k}) \\ 10.\ \mathbf{methods}\ (P, \tau) = \overline{meth'_l}^{\,l} \\ 11.\ \overline{\mathbf{mtype}\ (P, ctx.dcl, meth'_l) = \pi_l}^{\,l} \\ 12.\ \overline{\mathbf{mtype}\ (P, \tau, meth'_l) = \pi'_l}^{\,l} \\ 13.\ \overline{meth'_l \in \overline{meth_k}^{\,k} \longrightarrow \pi_l = \pi'_l}^{\,l}\end{array}}{P \vdash_{ctx} (dcl, cl, \overline{cl_j\ f_j;}^{\,j}, \overline{meth\_def_k}^{\,k})}$$

$\boxed{P \vdash_{mi} cld}$ – well-formed class in mi

WF_CLASS

$$\frac{1.\ P \vdash_{mi.pn} (dcl, cl, \overline{fd}, \overline{meth\_def})}{P \vdash_{mi} \mathbf{package}\ pn;\ am\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def}\}}$$

$\boxed{P \vdash_{mi} md}$ – well-formed module instance

WF_MODULE

$$\frac{\begin{array}{l}1.\ \overline{full\_name(cld_k) = fqn_k}^{\,k} \\ 2.\ \mathbf{distinct}\ (\overline{fqn_k}^{\,k}) \\ 3.\ \overline{P \vdash_{mi} cld_k}^{\,k} \\ 4.\ acyclic\_clds_{mi} P\end{array}}{P \vdash_{mi} \mathbf{module}\ mn\{\overline{cld_k}^{\,k}\ \overline{m}\ \overline{fqn}\}}$$

$\boxed{MH \vdash \phi}$ – well-formed module instance cache

WF_RMIS

$$\frac{1.\ \mathbf{ran}\ (\phi)\ \subseteq\ \mathbf{dom}\ (MH)}{MH \vdash \phi}$$

$\boxed{P \vdash R}$ – well-formed repository

WF_BOOTSTRAP_REP

$$\frac{\begin{array}{l}1.\ find\_md\_in\_mds(\overline{md^{\mathsf{c}}}, core\_m) = md^{\mathsf{c}} \\ 2.\ MH \vdash \phi\end{array}}{(RC, MH) \vdash \mathbf{bootstrap\ repository}\ \{\overline{md^{\mathsf{c}}}; \phi\}}$$

WF_NORMAL_REP

$$\frac{\begin{array}{l}1.\ r \neq rn \quad 2.\ rn \in \mathbf{dom}\ (RC) \\ 3.\ MH \vdash \phi\end{array}}{(RC, MH) \vdash \mathbf{repository}\ r\ \mathbf{child\ of}\ rn\{\overline{md^{\mathsf{c}}}; \phi\}}$$

$\boxed{MH \vdash RC}$ – well-formed repository context

WF_RC

$$\frac{1.\ \forall rn \in \mathbf{dom}\,(RC).\exists R.\left(\begin{array}{l} RC(rn) = R \ \wedge\ R\_name(R) = rn \ \wedge \\ (RC, MH) \vdash R \end{array}\right) \quad 2.\ bootstrap\_r \in \mathbf{dom}\,(RC)}{MH \vdash RC}$$

$\boxed{RC \vdash MH}$ – well-formed module hierarchy

WF_MH

$$\frac{1.\ acyclic\_mh MH \quad 2.\ \forall mi \in \mathbf{dom}\,(MH).\exists md\ \overline{mi}.MH(mi) = (md, \overline{mi})\ \wedge\ (RC, MH) \vdash_{mi} md}{RC \vdash MH}$$

$\boxed{\vdash P}$ – well-formed program

WF_P

$$\frac{1.\ MH \vdash RC \quad 2.\ RC \vdash MH \quad 3.\ acyclic\_clds(RC, MH)}{\vdash (RC, MH)}$$

$\boxed{find\_pkg\_clds(\overline{cld^{\mathsf{c}}}_1, \overline{pn}) = \overline{cld^{\mathsf{c}}}_2}$ –

FPC_EMPTY

$$\overline{find\_pkg\_clds(, \overline{pn}) =}$$

FPC_CONS_TRUE

$$\frac{1.\ cld^{\mathsf{c}} = \mathbf{package}\ pn;\ am\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def^{\mathsf{c}}}\} \quad 2.\ pn \in \overline{pn} \quad 3.\ find\_pkg\_clds(cld^{\mathsf{c}}_2\ ..\ cld^{\mathsf{c}}_k, \overline{pn}) = \overline{cld^{\mathsf{c}}}}{find\_pkg\_clds(cld^{\mathsf{c}}\ cld^{\mathsf{c}}_2\ ..\ cld^{\mathsf{c}}_k, \overline{pn}) = cld^{\mathsf{c}} \# \overline{cld^{\mathsf{c}}}}$$

FPC_CONS_FALSE

$$\frac{1.\ cld^{\mathsf{c}} = \mathbf{package}\ pn;\ am\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def^{\mathsf{c}}}\} \quad 2.\ pn \notin \overline{pn} \quad 3.\ find\_pkg\_clds(cld^{\mathsf{c}}_2\ ..\ cld^{\mathsf{c}}_k, \overline{pn}) = \overline{cld^{\mathsf{c}}}}{find\_pkg\_clds(cld^{\mathsf{c}}\ cld^{\mathsf{c}}_2\ ..\ cld^{\mathsf{c}}_k, \overline{pn}) = \overline{cld^{\mathsf{c}}}}$$

$\boxed{SRC \vdash mf \rightsquigarrow md^{\mathsf{c}}}$ – packaging a module file to a module def., compile-time code

PCG_MF

$$\frac{1.\ find\_pkg\_clds(\overline{cld^{\mathsf{c}}}_1, \overline{pn_j}^{\,j}) = \overline{cld^{\mathsf{c}}}_2}{\overline{cld^{\mathsf{c}}}_1 \vdash \mathbf{superpackage}\ mn\{\overline{\mathbf{member}\ pn_j;}^{\,j}\ \overline{\mathbf{import}\ m_k;}^{\,k}\ \overline{\mathbf{export}\ fqn_l;}^{\,l}\} \rightsquigarrow \mathbf{module}\ mn\{\overline{cld^{\mathsf{c}}}_2\ \overline{m_k}^{\,k}\ \overline{fqn_l}^{\,l}\}}$$

$\boxed{\vdash_{ctx} s^{\mathsf{c}} \rightsquigarrow s}$ – context insertion for a statement

CI_S_BLOCK

$$\frac{1.\ \overline{\vdash_{ctx} s^{\mathsf{c}}_k \rightsquigarrow s_k}^{\,k}}{\vdash_{ctx} \{\overline{s^{\mathsf{c}}_k}^{\,k}\} \rightsquigarrow \{\overline{s_k}^{\,k}\}}$$

CI_S_VAR_ASSIGN

$$\overline{\vdash_{ctx} var = x; \rightsquigarrow var = x;}$$

CI_S_FIELD_READ

$$\overline{\vdash_{ctx} var = x.f; \rightsquigarrow var = x.f;}$$

CI_S_FIELD_WRITE

$$\overline{\vdash_{ctx} x.f = y; \rightsquigarrow x.f = y;}$$

CI_S_IF

$$\frac{1.\ \vdash_{ctx} s^{\mathsf{c}}_1 \rightsquigarrow s_1 \quad 2.\ \vdash_{ctx} s^{\mathsf{c}}_2 \rightsquigarrow s_2}{\vdash_{ctx} \mathbf{if}\ (x == y) s^{\mathsf{c}}_1\ \mathbf{else}\ s^{\mathsf{c}}_2 \rightsquigarrow \mathbf{if}\ (x == y) s_1\ \mathbf{else}\ s_2}$$

CI_S_MCALL

$$\overline{\vdash_{ctx} var = x.meth(\overline{y_k}^{\,k}); \rightsquigarrow var = x.meth(\overline{y_k}^{\,k});}$$

CI_S_NEW

$$\overline{\vdash_{ctx} var = \mathbf{new}\ cl(); \rightsquigarrow var = \mathbf{new}_{ctx}\ cl();}$$

$\boxed{\vdash_{ctx} meth\_def^{\mathsf{c}} \rightsquigarrow meth\_def}$ – context insertion for method def.'s

CI_METH_DEF

$$\frac{1.\ \overline{\vdash_{ctx} s^{\mathsf{c}} \rightsquigarrow s}^{\,k}}{\vdash_{ctx} cl\ meth(\overline{vd})\{\overline{s_k^{\mathsf{c}}}^{\,k}\ \mathbf{return}\ y;\} \rightsquigarrow cl\ meth(\overline{vd})\{\overline{s}^{\,k}\ \mathbf{return}\ y;\}}$$

$\boxed{\vdash_{mi} cld^{\mathsf{c}} \rightsquigarrow cld}$ – context insertion for class def.'s

CI_CLD

$$\frac{\begin{array}{l}1.\ cld^{\mathsf{c}} = \mathbf{package}\ pn;\ am\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def_k^{\mathsf{c}}}^{\,k}\} \\ 2.\ \overline{\vdash_{mi.pn} meth\_def_k^{\mathsf{c}} \rightsquigarrow meth\_def_k}^{\,k} \\ 3.\ cld = \mathbf{package}\ pn;\ am\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\{\overline{fd}\ \overline{meth\_def_k}^{\,k}\}\end{array}}{\vdash_{mi} cld^{\mathsf{c}} \rightsquigarrow cld}$$

$\boxed{\vdash_{mi} md^{\mathsf{c}} \rightsquigarrow md}$ – module def. translation

CI_MODULE

$$\frac{1.\ \overline{\vdash_{mi} cld_k^{\mathsf{c}} \rightsquigarrow cld_k}^{\,k}}{\vdash_{mi} \mathbf{module}\ mn\{\overline{cld_k^{\mathsf{c}}}^{\,k}\ \overline{m}\ \overline{fqn}\} \rightsquigarrow \mathbf{module}\ mn\{\overline{cld_k}^{\,k}\ \overline{m}\ \overline{fqn}\}}$$

$\boxed{\theta \vdash s \rightsquigarrow s'}$ – variable translation for a statement

TR_S_BLOCK

$$\frac{1.\ \overline{\theta \vdash s_k \rightsquigarrow s'_k}^{\,k}}{\theta \vdash \{\ \overline{s_k}^{\,k}\ \} \rightsquigarrow \{\ \overline{s'_k}^{\,k}\ \}}$$

TR_S_VAR_ASSIGN

$$\frac{\begin{array}{l}1.\ \theta(var) = var' \\ 2.\ \theta(x) = x'\end{array}}{\theta \vdash var = x; \rightsquigarrow var' = x';}$$

TR_S_FIELD_READ

$$\frac{\begin{array}{l}1.\ \theta(var) = var' \\ 2.\ \theta(x) = x'\end{array}}{\theta \vdash var = x.f; \rightsquigarrow var' = x'.f;}$$

TR_S_FIELD_WRITE

$$\frac{\begin{array}{l}1.\ \theta(x) = x' \\ 2.\ \theta(y) = y'\end{array}}{\theta \vdash x.f = y; \rightsquigarrow x'.f = y';}$$

TR_S_IF

$$\frac{\begin{array}{lll}1.\ \theta(x) = x' & 2.\ \theta(y) = y' & 3.\ \theta \vdash s_1 \rightsquigarrow s'_1 \\ 4.\ \theta \vdash s_2 \rightsquigarrow s'_2 & & \end{array}}{\theta \vdash \mathbf{if}\ (x == y) s_1\ \mathbf{else}\ s_2 \rightsquigarrow \mathbf{if}\ (x' == y') s'_1\ \mathbf{else}\ s'_2}$$

TR_S_NEW

$$\frac{1.\ \theta(var) = var'}{\theta \vdash var = \mathbf{new}_{ctx}\, cl(); \rightsquigarrow var' = \mathbf{new}_{ctx}\, cl();}$$

TR_S_MCALL

$$\frac{\begin{array}{lll}1.\ \theta(var) = var' & 2.\ \theta(x) = x' & \\ 3.\ \overline{\theta(y_k) = y'_k}^{\,k} & & \end{array}}{\theta \vdash var = x.meth(\overline{y_k}^{\,k}); \rightsquigarrow var' = x'.meth(\overline{y'_k}^{\,k});}$$

$\boxed{config \longrightarrow config'}$ – reduction of a statement

R_BLOCK

$$\frac{}{(P, L, H, \{\ \overline{s_k}^{\,k}\ \}\ \overline{s'_l}^{\,l}) \longrightarrow (P, L, H,\ \overline{s_k}^{\,k}\ \overline{s'_l}^{\,l})}$$

R_VAR_ASSIGN

$$\frac{1.\ L(x) = v}{(P, L, H, var = x;\ \overline{s_l}^{\,l}) \longrightarrow (P, L[var \mapsto v], H,\ \overline{s_l}^{\,l})}$$

R_FIELD_READ_NPE

$$\frac{1.\ L(x) = \mathbf{null}}{(P, L, H, var = x.f;\ \overline{s_l}^{\,l}) \longrightarrow (P, L, H, \mathbf{NPE})}$$

R_FIELD_READ

$$\frac{\begin{array}{ll}1.\ L(x) = oid & 2.\ H(oid, f) = v\end{array}}{(P, L, H, var = x.f;\ \overline{s_l}^{\,l}) \longrightarrow (P, L[var \mapsto v], H,\ \overline{s_l}^{\,l})}$$

R_FIELD_WRITE_NPE

$$\frac{1.\ L(x) = \mathbf{null}}{(P, L, H, x.f = y;\ \overline{s_l}^{\,l}) \longrightarrow (P, L, H, \mathbf{NPE})}$$

R_FIELD_WRITE

$$\frac{\begin{array}{ll}1.\ L(x) = oid & 2.\ L(y) = v\end{array}}{(P, L, H, x.f = y;\ \overline{s_l}^{\,l}) \longrightarrow (P, L, H[(oid, f) \mapsto v],\ \overline{s_l}^{\,l})}$$

R_IF_TRUE

$$\frac{\begin{array}{lll}1.\ L(x) = v & 2.\ L(y) = w & 3.\ v = w\end{array}}{(P, L, H, \mathbf{if}\ (x == y) s_1\ \mathbf{else}\ s_2\ \overline{s'_l}^{\,l}) \longrightarrow (P, L, H, s_1\ \overline{s'_l}^{\,l})}$$

R_IF_FALSE

$$\frac{\begin{array}{lll}1.\ L(x) = v & 2.\ L(y) = w & 3.\ v \neq w\end{array}}{(P, L, H, \mathbf{if}\ (x == y) s_1\ \mathbf{else}\ s_2\ \overline{s'_l}^{\,l}) \longrightarrow (P, L, H, s_2\ \overline{s'_l}^{\,l})}$$

R_NEW

$$\frac{\begin{array}{ll}1.\ find\_type(P, ctx, cl) = \tau & 2.\ \mathbf{fields}\ (P, \tau) = \overline{f_k}^{\,k} \\ 3.\ oid \notin \mathbf{dom}\ (H) & 4.\ H' = H[oid \mapsto (\tau, \overline{f_k \mapsto \mathbf{null}}^{\,k})]\end{array}}{(P, L, H, var = \mathbf{new}_{ctx}\, cl();\ \overline{s_l}^{\,l}) \longrightarrow (P, L[var \mapsto oid], H',\ \overline{s_l}^{\,l})}$$

R_MCALL_NPE

$$\frac{1.\ L(x) = \mathbf{null}}{(P, L, H, var = x.meth(\overline{y_k}^{\,k});\ \overline{s_l}^{\,l}) \longrightarrow (P, L, H, \mathbf{NPE})}$$

R_MCALL

$$\frac{\begin{array}{ll}1.\ L(x) = oid & 2.\ H(oid) = \tau \\ 3.\ find\_meth\_def(P, \tau, meth) = (ctx, cl\ meth(\overline{cl_k\ var_k}^{\,k})\{\overline{s'_j}^{\,j}\ \mathbf{return}\ y;\}) & \\ 4.\ \overline{var'_k}^{\,k} \perp \mathbf{dom}\,(L) & 5.\ \mathbf{distinct}\,(\overline{var'_k}^{\,k}) \quad 6.\ x' \notin \mathbf{dom}\,(L) \\ 7.\ x' \notin \overline{var'_k}^{\,k} & 8.\ \overline{L(y_k) = v_k}^{\,k} \\ 9.\ L' = L[\overline{var'_k \mapsto v_k}^{\,k}][x' \mapsto oid] & \\ 10.\ \theta = [\overline{var_k \mapsto var'_k}^{\,k}][\mathbf{this} \mapsto x'] & 11.\ \overline{\theta \vdash s'_j \rightsquigarrow s''_j}^{\,j} \\ 12.\ \theta(y) = y' & \end{array}}{(P, L, H, var = x.meth(\overline{y_k}^{\,k});\ \overline{s_l}^{\,l}) \longrightarrow (P, L', H,\ \overline{s''_j}^{\,j}\ var = y';\ \overline{s_l}^{\,l})}$$

$\boxed{config \xrightarrow{\overline{ia}} config'}$ – reduction of internal actions

R_NO_ACTION

$$\frac{}{config \longrightarrow config}$$

R_ACTION_LIST

$$\frac{\begin{array}{ll}1.\ (P, L, H, \overline{s_l}^{\,l}) \xrightarrow{ia} (P', L, H, \overline{s_l}^{\,l}) & \\ 2.\ (P', L, H, \overline{s_l}^{\,l}) \xrightarrow{ia_2\ ..\ ia_k} (P'', L, H, \overline{s_l}^{\,l}) & \end{array}}{(P, L, H, \overline{s_l}^{\,l}) \xrightarrow{ia\ ia_2\ ..\ ia_k} (P'', L, H, \overline{s_l}^{\,l})}$$

R_INSTALL

$$\frac{\begin{array}{ll}1.\ RC(rn) = R & 2.\ R\_body(R) = (\overline{md^c_k}^{\,k}, \phi) \\ 3.\ md\_name(md^c) = m & \\ 4.\ \overline{md\_name(md^c_k) = mn_k}^{\,k} & 5.\ m \notin \overline{mn_k}^{\,k} \\ 6.\ R\_update(R, md^c\ \overline{md^c_k}^{\,k}, \phi) = R' & \\ 7.\ RC' = RC[rn \mapsto R'] & \end{array}}{((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{rn.\,\mathbf{install}\,(md^c)} ((RC', MH), L, H, \overline{s_l}^{\,l})}$$

R_UNINSTALL

$$\frac{\begin{array}{ll}1.\ RC(rn) = R & 2.\ R\_body(R) = (\overline{md^c}_1, \phi) \\ 3.\ find\_md\_in\_mds(\overline{md^c}_1, m) = md^c & \\ 4.\ mds\_rm(\overline{md^c}_1, md^c) = \overline{md^c}_2 & \\ 5.\ R\_update(R, \overline{md^c}_2, \phi \setminus md^c) = R' & \\ 6.\ RC' = RC[rn \mapsto R'] & \end{array}}{((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{rn.\,\mathbf{uninstall}\,(m)} ((RC', MH), L, H, \overline{s_l}^{\,l})}$$

R_EXISTING_INSTANCE

$$\frac{\begin{array}{ll}1.\ find\_md(RC, rn_1, m) = (rn_2, md^c) & 2.\ RC(rn_2) = R_2 \\ 3.\ R\_body(R_2) = (\overline{md^c}_2, \phi_2) & 4.\ \phi_2(md^c) = mi \end{array}}{((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{mi = rn_1.get\_instance(m)} ((RC, MH), L, H, \overline{s_l}^{\,l})}$$

R_NEW_INSTANCE

$$\frac{\begin{array}{ll}1.\ find\_md(RC, rn_1, m) = (rn_2, md^c) & 2.\ RC(rn_2) = R_2 \\ 3.\ R\_body(R_2) = (\overline{md^c}_2, \phi_2) & 4.\ \phi_2(md^c) = \mathbf{null} \\ 5.\ md^c = \mathbf{module}\ m\{\overline{cld^c}\ \overline{m_k}^{\,k}\ \overline{fqn}\} & \\ 6.\ ((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{\overline{mi_k = rn_2.get\_instance(m_k)}^{\,k}} ((RC', MH'), L, H, \overline{s_l}^{\,l}) & \\ 7.\ mi \notin \mathbf{dom}\,(MH') & 8.\ \vdash_{mi} md^c \rightsquigarrow md \\ 9.\ MH'' = MH'\,[mi \mapsto (md, \overline{mi_k}^{\,k})] & 10.\ RC'(rn_2) = R'_2 \\ 11.\ R\_body(R'_2) = (\overline{md^c}_3, \phi_3) & \\ 12.\ R\_update(R'_2, \overline{md^c}_3, \phi_3[md^c \mapsto mi]) = R''_2 & \\ 13.\ RC'' = RC'[rn_2 \mapsto R''_2] & 14.\ (RC'', MH'') \vdash_{mi} md \end{array}}{((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{mi = rn_1.get\_instance(m)} ((RC'', MH''), L, H, \overline{s_l}^{\,l})}$$

$\boxed{config \xrightarrow{a} config'}$ – reduction of an administrator action

ADMIN_INSTALL

$$\frac{1.\ ((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{rn.\,\mathbf{install}\,(md^{\mathsf{c}})} ((RC', MH), L, H, \overline{s_l}^{\,l})}{((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{rn.\,\mathbf{install}\,(md^{\mathsf{c}});} ((RC', MH), L, H, \overline{s_l}^{\,l})}$$

ADMIN_UNINSTALL

$$\frac{1.\ ((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{rn.\,\mathbf{uninstall}\,(m)} ((RC', MH), L, H, \overline{s_l}^{\,l})}{((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{rn.\,\mathbf{uninstall}\,(m);} ((RC', MH), L, H, \overline{s_l}^{\,l})}$$

ADMIN_NEW_INSTANCE

$$\frac{1.\ ((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{mi=rn_1.get\_instance(m)} ((RC', MH'), L, H, \overline{s_l}^{\,l})}{((RC, MH), L, H, \overline{s_l}^{\,l}) \xrightarrow{rn_1.\,\mathbf{initialise}\,(m);} ((RC', MH'), L, H, \overline{s_l}^{\,l})}$$

$\boxed{(P, mi, P') \in wf\_P\_change}$ – well-formed program change (proof related)

WRC_INSTALL

$$\frac{\begin{array}{ll}1.\ \vdash (RC, MH) & 2.\ mi \notin \mathbf{dom}\,(MH)\\ 3.\ RC(rn) = R & 4.\ R\_body(R) = (\overline{md^{\mathsf{c}}}, \phi)\\ 5.\ md\_name(md^{\mathsf{c}}) = m & \\ 6.\ R\_update(R, md^{\mathsf{c}} \# \overline{md^{\mathsf{c}}}, \phi) = R' & \end{array}}{((RC, MH), mi, (RC[rn \mapsto R'], MH)) \in wf\_P\_change}$$

WRC_UNINSTALL

$$\frac{\begin{array}{ll}1.\ \vdash (RC, MH) & 2.\ mi \notin \mathbf{dom}\,(MH)\\ 3.\ RC(rn) = R & 4.\ R\_body(R) = (\overline{md^{\mathsf{c}}}_1, \phi)\\ 5.\ find\_md\_in\_mds(\overline{md^{\mathsf{c}}}_1, m) = md^{\mathsf{c}} & \\ 6.\ mds\_rm(\overline{md^{\mathsf{c}}}_1, md^{\mathsf{c}}) = \overline{md^{\mathsf{c}}}_2 & \\ 7.\ R\_update(R, \overline{md^{\mathsf{c}}}_2, \phi \setminus md^{\mathsf{c}}) = R' & \end{array}}{((RC, MH), mi, (RC[rn \mapsto R'], MH)) \in wf\_P\_change}$$

WRC_NEW_INSTANCE

$$\frac{\begin{array}{l}1.\ \vdash (RC, MH)\\ 2.\ mi \notin \mathbf{dom}\,(MH)\\ 3.\ RC(rn) = R\\ 4.\ R\_body(R) = (\overline{md^{\mathsf{c}}}, \phi)\\ 5.\ \overline{mi} \subseteq \mathbf{dom}\,(MH)\\ 6.\ md\_name(md^{\mathsf{c}}) = m\\ 7.\ R\_update(R, \overline{md^{\mathsf{c}}}, \phi[md^{\mathsf{c}} \mapsto \overline{mi}]) = R'\\ 8.\ RC' = RC[rn \mapsto R']\\ 9.\ MH' = MH\,[\overline{mi} \mapsto (md, \overline{mi})]\\ 10.\ (RC', MH') \vdash_{mi} md\end{array}}{((RC, MH), mi, (RC', MH')) \in wf\_P\_change}$$