

*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  
*amn* abstract module name  
*j, k, l* index

<i>mf</i>	::=   <i>repl</i> <b>superpackage</b> <i>mn</i> { $\overline{\text{member } pn_j;}$ $\overline{\text{imp}_k;}$ $\overline{\text{export } fqn_l;}$ }	module file def.
<i>repl</i>	::=     <b>replicating</b>   <b>singleton</b>   <i>md_repl</i> ( <i>md<sup>c</sup></i> )	replication modifier default replicating singleton M get <i>md<sup>c</sup></i> 's repl
<i>mn</i>	::=   <i>core_m</i>   <i>m</i>	module name core module standard module
<i>imp</i>	::=   <b>import</b> <i>m br</i>   <b>import shared</b> <i>m br</i>   <b>import own</b> <i>m br</i>   <b>import</b> <i>m as amn br</i>	import statement default shared own as
<i>br</i>	::=     <b>with</b> <i>fqn<sub>1</sub> as fqn'<sub>1</sub>, ..., fqn<sub>k</sub> as fqn'<sub>k</sub></i>	boundary renaming (( <i>fqn</i> × <i>fqn</i> ) list) M no renaming M renaming pairs
<i>fqn</i>	::=   <i>pn.dcl</i>   <i>br</i> [ <i>fqn</i> ]	fully-qualified name def. M rename if <i>fqn</i> in <i>br</i> , else leave alone
<i>SRC</i>	::=   $\overline{cld^c}$	source files ( $\overline{cld^c}$ ) M def.
$\overline{cld^c}$	::=   <i>cld<sub>1</sub><sup>c</sup> .. cld<sub>k</sub><sup>c</sup></i>   <i>cld<sup>c</sup> # cld<sup>c</sup></i>	class def.'s, compile-time code ( <i>cld<sup>c</sup></i> list) M def. M cons
<i>cld<sup>c</sup></i>	::=   <i>pd am class dcl extends cl</i> { $\overline{fd \text{ meth\_def}^c}$ }	class, compile-time code def.
<i>pd</i>	::=   <b>package</b> <i>pn</i> ;	package declaration ( <i>pn</i> ) M def.

$am$	$::=$     <b>public</b>	access modifier default public
$C, cl$	$::=$   <b>Object</b>   $fqn$	class name top class fully qualified name
$\overline{fd}$	$::=$   []   $fd_1 .. fd_k$	field declarations ( $fd$ list) M empty M def.
$fd$	$::=$   $cl f;$	field declaration def.
$\overline{meth\_def^c}$	$::=$   $meth\_def_1^c .. meth\_def_k^c$	method def.'s, compile-time code ( $meth\_def^c$ list) M def.
$meth\_def^c$	$::=$   $meth\_sig\{meth\_body^c\}$	method def., compile-time code def.
$meth\_sig$	$::=$   $cl meth(\overline{vd})$	method signature def.
$\overline{vd}$	$::=$   $vd_1 .. vd_k$	variable declarations ( $vd$ list) M def.
$vd$	$::=$   $cl var$	variable declaration def.
$meth\_body^c$	$::=$   $s_1^c .. s_k^c \mathbf{return} \overline{y};$	method body, compile-time code def.
$s^c$	$::=$   $\{ \overline{s_k^c}^k \}$   $var = x;$   $var = x.f;$   $x.f = y;$   $\mathbf{if} (x == y) s_1^c \mathbf{else} s_2^c$   $var = \mathbf{new} cl();$   $var = x.meth(\overline{y});$	statement, compile-time code block variable assignment field read field write conditional branch object construction method call
$TVar, x, y$	$::=$   $var$   <b>this</b>	term variable normal variable ref. to current object
$\overline{x}, \overline{y}$	$::=$   $x_1 .. x_k$	term variables ( $x$ list) M def
$P$	$::=$   $(RC, MH)$	program def.

$RC$	$::=$ $ $ $[\ ]$ $ $ $RC[rn \mapsto R]$	repository context ( $rn \rightarrow R$ ) M empty repository context M $rn$ maps to $R$ in $RC$
$rn$	$::=$ $ $ $bootstrap_r$ $ $ $r$	repository name bootstrap standard
$R$	$::=$ $ $ <b>bootstrap repository</b> $\{\overline{md^c}; \phi\}$ $ $ <b>repository <math>r</math> child of <math>rn\{\overline{md^c}; \phi\}</math></b>	repository bootstrap standard
$\overline{md^c}$	$::=$ $ $ $md_1^c .. md_k^c$ $ $ $md^c \# \overline{md^c}$	module def.'s, compile-time code ( $md^c$ list) M def. M cons
$md^c$	$::=$ $ $ <b>repl module</b> $mn\{\overline{cld^c} \overline{imp_k^c} \overline{fq_n}\}$	module definition def.
$\overline{fq_n}$	$::=$ $ $ $fq_{n_1} .. fq_{n_k}$ $ $ $\overline{fq_n} \cap \overline{fq_n}'$	fully-qualified names ( $fq_n$ list) M def. M intersection of $\overline{fq_n}$ and $\overline{fq_n}'$
$\phi$	$::=$ $ $ $[\ ]$ $ $ $\phi[md^c \mapsto imp\_dep \mapsto mi]$ $ $ $\phi \setminus md^c$	$R$ cache ( $md^c \rightarrow (imp\_dep \rightarrow mi)$ ) M empty repository's cache M map $imp\_dep$ to $mi$ in map for $md^c$ M remove mapping for $md^c$
$imp\_dep$	$::=$ $ $ <b>Shared</b> $ $ <b>Own</b> $mi$ $ $ <b>As</b> $amn$	import dependency default import instance of imported module ref. to imported module
$MH$	$::=$ $ $ $[\ ]$ $ $ $[mi \mapsto mhv]$ $ $ $MH_1 .. MH_k$	module hierarchy ( $mi \rightarrow mhv$ ) M empty module hierarchy M maps $mi$ to its def. and imports M composes many
$mhv$	$::=$ $ $ $(md, \overline{mibr})$	module hierarchy value ( $md \times \overline{mibr}$ ) M def.
$\overline{mibr}$	$::=$ $ $ $[\ ]$ $ $ $mibr_1 .. mibr_k$	associated boundary renamings ( $mibr$ list) M empty M def.
$mibr$	$::=$ $ $ $mi \ br$	assoc. boundary renaming ( $mi \times br$ ) M def.
$md$	$::=$ $ $ <b>repl module</b> $mn\{\overline{cld} \overline{imp_k^c} \overline{fq_n}\}$	module instance def.

$\overline{cld}$	::=		class def.'s ( <i>cld</i> list)
		$[\ ]$	M empty
		$cld_1 .. cld_k$	M def.
<i>cld</i>	::=		class def.
		<i>pd am</i> <b>class</b> <i>dcl</i> <b>extends</b> $cl\{\overline{fd\ meth\_def}\}$	def.
$\overline{meth\_def}$	::=		method def.'s ( <i>meth_def</i> list)
		$[\ ]$	M empty
		$meth\_def_1 .. meth\_def_k$	M def.
<i>meth_def</i>	::=		method definition
		<i>meth_sig</i> { <i>meth_body</i> }	def.
<i>meth_body</i>	::=		method body
		$s_1 .. s_k$ <b>return</b> <i>y</i> ;	def.
<i>s</i>	::=		statement
		$\{\overline{s_k^k}\}$	block
		<i>var</i> = <i>x</i> ;	variable assignment
		<i>var</i> = <i>x.f</i> ;	field read
		<i>x.f</i> = <i>y</i> ;	field write
		<b>if</b> ( <i>x</i> == <i>y</i> ) <i>s</i> <b>else</b> <i>s'</i>	conditional branch
		<i>var</i> = <b>new</b> <sub><i>ctx</i></sub> <i>cl</i> ();	object creation
		<i>var</i> = <i>x.meth</i> ( $\overline{y}$ );	method call
<i>ctx</i>	::=		context
		<i>mi.pn</i>	def.
$\overline{mi}$	::=		module instance identifiers ( <i>mi</i> list)
		$[\ ]$	M empty
		$mi_1 .. mi_k$	M def.
		$mis\_of(\overline{mibr})$	M module instances of $\overline{mibr}$
$md_{opt}^c$	::=		module def., compile-time code option ( $md^c$ option)
		<b>null</b>	M none
		$md^c$	M some
$\overline{f}$	::=		fields ( <i>f</i> list)
		$[\ ]$	M empty
		$f_1 .. f_k$	M def.
		$\overline{f}; \overline{f}'$	M append
$\overline{f}_{opt}$	::=		fields option ( $\overline{f}$ option)
		<b>null</b>	M none
		$\overline{f}$	M some
$\overline{meth}$	::=		method names ( <i>meth</i> list)
		$[\ ]$	M empty
		$meth_1 .. meth_k$	M def.
		$\overline{meth}; \overline{meth}'$	M append

$meth\_def_{opt}$	$::=$   <b>null</b>   $meth\_def$	M M	method def. option ( $meth\_def$ option) none some
$ctxmeth\_def_{opt}$	$::=$   <b>null</b>   $(ctx, meth\_def)$	M M	method def. in context option ( $(ctx \times meth\_def)$ option) none some
$cld_{opt}$	$::=$   <b>null</b>   $cld$	M M	class def. option ( $cld$ list) none some
$ctxcld$	$::=$   $(ctx, cld)$	M	class def. in context ( $ctx \times cld$ ) def.
$\overline{ctxcld}$	$::=$   []   $ctxcld_1 .. ctxcld_k$   $ctxcld@[ctxcld]$	M M M	class def.'s in context ( $ctxcld$ list) empty def. rev cons
$ctxcld_{opt}$	$::=$   <b>null</b>   $ctxcld$	M M	class def. lookup result ( $ctxcld$ option) none some
$\overline{ctxcld}_{opt}$	$::=$   <b>null</b>   $\overline{ctxcld}$	M M	class def.'s lookup result ( $\overline{ctxcld}$ option) none some
$\overline{pn}$	$::=$   $pn_1 .. pn_k$	M	package names ( $pn$ list) def.
$\overline{m}$	$::=$   $m_1 .. m_k$	M	module names ( $m$ list) def.
$mi_{opt}$	$::=$   <b>null</b>   $mi$   $\phi(md^c, imp\_dep)$	M M M	module instance option ( $mi$ option) none some module instance lookup
$mhv_{opt}$	$::=$   <b>null</b>   $mhv$   $MH(mi)$	M M M	module hierarchy value option ( $mhv$ option) none some lookup
$R_{opt}$	$::=$   <b>null</b>   $R$   $RC(rn)$	M M M	repository option ( $R$ option) none some repository lookup
$rnmd^c_{opt}$	$::=$   <b>null</b>	M	module def., compile-time code lookup value ( $rnmd^c$ option) none

		$(rn, md^c)$	M	some
$Type, \tau$	::=			type
		<b>Object</b>		supertype of all types
		$ctx.dcl$		class identifier
$\tau_{opt}$	::=			result of type lookup ( $\tau$ option)
		<b>null</b>	M	none
		$\tau$	M	some
		$\Gamma(x)$	M	static type lookup
		$H(oid)$	M	dynamic type lookup
$\tau_{opt}^\perp$	::=			result of type lookup that can abort
		$\tau_{opt}$		result of type lookup
		$\perp$		failed to find a type
$\bar{\tau}$	::=			types ( $\tau$ list)
		$\tau_1 .. \tau_k$	M	def.
$\pi$	::=			method type
		$\bar{\tau} \rightarrow \tau$		def.
$\Gamma$	::=			type environment ( $x \rightarrow \tau$ )
		$[x_1 \mapsto \tau_1 .. x_k \mapsto \tau_k]$	M	type mappings
		$\Gamma[x \mapsto \tau]$	M	$\Gamma$ with $x \mapsto \tau$
$\theta$	::=			variable mapping ( $x \rightarrow x$ )
		$[x_1 \mapsto y_1 .. x_k \mapsto y_k]$	M	variable mappings
		$\theta[x \mapsto y]$	M	$\theta$ with $x \mapsto y$
$Val, v, w$	::=			value
		<b>null</b>		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 ( $x \rightarrow v$ )
		$[\ ]$	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 ( $oid \rightarrow (\tau \times (f \rightarrow v))$ )
		$[\ ]$	M	empty heap
		$H[oid \mapsto (\tau, f_1 \mapsto v_1 .. f_k \mapsto v_k)]$	M	$H$ with new $oid$ of type $\tau$
		$H[(oid, f) \mapsto v]$	M	$H$ with $(oid, f) \mapsto v$
$config$	::=			configuration
		$(P, L, H, \bar{s}_k^k)$		normal configuration

		$(P, L, H, Exception)$		exception occurred
$Exception$	::=			exception
		<b>NPE</b>		null-pointer exception
$a$	::=			administrator action
		$rn.install(md^c);$		install
		$rn.uninstall(m);$		uninstall
		$rn.initialise(imp);$		initialise
$ia$	::=			internal action
		$rn.install(md^c)$		install
		$rn.uninstall(m)$		uninstall
		$mi = rn.get_instance(imp)$		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
		<b>size (dom <math>RC</math>)</b>	M	size of domain of $RC$

$\boxed{imp\_name(imp) = m}$  – extract the module name from an import statement

IN\_DEFAULT

IN\_SHARED

IN\_OWN

$\overline{imp\_name(\mathbf{import} m br) = m}$      $\overline{imp\_name(\mathbf{import} \mathbf{shared} m br) = m}$      $\overline{imp\_name(\mathbf{import} \mathbf{own} m br) = m}$   
IN\_AS

$\overline{imp\_name(\mathbf{import} m \mathbf{as} amn br) = m}$

$\boxed{imp\_br(imp) = br}$  – get boundary renaming of an import statement

IB\_DEFAULT

IB\_SHARED

IB\_OWN

$\overline{imp\_br(\mathbf{import} m br) = br}$      $\overline{imp\_br(\mathbf{import} \mathbf{shared} m br) = br}$      $\overline{imp\_br(\mathbf{import} \mathbf{own} m br) = br}$   
IB\_AS

$\overline{imp\_br(\mathbf{import} m \mathbf{as} amn br) = br}$

$\boxed{imp\_dep\_of(md^c, mi, imp) = imp\_dep}$  – generate import dependency from  $md^c$ ,  $mi$  and  $imp$

IDO\_SINGLETON

IDO\_DEFAULT\_SHARED

$\frac{1. md\_repl(md^c) = \mathbf{singleton}}{imp\_dep\_of(md^c, mi, imp) = \mathbf{Shared}}$   
IDO\_DEFAULT\_OWN

$\frac{1. md\_repl(md^c) =}{imp\_dep\_of(md^c, mi, \mathbf{import} m br) = \mathbf{Shared}}$   
IDO\_SHARED

$\frac{1. md\_repl(md^c) = \mathbf{replicating}}{imp\_dep\_of(md^c, mi, \mathbf{import} m br) = \mathbf{Own} mi}$   
IDO\_OWN

$\frac{1. md\_repl(md^c) =}{imp\_dep\_of(md^c, mi, \mathbf{import} \mathbf{shared} m br) = \mathbf{Shared}}$   
IDO\_AS

$\frac{1. md\_repl(md^c) \neq \mathbf{singleton}}{imp\_dep\_of(md^c, mi, \mathbf{import} \mathbf{own} m br) = \mathbf{Own} mi}$

$\frac{1. md\_repl(md^c) \neq \mathbf{singleton}}{imp\_dep\_of(md^c, mi, \mathbf{import} m \mathbf{as} amn br) = \mathbf{As} amn}$

$R\_name(R) = rn$  – extract repository’s name

R\_NAME\_BOOTSTRAP

R\_NAME\_STANDARD

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

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

R\_BODY\_BOOTSTRAP

R\_BODY\_STANDARD

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

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

R\_UPDATE\_BOOTSTRAP

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

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

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

MDS\_RM\_EMPTY

MDS\_RM\_CONS\_TRUE

MDS\_RM\_CONS\_FALSE

$m\mathit{ds\_rm}(, md^c) = \frac{1. m\mathit{ds\_rm}(\overline{md^c}_1, md^c) = \overline{md^c}_2}{m\mathit{ds\_rm}(md^c \# \overline{md^c}_1, md^c) = \overline{md^c}_2}$      $\frac{1. md^c_1 \neq md^c}{2. m\mathit{ds\_rm}(\overline{md^c}_1, md^c) = \overline{md^c}_2}$   
 $m\mathit{ds\_rm}(md^c_1 \# \overline{md^c}_1, md^c) = md^c_1 \# \overline{md^c}_2$

$m\mathit{d\_name}(md^c) = mn$  – extract the module name from a module definition

MD\_NAME

$\frac{1. md^c = \mathit{repl\ module}\ mn\ \{\overline{cld^c}\ \overline{imp}_k^k\ \overline{fqn}\}}{m\mathit{d\_name}(md^c) = mn}$

$f\mathit{ull\_name}(cld) = fq\mathit{n}$  – extract the full name from a class

FULL\_NAME

$f\mathit{ull\_name}(\mathbf{package}\ pn; \mathit{am}\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\ \{\overline{fd}\ \overline{meth\_def}\}) = pn.dcl$

$p\mathit{ackage\_name}(cld) = pn$  – extract the package name from a class

PACKAGE\_NAME

$p\mathit{ackage\_name}(\mathbf{package}\ pn; \mathit{am}\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\ \{\overline{fd}\ \overline{meth\_def}\}) = pn$

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

CLASS\_NAME

$c\mathit{lass\_name}(pd\ \mathit{am}\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\ \{\overline{fd}\ \overline{meth\_def}\}) = dcl$

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

SUPERCLASS\_NAME

$s\mathit{uperclass\_name}(pd\ \mathit{am}\ \mathbf{class}\ dcl\ \mathbf{extends}\ cl\ \{\overline{fd}\ \overline{meth\_def}\}) = cl$

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

CLASS\_FIELDS

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

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

CLASS\_METHODS

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

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

METHOD\_NAME

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

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

DF\_DEF

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

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

FMIM\_EMPTY

FMIM\_CONS\_TRUE

$$\overline{find\_md\_in\_mds(\overline{md^c}, mn) = \mathbf{null}} \quad \frac{\begin{array}{l} 1. \overline{md^c} = \mathbf{repl\ module}\ mn\{\overline{cld^c}\ \overline{imp_k^k}\ \overline{fq_n}\} \\ \overline{find\_md\_in\_mds(md^c\ md_2^c \dots md_k^c, mn) = md^c} \end{array}}{\text{FMIM_CONS_FALSE}}$$

$$\frac{\begin{array}{l} 1. \overline{md^c} = \mathbf{repl\ module}\ mn'\{\overline{cld^c}\ \overline{imp_k^k}\ \overline{fq_n}\} \\ 2. mn \neq mn' \\ 3. \overline{find\_md\_in\_mds(md_2^c \dots md_k^c, mn) = md_{opt}^c} \end{array}}{\overline{find\_md\_in\_mds(md^c\ md_2^c \dots md_k^c, mn) = md_{opt}^c}}$$

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

FMR\_NULL

FMR\_BOOTSTRAP\_NULL

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

FMR\_BOOTSTRAP

FMR\_STANDARD\_FAIL

$$\frac{\begin{array}{l} 1. RC(rn) = \mathbf{bootstrap\ repository}\ \{\overline{md^c}; \phi\} \\ 2. \overline{find\_md\_in\_mds(\overline{md^c}, mn) = md^c} \\ \overline{find\_md\_rec(RC, rn, mn, nn) = (rn, md^c)} \end{array}}{\text{FMR_STANDARD_REC}} \quad \frac{\begin{array}{l} 1. RC(rn_1) = \mathbf{repository}\ r\ \mathbf{child\ of}\ rn_2\ \{\overline{md^c}; \phi\} \\ 2. \mathbf{size}(\mathbf{dom}\ RC) \leq nn \\ \overline{find\_md\_rec(RC, rn_1, mn, nn) = \mathbf{null}} \end{array}}{\text{FMR_STANDARD_SELF}}$$

FMR\_STANDARD\_REC

FMR\_STANDARD\_SELF

$$\frac{\begin{array}{l} 1. RC(rn_1) = \mathbf{repository}\ r\ \mathbf{child\ of}\ rn_2\ \{\overline{md^c}; \phi\} \\ 2. \mathbf{size}(\mathbf{dom}\ RC) > nn \\ 3. \overline{find\_md\_rec(RC, rn_2, mn, nn+1) = (rn_3, md^c)} \\ \overline{find\_md\_rec(RC, rn_1, mn, nn) = (rn_3, md^c)} \end{array}}{\text{FMR_STANDARD_SELF}} \quad \frac{\begin{array}{l} 1. RC(rn_1) = \mathbf{repository}\ r\ \mathbf{child\ of}\ rn_2\ \{\overline{md^c}; \phi\} \\ 2. \mathbf{size}(\mathbf{dom}\ RC) > nn \\ 3. \overline{find\_md\_rec(RC, rn_2, mn, nn+1) = \mathbf{null}} \\ 4. \overline{find\_md\_in\_mds(\overline{md^c}, mn) = md^c} \\ \overline{find\_md\_rec(RC, rn_1, mn, nn) = (rn_1, md^c)} \end{array}}{\text{FMR_STANDARD_SELF}}$$

FMR\_STANDARD\_NULL

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

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

FM\_DEF

1.  $\text{find\_md\_rec}(RC, rn, mn, 0) = rnmd_{opt}^c$
- 
- $\text{find\_md}(RC, rn, mn) = rnmd_{opt}^c$

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

FCIM\_EMPTY

FCIM\_NULL

1.  $\neg \text{distinct\_fqns}(cld \ cld_2 \dots cld_k)$
- 
- $\text{find\_cld\_in\_module}(\overline{[]}, fq_n) = \text{null}$        $\text{find\_cld\_in\_module}(cld \ cld_2 \dots cld_k, fq_n) = \text{null}$
- FCIM\_CONS\_TRUE

1.  $\text{distinct\_fqns}(cld \ cld_2 \dots cld_k)$
  2.  $cld = \text{package } pn; \text{ public class } dcl \text{ extends } cl\{\overline{fd \ meth\_def}\}$
- 
- $\text{find\_cld\_in\_module}(cld \ cld_2 \dots cld_k, pn.dcl) = cld$
- FCIM\_CONS\_FALSE

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

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

FCIC\_NO\_REP\_EX

FCIC\_NOT\_BOOTSTRAP\_EX

1.  $RC(\text{bootstrap}_r) = \text{null}$
- 
- $\text{find\_cld\_in\_core}((RC, MH), fq_n) = \text{null}$
1.  $RC(\text{bootstrap}_r) = \text{repository } r \text{ child of } rn\{\overline{md^c}; \phi\}$
- 
- $\text{find\_cld\_in\_core}((RC, MH), fq_n) = \text{null}$
- FCIC\_NO\_CORE\_EX

1.  $RC(\text{bootstrap}_r) = \text{bootstrap repository } \{\overline{md^c}; \phi\}$
  2.  $\text{find\_md\_in\_mds}(\overline{md^c}, \text{core}_m) = \text{null}$
- 
- $\text{find\_cld\_in\_core}((RC, MH), fq_n) = \text{null}$
- FCIC\_NO\_CORE\_MI\_EX

1.  $RC(\text{bootstrap}_r) = \text{bootstrap repository } \{\overline{md^c}; \phi\}$
  2.  $\text{find\_md\_in\_mds}(\overline{md^c}, \text{core}_m) = md^c$
  3.  $\phi(md^c, \text{Shared}) = \text{null}$
- 
- $\text{find\_cld\_in\_core}((RC, MH), fq_n) = \text{null}$
- FCIC\_NO\_MDMIS\_EX

1.  $RC(\text{bootstrap}_r) = \text{bootstrap repository } \{\overline{md^c}; \phi\}$
  2.  $\text{find\_md\_in\_mds}(\overline{md^c}, \text{core}_m) = md^c$
  3.  $\phi(md^c, \text{Shared}) = mi$     4.  $MH(mi) = \text{null}$
- 
- $\text{find\_cld\_in\_core}((RC, MH), fq_n) = \text{null}$
- FCIC\_FALSE

1.  $RC(\text{bootstrap}_r) = \text{bootstrap repository } \{\overline{md^c}; \phi\}$
  2.  $\text{find\_md\_in\_mds}(\overline{md^c}, \text{core}_m) = md^c$
  3.  $\phi(md^c, \text{Shared}) = mi$
  4.  $MH(mi) = (\text{repl module } mn\{\overline{cld \ imp_k^k \ fq_n}, \overline{mibr}\})$
  5.  $\text{find\_cld\_in\_module}(\overline{cld}, fq_n) = \text{null}$
- 
- $\text{find\_cld\_in\_core}((RC, MH), fq_n) = \text{null}$

1.  $RC(\text{bootstrap}_r) = \mathbf{bootstrap\ repository} \{ \overline{md^c}; \phi \}$
  2.  $\text{find\_md\_in\_mds}(\overline{md^c}, \text{core}_m) = \overline{md^c}$
  3.  $\phi(\overline{md^c}, \mathbf{Shared}) = mi$
  4.  $MH(mi) = (\text{repl\ module } mn\{ \overline{cld\ imp}_k^k\ \overline{fq_n}, \overline{mibr} \})$
  5.  $\text{find\_cld\_in\_module}(\overline{cld}, \overline{fq_n}) = \overline{cld}$
  6.  $\text{package\_name}(\overline{cld}) = pn$
- 
- $$\text{find\_cld\_in\_core}((RC, MH), \overline{fq_n}) = (mi.pn, \overline{cld})$$

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

- REACHABLE\_EMPTY
1.  $MH(mi) = (md, \overline{mibr})$
  2.  $(MH, \text{mis\_of}(\overline{mibr}), nn') \in \mathbf{reachable}$
  3.  $(MH, mi_2 .. mi_k, nn) \in \mathbf{reachable}$
- 
- $$(MH, [ ], 0) \in \mathbf{reachable}$$
- 
- $$(MH, mi_2 .. mi_k, nn' + nn + 1) \in \mathbf{reachable}$$

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

1.  $\mathbf{finite}(\mathbf{dom}(MH))$
  2.  $\forall \overline{mi}, \overline{mi}' \subseteq \mathbf{dom}(MH) \longrightarrow (\exists nn. (MH, \overline{mi}, nn) \in \mathbf{reachable})$
  3.  $\forall mi \in \mathbf{dom}(MH). \exists md\ \overline{mibr}. MH(mi) = (md, \overline{mibr}) \wedge \text{mis\_of}(\overline{mibr}) \subseteq \mathbf{dom}(MH)$
- 
- $$\mathbf{acyclic\_mhMH}$$

$\text{find\_cld\_in\_imports}(MH, \overline{mibr}, \overline{fq_n}) = \text{ctxcld}_{opt}$  – class lookup in imports

- FCII\_EMPTY
1.  $\neg \left( \begin{array}{l} \mathbf{acyclic\_mhMH} \wedge mi \in \mathbf{dom}(MH) \wedge \\ \text{mis\_of}(mibr_2 .. mibr_k) \subseteq \mathbf{dom}(MH) \end{array} \right)$
- 
- $$\text{find\_cld\_in\_imports}(MH, [ ], \overline{fq_n}) = \mathbf{null}$$
- 
- $$\text{find\_cld\_in\_imports}(MH, mi\ br\ mibr_2 .. mibr_k, \overline{fq_n}) = \mathbf{null}$$
- FCII\_SKIP

1.  $\left( \begin{array}{l} \mathbf{acyclic\_mhMH} \wedge MH(mi) = (md, \overline{mibr}) \wedge \\ \text{mis\_of}(mibr_2 .. mibr_k) \subseteq \mathbf{dom}(MH) \end{array} \right)$
  2.  $\left( \begin{array}{l} (md = \text{repl\ module } mn\{ \overline{cld\ imp}_j^j\ \overline{fq_n} \} \wedge br[\overline{fq_n}] \notin \overline{fq_n}) \vee \\ (fq_n \notin \mathbf{dom}(br) \wedge fq_n \in \mathbf{ran}(br)) \end{array} \right)$
  3.  $\text{find\_cld\_in\_imports}(MH, mibr_2 .. mibr_k, \overline{fq_n}) = \text{ctxcld}_{opt}$
- 
- $$\text{find\_cld\_in\_imports}(MH, mi\ br\ mibr_2 .. mibr_k, \overline{fq_n}) = \text{ctxcld}_{opt}$$
- FCII\_SELF

1.  $\left( \begin{array}{l} \mathbf{acyclic\_mhMH} \wedge MH(mi) = (md, \overline{mibr}) \wedge \\ \text{mis\_of}(mibr_2 .. mibr_k) \subseteq \mathbf{dom}(MH) \end{array} \right)$
  2.  $\left( \begin{array}{l} (md = \text{repl\ module } mn\{ \overline{cld\ imp}_j^j\ \overline{fq_n} \} \wedge br[\overline{fq_n}] \in \overline{fq_n}) \wedge \\ (fq_n \in \mathbf{dom}(br) \vee fq_n \notin \mathbf{ran}(br)) \end{array} \right)$
  3.  $\text{find\_cld\_in\_module}(\overline{cld}, br[\overline{fq_n}]) = \overline{cld} \wedge \text{package\_name}(\overline{cld}) = pn$
- 
- $$\text{find\_cld\_in\_imports}(MH, mi\ br\ mibr_2 .. mibr_k, \overline{fq_n}) = (mi.pn, \overline{cld})$$
- FCII\_REC

1.  $\left( \begin{array}{l} \mathbf{acyclic\_mhMH} \wedge MH(mi) = (md, \overline{mibr}) \wedge \\ \text{mis\_of}(mibr_2 .. mibr_k) \subseteq \mathbf{dom}(MH) \end{array} \right)$
  2.  $\left( \begin{array}{l} (md = \text{repl\ module } mn\{ \overline{cld\ imp}_j^j\ \overline{fq_n} \} \wedge br[\overline{fq_n}] \in \overline{fq_n}) \wedge \\ (fq_n \in \mathbf{dom}(br) \vee fq_n \notin \mathbf{ran}(br)) \end{array} \right)$
  3.  $\text{find\_cld\_in\_module}(\overline{cld}, br[\overline{fq_n}]) = \mathbf{null}$
  4.  $\text{find\_cld\_in\_imports}(MH, \overline{mibr}, br[\overline{fq_n}]) = \text{ctxcld}$
- 
- $$\text{find\_cld\_in\_imports}(MH, mi\ br\ mibr_2 .. mibr_k, \overline{fq_n}) = \text{ctxcld}$$

$$\begin{array}{l}
1. \left( \begin{array}{l} \text{acyclic\_mh} MH \wedge MH(mi) = (md, \overline{mibr}) \wedge \\ \text{mis\_of}(mibr_2 .. mibr_k) \subseteq \mathbf{dom}(MH) \end{array} \right) \\
2. \left( \begin{array}{l} (md = \text{repl module } mn\{\overline{cld\_imp_j^j} \overline{fq_n}\} \wedge br[fqn] \in \overline{fq_n}) \wedge \\ (fq_n \in \mathbf{dom}(br) \vee fq_n \notin \mathbf{ran}(br)) \end{array} \right) \\
3. \text{find\_cld\_in\_module}(\overline{cld}, br[fqn]) = \mathbf{null} \\
4. \text{find\_cld\_in\_imports}(MH, \overline{mibr}, br[fqn]) = \mathbf{null} \\
5. \text{find\_cld\_in\_imports}(MH, mibr_2 .. mibr_k, fq_n) = \text{ctxcld}_{opt} \\
\hline
\text{find\_cld\_in\_imports}(MH, mi br mibr_2 .. mibr_k, fq_n) = \text{ctxcld}_{opt}
\end{array}$$

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

FCIS\_EMPTY

FCIS\_NULL

$$\frac{}{\text{find\_cld\_in\_self}([], pn, fq_n) = \mathbf{null}} \quad \frac{1. \neg \text{distinct\_fqns}(\text{cld } cld_2 .. cld_k)}{\text{find\_cld\_in\_self}(\text{cld } cld_2 .. cld_k, pn, fq_n) = \mathbf{null}}$$

FCIS\_CONS\_TRUE

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

FCIS\_CONS\_FALSE

$$\begin{array}{l}
1. \text{distinct\_fqns}(\text{cld } cld_2 .. cld_k) \\
2. \text{cld} = \mathbf{package } pn''; \text{ am class } dcl' \text{ extends } cl\{\overline{fd} \overline{meth\_def}\} \\
3. (pn \neq pn' \wedge am \neq \mathbf{public}) \vee pn' \neq pn'' \vee dcl \neq dcl' \\
4. \text{find\_cld\_in\_self}(\text{cld}_2 .. \text{cld}_k, pn, pn'.dcl) = \text{cld}_{opt} \\
\hline
\text{find\_cld\_in\_self}(\text{cld } cld_2 .. \text{cld}_k, pn, pn'.dcl) = \text{cld}_{opt}
\end{array}$$

$\boxed{\text{no\_core\_renaming\_in\_mibrs}(P, \overline{mibr})}$  – there is no renaming of core classes in  $\overline{mibr}$

NCRIM\_DEF

$$\frac{1. \forall br \in br_1 .. br_k. \forall fq_n. \left( \begin{array}{l} (\exists \text{ctx } cld. \text{find\_cld\_in\_core}(P, fq_n) = (\text{ctx}, cld)) \longrightarrow \\ fq_n \notin br \end{array} \right)}{\text{no\_core\_renaming\_in\_mibrs}(P, mi_1 br_1 .. mi_k br_k)}$$

$\boxed{\text{no\_core\_renaming}P}$  – there is no renaming of core classes in  $P$

NCR\_DEF

$$\frac{1. \forall mi \in \mathbf{dom}(MH). \exists md \overline{mibr}. \left( \begin{array}{l} MH(mi) = (md, \overline{mibr}) \wedge \\ \text{no\_core\_renaming\_in\_mibrs}((RC, MH), \overline{mibr}) \end{array} \right)}{\text{no\_core\_renaming}(RC, MH)}$$

$\boxed{\text{find\_cld}(P, \text{ctx}, fq_n) = \text{ctxcld}_{opt}}$  – class lookup

FC\_ERR

FC\_CORE

$$\frac{1. \neg \text{no\_core\_renaming}P}{\text{find\_cld}(P, \text{ctx}, fq_n) = \mathbf{null}} \quad \frac{1. \text{no\_core\_renaming}P \\ 2. \text{find\_cld\_in\_core}(P, fq_n) = \text{ctxcld}}{\text{find\_cld}(P, \text{ctx}, fq_n) = \text{ctxcld}}$$

FC\_SELF

$$\begin{array}{l}
\text{FC_NULL} \\
1. \text{no\_core\_renaming}(RC, MH) \\
2. \text{find\_cld\_in\_core}((RC, MH), fq_n) = \mathbf{null} \\
3. MH(mi) = \mathbf{null} \\
\hline
\text{find\_cld}((RC, MH), mi.pn, fq_n) = \mathbf{null}
\end{array}
\quad
\begin{array}{l}
1. \text{no\_core\_renaming}(RC, MH) \\
2. \text{find\_cld\_in\_core}((RC, MH), fq_n) = \mathbf{null} \\
3. MH(mi) = (md, \overline{mibr}) \\
4. md = \text{repl module } mn\{\overline{cld\_imp_k^k} \overline{fq_n}\} \\
5. \text{find\_cld\_in\_self}(\overline{cld}, pn, fq_n) = \text{cld} \\
6. \text{package\_name}(\overline{cld}) = pn' \\
\hline
\text{find\_cld}((RC, MH), mi.pn, fq_n) = (mi.pn', \overline{cld})
\end{array}$$

1.  $no\_core\_renaming(RC, MH)$
2.  $find\_cld\_in\_core((RC, MH), fq_n) = \mathbf{null}$
3.  $MH(mi) = (md, \overline{mibr})$
4.  $md = repl \mathbf{module} mn \{ \overline{cld\_imp_k} \overline{fq_n} \}$
5.  $find\_cld\_in\_self(\overline{cld}, pn, fq_n) = \mathbf{null}$
6.  $\frac{find\_cld\_in\_imports(MH, \overline{mibr}, fq_n) = \overline{ctxcld\_opt}}{find\_cld((RC, MH), mi.pn, fq_n) = \overline{ctxcld\_opt}}$

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

$$\frac{}{find\_type(P, ctx, \mathbf{Object}) = \mathbf{Object}} \quad \text{FT\_OBJ} \qquad \frac{}{find\_type(P, ctx, fq_n) = \mathbf{null}} \quad \text{FT\_NULL} \qquad \frac{1. find\_cld(P, ctx, pn.dcl) = (ctx', cld) \quad 2. class\_name(cld) = dcl'}{find\_type(P, ctx, pn.dcl) = ctx'.dcl'} \quad \text{FT\_DCL}$$

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

$$\frac{}{(P, ctx, \mathbf{Object}, 0) \in path\_length} \quad \text{PL\_OBJ} \qquad \frac{1. find\_cld(P, ctx, fq_n) = (ctx', cld) \quad 2. superclass\_name(cld) = cl \quad 3. (P, ctx', cl, nn) \in path\_length}{(P, ctx, fq_n, nn+1) \in path\_length} \quad \text{PL\_FQN}$$

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

$$\frac{1. \forall pn fq_n. (\exists ctx' cld. find\_cld(P, mi.pn, fq_n) = (ctx', cld)) \longrightarrow \exists nn. (P, mi.pn, fq_n, nn) \in path\_length}{acyclic\_clds_{mi}P} \quad \text{ACM\_DEF}$$

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

$$\frac{1. \forall mi. acyclic\_clds_{mi}P}{acyclic\_cldsP} \quad \text{AC\_DEF}$$

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

$$\frac{}{find\_path\_rec(P, ctx, \mathbf{Object}, \overline{ctxcld}) = \overline{ctxcld}} \quad \text{FPR\_OBJ} \qquad \frac{1. (\neg acyclic\_cldsP) \vee find\_cld(P, ctx, fq_n) = \mathbf{null}}{find\_path\_rec(P, ctx, fq_n, \overline{ctxcld}) = \mathbf{null}} \quad \text{FPR\_NULL}$$

$$\frac{1. acyclic\_cldsP \wedge find\_cld(P, ctx, fq_n) = (ctx', cld) \quad 2. superclass\_name(cld) = cl \quad 3. find\_path\_rec(P, ctx', cl, \overline{ctxcld} @ [(ctx', cld)]) = \overline{ctxcld\_opt}}{find\_path\_rec(P, ctx, fq_n, \overline{ctxcld}) = \overline{ctxcld\_opt}} \quad \text{FPR\_FQN}$$

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

$$\frac{1. find\_path\_rec(P, ctx, cl, []) = \overline{ctxcld\_opt}}{find\_path(P, ctx, cl) = \overline{ctxcld\_opt}} \quad \text{FP\_DEF}$$

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

$$\frac{\text{FPTY\_OBJ}}{\overline{find\_path(P, \text{Object}) = []}} \quad \frac{\text{FPTY\_DCL} \quad 1. \overline{find\_path(P, mi.pn, pn.dcl) = \overline{ctxcld}_{opt}}}{\overline{find\_path(P, mi.pn.dcl) = \overline{ctxcld}_{opt}}}$$

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

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

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

$$\frac{\text{FIELDS\_NONE} \quad 1. \overline{find\_path(P, \tau) = \text{null}}}{\overline{fields(P, \tau) = \text{null}}} \quad \frac{\text{FIELDS\_SOME} \quad \begin{array}{l} 1. \overline{find\_path(P, \tau) = \overline{ctxcld}} \\ 2. \overline{fields\_in\_path(\overline{ctxcld}) = \overline{f}} \end{array}}{\overline{fields(P, \tau) = \overline{f}}}$$

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

$$\frac{\text{MIP\_EMPTY}}{\overline{methods\_in\_path([]) = []}} \quad \frac{\text{MIP\_CONS} \quad \begin{array}{l} 1. \overline{class\_methods(cld) = \overline{meth\_def_l^l}} \\ 2. \overline{meth\_def_l = cl_l meth_l(\overline{vd_l}) \{ meth\_body_l \}} \\ 3. \overline{methods\_in\_path(cld_2 .. cld_k) = \overline{meth}'} \\ 4. \overline{meth = \overline{meth_l^l}; \overline{meth}'} \end{array}}{\overline{methods\_in\_path(cld cld_2 .. cld_k) = \overline{meth}}}$$

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

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

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

$$\frac{\text{FTIF\_EMPTY}}{\overline{ftype\_in\_fds(P, ctx, [], f) = \text{null}}} \quad \frac{\text{FTIF\_CONS\_BOT} \quad 1. \overline{find\_type(P, ctx, cl) = \text{null}}}{\overline{ftype\_in\_fds(P, ctx, cl f; fd_2 .. fd_k, f) = \perp}} \quad \frac{\text{FTIF\_CONS\_TRUE} \quad 1. \overline{find\_type(P, ctx, cl) = \tau}}{\overline{ftype\_in\_fds(P, ctx, cl f; fd_2 .. fd_k, f) = \tau}} \quad \frac{\text{FTIF\_CONS\_FALSE} \quad \begin{array}{l} 1. f \neq f' \\ 2. \overline{ftype\_in\_fds(P, ctx, fd_2 .. fd_k, f')} = \tau_{opt}^\perp \end{array}}{\overline{ftype\_in\_fds(P, ctx, cl f; fd_2 .. fd_k, f) = \tau_{opt}^\perp}}$$

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

$$\begin{array}{c}
\text{FTIP\_EMPTY} \\
\frac{}{f\text{type\_in\_path}(P, [], f) = \mathbf{null}} \\
\text{FTIP\_CONS\_TRUE} \\
\frac{1. \text{class\_fields}(cld) = \overline{fd} \\ 2. f\text{type\_in\_fds}(P, ctx, \overline{fd}, f) = \perp}{f\text{type\_in\_path}(P, (ctx, cld) \text{ ctxcld}_2 .. \text{ctxcld}_k, f) = \mathbf{null}} \\
\text{FTIP\_CONS\_FALSE} \\
\frac{1. \text{class\_fields}(cld) = \overline{fd} \\ 2. f\text{type\_in\_fds}(P, ctx, \overline{fd}, f) = \mathbf{null} \\ 3. f\text{type\_in\_path}(P, \text{ctxcld}_2 .. \text{ctxcld}_k, f) = \tau_{opt}}{f\text{type\_in\_path}(P, (ctx, cld) \text{ ctxcld}_2 .. \text{ctxcld}_k, f) = \tau_{opt}}
\end{array}$$

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

$$\begin{array}{c}
\text{FTYPE} \\
\frac{1. f\text{ind\_path}(P, \tau) = \overline{ctxcld} \\ 2. f\text{type\_in\_path}(P, \overline{ctxcld}, f) = \tau'}{f\text{type}(P, \tau, f) = \tau'}
\end{array}$$

$\boxed{f\text{ind\_meth\_def\_in\_list}(\text{meth\_def}, \text{meth}) = \text{meth\_def}_{opt}}$  – meth. def. lookup (list)

$$\begin{array}{c}
\text{FMDIL\_EMPTY} \\
\frac{}{f\text{ind\_meth\_def\_in\_list}([], \text{meth}) = \mathbf{null}} \\
\text{FMDIL\_CONS\_TRUE} \\
\frac{1. \text{meth\_def} = cl \text{ meth}(\overline{vd})\{\text{meth\_body}\}}{f\text{ind\_meth\_def\_in\_list}(\text{meth\_def} \text{ meth\_def}_2 .. \text{meth\_def}_k, \text{meth}) = \text{meth\_def}} \\
\text{FMDIL\_CONS\_FALSE} \\
\frac{1. \text{meth\_def} = cl \text{ meth}'(\overline{vd})\{\text{meth\_body}\} \quad 2. \text{meth} \neq \text{meth}' \\ 3. f\text{ind\_meth\_def\_in\_list}(\text{meth\_def}_2 .. \text{meth\_def}_k, \text{meth}) = \text{meth\_def}_{opt}}{f\text{ind\_meth\_def\_in\_list}(\text{meth\_def} \text{ meth\_def}_2 .. \text{meth\_def}_k, \text{meth}) = \text{meth\_def}_{opt}}
\end{array}$$

$\boxed{f\text{ind\_meth\_def\_in\_path}(\overline{ctxcld}, \text{meth}) = \text{ctxmeth\_def}_{opt}}$  – meth. def. lookup (path)

$$\begin{array}{c}
\text{FMDIP\_EMPTY} \\
\frac{}{f\text{ind\_meth\_def\_in\_path}([], \text{meth}) = \mathbf{null}} \\
\text{FMDIP\_CONS\_TRUE} \\
\frac{1. \text{class\_methods}(cld) = \overline{\text{meth\_def}} \\ 2. f\text{ind\_meth\_def\_in\_list}(\text{meth\_def}, \text{meth}) = \text{meth\_def}}{f\text{ind\_meth\_def\_in\_path}((ctx, cld) \text{ ctxcld}_2 .. \text{ctxcld}_k, \text{meth}) = (ctx, \text{meth\_def})} \\
\text{FMDIP\_CONS\_FALSE} \\
\frac{1. \text{class\_methods}(cld) = \overline{\text{meth\_def}} \\ 2. f\text{ind\_meth\_def\_in\_list}(\text{meth\_def}, \text{meth}) = \mathbf{null} \\ 3. f\text{ind\_meth\_def\_in\_path}(\text{ctxcld}_2 .. \text{ctxcld}_k, \text{meth}) = \text{ctxmeth\_def}_{opt}}{f\text{ind\_meth\_def\_in\_path}((ctx, cld) \text{ ctxcld}_2 .. \text{ctxcld}_k, \text{meth}) = \text{ctxmeth\_def}_{opt}}
\end{array}$$

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

$$\begin{array}{c}
\text{FMD\_NULL} \\
\frac{1. f\text{ind\_path}(P, \tau) = \mathbf{null}}{f\text{ind\_meth\_def}(P, \tau, \text{meth}) = \mathbf{null}} \\
\text{FMD\_OPT} \\
\frac{1. f\text{ind\_path}(P, \tau) = \overline{ctxcld} \\ 2. f\text{ind\_meth\_def\_in\_path}(\overline{ctxcld}, \text{meth}) = \text{ctxmeth\_def}_{opt}}{f\text{ind\_meth\_def}(P, \tau, \text{meth}) = \text{ctxmeth\_def}_{opt}}
\end{array}$$

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

MTYPE

1.  $\text{find\_meth\_def}(P, \tau, \text{meth}) = (\text{ctx}, \text{meth\_def})$
  2.  $\text{meth\_def} = \text{cl } \text{meth}(\overline{\text{cl}_k \text{ var}_k^k})\{\text{meth\_body}\}$
  3.  $\text{find\_type}(P, \text{ctx}, \text{cl}) = \tau'$
  4.  $\text{find\_type}(P, \text{ctx}, \text{cl}_k) = \tau_k$
  5.  $\pi = \overline{\tau_k^k} \rightarrow \tau'$
- 
- $\text{mtype}(P, \tau, \text{meth}) = \pi$

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

STY\_DCL

- |   |  |
|---|--|
| <p>STY_OBJ</p> $\frac{1. \text{find\_path}(P, \tau) = \overline{\text{ctxcld}}}{P \vdash \tau \prec \text{Object}}$ | <p>1. <math>\text{find\_path}(P, \tau) = \overline{\text{ctxcld}}</math><br/> 2. <math>\text{find\_cld}(P, \text{mi}'.\text{pn}', \text{pn}'.\text{dcl}') = \text{ctxcld}</math><br/> 3. <math>\text{ctxcld} \in \overline{\text{ctxcld}}</math></p> <hr/> <p><math>P \vdash \tau \prec \text{mi}'.\text{pn}'.\text{dcl}'</math></p> |
|---|--|

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

STY\_MANY

1.  $\bar{\tau} = \overline{\tau_k^k}$
  2.  $\bar{\tau}' = \overline{\tau'_k^k}$
  3.  $\overline{P \vdash \tau_k \prec \tau'_k}$
- 
- $P \vdash \bar{\tau} \prec \bar{\tau}'$

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

STY\_OPTION

1.  $\tau_{\text{opt}} = \tau$
  2.  $\tau'_{\text{opt}} = \tau'$
  3.  $P \vdash \tau \prec \tau'$
- 
- $P \vdash \tau_{\text{opt}} \prec \tau'_{\text{opt}}$

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

- |  |   |
|--|---|
| <p>WF_NULL</p> $\frac{1. \tau_{\text{opt}} = \tau}{P, H \vdash \text{null} \prec \tau_{\text{opt}}}$ | <p>WF_OBJECT</p> $\frac{1. P \vdash H(\text{oid}) \prec \tau_{\text{opt}}}{P, H \vdash \text{oid} \prec \tau_{\text{opt}}}$ |
|--|---|

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

WF\_VARSTATE

1. **finite** ( $\text{dom}(L)$ )
  2.  $\forall x \in \text{dom}(\Gamma). P, H \vdash L(x) \prec \Gamma(x)$
- 
- $P, \Gamma, H \vdash L$

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

WF\_HEAP

1. **finite** ( $\text{dom}(H)$ )
  2.  $\forall \text{oid} \in \text{dom}(H). \left( \begin{array}{l} \exists \tau. H(\text{oid}) = \tau \wedge \exists \bar{f}. \text{fields}(P, \tau) = \bar{f} \wedge \\ \forall f \in \bar{f}. \exists \tau'. \left( \begin{array}{l} \text{ftype}(P, \tau, f) = \tau' \wedge \\ P, H \vdash H(\text{oid}, f) \prec \tau' \end{array} \right) \end{array} \right)$
- 
- $P \vdash H$

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

$$\begin{array}{c}
\text{WF\_ALL\_EX} \\
\frac{1. \vdash P \quad 2. P \vdash H \quad 3. P, \Gamma, H \vdash L}{\Gamma \vdash (P, L, H, \text{Exception})} \\
\text{WF\_ALL} \\
\frac{1. \vdash P \quad 2. P \vdash H \quad 3. P, \Gamma, H \vdash L \quad 4. \overline{P, \Gamma \vdash s_k^k}}{\Gamma \vdash (P, L, H, \overline{s_k^k})}
\end{array}$$

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

$$\begin{array}{c}
\text{WF\_BLOCK} \\
\frac{1. \overline{P, \Gamma \vdash s_k^k}}{P, \Gamma \vdash \{ \overline{s_k^k} \}} \\
\text{WF\_VAR\_ASSIGN} \\
\frac{1. P \vdash \Gamma(x) \prec \Gamma(\text{var})}{P, \Gamma \vdash \text{var} = x;} \\
\text{WF\_FIELD\_READ} \\
\frac{1. \Gamma(x) = \tau \quad 2. \mathbf{ftype}(P, \tau, f) = \tau' \quad 3. P \vdash \tau' \prec \Gamma(\text{var})}{P, \Gamma \vdash \text{var} = x.f;} \\
\text{WF\_FIELD\_WRITE} \\
\frac{1. \Gamma(x) = \tau \quad 2. \mathbf{ftype}(P, \tau, f) = \tau' \quad 3. P \vdash \Gamma(y) \prec \tau'}{P, \Gamma \vdash x.f = y;} \\
\text{WF\_IF} \\
\frac{1. P \vdash \Gamma(x) \prec \Gamma(y) \vee P \vdash \Gamma(y) \prec \Gamma(x) \quad 2. P, \Gamma \vdash s_1 \quad 3. P, \Gamma \vdash s_2}{P, \Gamma \vdash \mathbf{if}(x == y) s_1 \mathbf{else} s_2} \\
\text{WF\_NEW} \\
\frac{1. \text{find\_type}(P, \text{ctx}, \text{cl}) = \tau \quad 2. P \vdash \tau \prec \Gamma(\text{var})}{P, \Gamma \vdash \text{var} = \mathbf{new}_{\text{ctx}} \text{cl}();} \\
\text{WF\_MCCALL} \\
\frac{1. \overline{y} = \overline{y_k^k} \quad 2. \Gamma(x) = \tau \quad 3. \mathbf{mtype}(P, \tau, \text{meth}) = \overline{\tau_k^k} \rightarrow \tau' \quad 4. \overline{P \vdash \Gamma(y_k) \prec \tau_k^k} \quad 5. P \vdash \tau' \prec \Gamma(\text{var})}{P, \Gamma \vdash \text{var} = x.\text{meth}(\overline{y});}
\end{array}$$

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

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

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

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

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

$$\begin{array}{c}
\text{WF\_CLASS} \\
\frac{1. P \vdash_{mi.pn} (\text{dcl}, \text{cl}, \overline{\text{fd}}, \overline{\text{meth\_def}})}{P \vdash_{mi} \mathbf{package} \text{pn}; \mathbf{am} \mathbf{class} \text{dcl} \mathbf{extends} \text{cl} \{ \overline{\text{fd}} \overline{\text{meth\_def}} \}}
\end{array}$$

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

WF\_MODULE

$$\frac{\begin{array}{l} 1. \overline{full\_name(cld_j)} = \overline{fq_n^j} \quad 2. \mathbf{distinct}(\overline{fq_n^j}) \\ 3. (RC, MH) \vdash_{mi} \overline{cld_j^j} \quad 4. \mathit{acyclic\_clds}_{mi}(RC, MH) \\ 5. MH(mi) = (md, \overline{mibr}) \wedge \mathit{no\_core\_renaming\_in\_mibrs}((RC, MH), \overline{mibr}) \end{array}}{(RC, MH) \vdash_{mi} \mathbf{repl\ module} mn\{\overline{cld_j^j} \overline{imp_k^k} \overline{fq_n^j}\}}$$

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

WF\_RMIS

$$\frac{1. \forall md^c \mathit{imp\_dep}. \forall mi. \phi(md^c, \mathit{imp\_dep}) = mi \longrightarrow mi \in \mathbf{dom}(MH)}{MH \vdash \phi}$$

$P \vdash R$  – well-formed repository

WF\_BOOTSTRAP\_REP

WF\_NORMAL\_REP

$$\frac{\begin{array}{l} 1. \mathit{find\_md\_in\_mds}(\overline{md^c}, \mathit{core\_m}) = md^c \\ 2. MH \vdash \phi \end{array}}{(RC, MH) \vdash \mathbf{bootstrap\ repository} \{\overline{md^c}; \phi\}} \quad \frac{\begin{array}{l} 1. r \neq rn \quad 2. rn \in \mathbf{dom}(RC) \\ 3. MH \vdash \phi \end{array}}{(RC, MH) \vdash \mathbf{repository} r \mathbf{child\ of} rn\{\overline{md^c}; \phi\}}$$

$MH \vdash RC$  – well-formed repository context

WF\_RC

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

$RC \vdash MH$  – well-formed module hierarchy

WF\_MH

$$\frac{\begin{array}{l} 1. \mathit{acyclic\_mh} MH \\ 2. \forall mi \in \mathbf{dom}(MH). \exists md \overline{mibr}. MH(mi) = (md, \overline{mibr}) \wedge (RC, MH) \vdash_{mi} md \end{array}}{RC \vdash MH}$$

$\vdash P$  – well-formed program

WF\_P

$$\frac{\begin{array}{l} 1. MH \vdash RC \\ 2. RC \vdash MH \\ 3. \mathit{acyclic\_clds}(RC, MH) \end{array}}{\vdash (RC, MH)}$$

$\overline{find\_pkg\_clds(cld^c_1, \overline{pn})} = \overline{cld^c_2}$  –

FPC\_CONS\_TRUE

FPC\_EMPTY

$$\frac{\overline{find\_pkg\_clds(\overline{pn})} = \begin{array}{l} 1. cld^c = \mathbf{package} pn; \mathit{am\ class} dcl \mathbf{extends} cl\{\overline{fd} \overline{meth\_def^c}\} \\ 2. pn \in \overline{pn} \\ 3. \overline{find\_pkg\_clds}(cld^c_2 .. cld^c_k, \overline{pn}) = \overline{cld^c} \end{array}}{\overline{find\_pkg\_clds}(cld^c \ cld^c_2 .. cld^c_k, \overline{pn}) = cld^c \ \#\overline{cld^c}}$$

FPC\_CONS\_FALSE

$$\frac{\begin{array}{l} 1. cld^c = \mathbf{package} pn; \mathit{am\ class} dcl \mathbf{extends} cl\{\overline{fd} \overline{meth\_def^c}\} \\ 2. pn \notin \overline{pn} \\ 3. \overline{find\_pkg\_clds}(cld^c_2 .. cld^c_k, \overline{pn}) = \overline{cld^c} \end{array}}{\overline{find\_pkg\_clds}(cld^c \ cld^c_2 .. cld^c_k, \overline{pn}) = \overline{cld^c}}$$

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

PCG\_MF

$$\frac{1. \overline{find\_pkg\_clds(\overline{cld^c_1}, \overline{pn_j^j}) = \overline{cld^c_2}}}{\overline{cld^c_1 \vdash repl \text{superpackage } mn\{\text{member } pn_j;^j \text{ imp}_k;^k \text{ export } fqnl;^l\} \rightsquigarrow repl \text{ module } mn\{\overline{cld^c_2} \text{ imp}_k^k \text{ fqnl}^l\}}$$

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

$$\begin{array}{c} \text{CLS\_BLOCK} \qquad \text{CLS\_VAR\_ASSIGN} \qquad \text{CLS\_FIELD\_READ} \qquad \text{CLS\_FIELD\_WRITE} \\ \frac{1. \overline{\vdash_{ctx} s_k^c \rightsquigarrow s_k^k}}{\vdash_{ctx} \{\overline{s_k^c}\} \rightsquigarrow \{\overline{s_k^k}\}} \quad \overline{\vdash_{ctx} var = x; \rightsquigarrow var = x;} \quad \overline{\vdash_{ctx} var = x.f; \rightsquigarrow var = x.f;} \quad \overline{\vdash_{ctx} x.f = y; \rightsquigarrow x.f = y;} \\ \text{CLS\_IF} \qquad \qquad \qquad \text{CLS\_MCALL} \\ \frac{1. \overline{\vdash_{ctx} s_1^c \rightsquigarrow s_1} \quad 2. \overline{\vdash_{ctx} s_2^c \rightsquigarrow s_2}}{\vdash_{ctx} \text{if } (x == y) s_1^c \text{ else } s_2^c \rightsquigarrow \text{if } (x == y) s_1 \text{ else } s_2} \quad \overline{\vdash_{ctx} var = x.meth(\overline{y_k^k}); \rightsquigarrow var = x.meth(\overline{y_k^k});} \\ \text{CLS\_NEW} \\ \overline{\vdash_{ctx} var = \text{new } cl(); \rightsquigarrow var = \text{new}_{ctx} cl();} \end{array}$$

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

CL\_METH\_DEF

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

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

CL\_MODULE

$$\frac{1. \overline{\vdash_{mi} cld_j^c \rightsquigarrow cld_j^j}}{\overline{\vdash_{mi} repl \text{ module } mn\{\overline{cld_j^c} \text{ imp}_k^k \text{ fqnl}\} \rightsquigarrow repl \text{ module } mn\{\overline{cld_j^j} \text{ imp}_k^k \text{ fqnl}\}}}$$

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

CL\_CLD

$$\begin{array}{c} 1. \overline{cld^c = \text{package } pn; \text{ am class } dcl \text{ extends } cl\{\overline{fd} \text{ meth\_def}_k^c\}} \\ 2. \overline{\vdash_{mi, pn} \text{ meth\_def}_k^c \rightsquigarrow \text{meth\_def}_k^k} \\ 3. \overline{cld = \text{package } pn; \text{ am class } dcl \text{ extends } cl\{\overline{fd} \text{ meth\_def}_k^k\}} \\ \hline \vdash_{mi} cld^c \rightsquigarrow cld \end{array}$$

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

$$\begin{array}{c} \text{TR\_S\_BLOCK} \qquad \text{TR\_S\_VAR\_ASSIGN} \qquad \text{TR\_S\_FIELD\_READ} \\ \frac{1. \overline{\theta \vdash s_k \rightsquigarrow s'_k}}{\theta \vdash \{\overline{s_k}\} \rightsquigarrow \{\overline{s'_k}\}} \quad \frac{1. \theta(var) = var' \quad 2. \theta(x) = x'}{\theta \vdash var = x; \rightsquigarrow var' = x';} \quad \frac{1. \theta(var) = var' \quad 2. \theta(x) = x'}{\theta \vdash var = x.f; \rightsquigarrow var' = x'.f;} \\ \text{TR\_S\_FIELD\_WRITE} \qquad \qquad \qquad \text{TR\_S\_IF} \\ \frac{1. \theta(x) = x' \quad 2. \theta(y) = y'}{\theta \vdash x.f = y; \rightsquigarrow x'.f = y';} \quad \frac{1. \theta(x) = x' \quad 2. \theta(y) = y' \quad 3. \theta \vdash s_1 \rightsquigarrow s'_1 \quad 4. \theta \vdash s_2 \rightsquigarrow s'_2}{\theta \vdash \text{if } (x == y) s_1 \text{ else } s_2 \rightsquigarrow \text{if } (x' == y') s'_1 \text{ else } s'_2} \\ \text{TR\_S\_MCALL} \\ \text{TR\_S\_NEW} \\ \frac{1. \theta(var) = var'}{\theta \vdash var = \text{new}_{ctx} cl(); \rightsquigarrow var' = \text{new}_{ctx} cl();} \quad \frac{1. \theta(var) = var' \quad 2. \theta(x) = x' \quad 3. \overline{\theta(y_k) = y'_k}}{\theta \vdash var = x.meth(\overline{y_k^k}); \rightsquigarrow var' = x'.meth(\overline{y'_k^k});} \end{array}$$

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

<p style="text-align: center;">R_BLOCK</p> $\frac{}{(P, L, H, \{\overline{s_k^k}\} \overline{s_l^l}) \longrightarrow (P, L, H, \overline{s_k^k} \overline{s_l^l})}$ <p style="text-align: center;">R_FIELD_READ_NPE</p> $\frac{1. L(x) = \mathbf{null}}{(P, L, H, var = x.f; \overline{s_l^l}) \longrightarrow (P, L, H, \mathbf{NPE})}$ <p style="text-align: center;">R_FIELD_WRITE_NPE</p> $\frac{1. L(x) = \mathbf{null}}{(P, L, H, x.f = y; \overline{s_l^l}) \longrightarrow (P, L, H, \mathbf{NPE})}$ <p style="text-align: center;">R_IF_TRUE</p> $\frac{1. L(x) = v \quad 2. L(y) = w \quad 3. v = w}{(P, L, H, \mathbf{if} (x == y) s_1 \mathbf{else} s_2 \overline{s_l^l}) \longrightarrow (P, L, H, s_1 \overline{s_l^l})}$	<p style="text-align: center;">R_VAR_ASSIGN</p> $\frac{1. L(x) = v}{(P, L, H, var = x; \overline{s_l^l}) \longrightarrow (P, L[var \mapsto v], H, \overline{s_l^l})}$ <p style="text-align: center;">R_FIELD_READ</p> $\frac{1. L(x) = oid \quad 2. H(oid, f) = v}{(P, L, H, var = x.f; \overline{s_l^l}) \longrightarrow (P, L[var \mapsto v], H, \overline{s_l^l})}$ <p style="text-align: center;">R_FIELD_WRITE</p> $\frac{1. L(x) = oid \quad 2. L(y) = v}{(P, L, H, x.f = y; \overline{s_l^l}) \longrightarrow (P, L, H[(oid, f) \mapsto v], \overline{s_l^l})}$ <p style="text-align: center;">R_IF_FALSE</p> $\frac{1. L(x) = v \quad 2. L(y) = w \quad 3. v \neq w}{(P, L, H, \mathbf{if} (x == y) s_1 \mathbf{else} s_2 \overline{s_l^l}) \longrightarrow (P, L, H, s_2 \overline{s_l^l})}$ <p style="text-align: center;">R_NEW</p> $\frac{1. find\_type(P, ctx, cl) = \tau \quad 2. \mathbf{fields} (P, \tau) = \overline{f_k^k} \quad 3. oid \notin \mathbf{dom} (H) \quad 4. H' = H[oid \mapsto (\tau, f_k \mapsto \mathbf{null}^k)]}{(P, L, H, var = \mathbf{new}_{ctx} cl(); \overline{s_l^l}) \longrightarrow (P, L[var \mapsto oid], H', \overline{s_l^l})}$ <p style="text-align: center;">R_MCALL_NPE</p> $\frac{1. L(x) = \mathbf{null}}{(P, L, H, var = x.meth(\overline{y_k^k}); \overline{s_l^l}) \longrightarrow (P, L, H, \mathbf{NPE})}$ <p style="text-align: center;">R_MCALL</p> $\frac{1. L(x) = oid \quad 2. H(oid) = \tau \quad 3. find\_meth\_def(P, \tau, meth) = (ctx, cl, meth(\overline{cl_k} \overline{var_k^k}) \{s_j^j \mathbf{return} y; \}) \quad 4. \overline{var_k^k} \perp \mathbf{dom} (L) \quad 5. \mathbf{distinct} (\overline{var_k^k}) \quad 6. x' \notin \mathbf{dom} (L) \quad 7. x' \notin \overline{var_k^k} \quad 8. \overline{L(y_k)} = v_k \quad 9. L' = L[\overline{var_k^k} \mapsto v_k^k][x' \mapsto oid] \quad 10. \theta = [\overline{var_k^k} \mapsto \overline{var_k^k}][\mathbf{this} \mapsto x'] \quad 11. \theta \vdash s_j^j \rightsquigarrow s_j^j \quad 12. \theta(y) = y'}{(P, L, H, var = x.meth(\overline{y_k^k}); \overline{s_l^l}) \longrightarrow (P, L', H, \overline{s_j^j} var = y'; \overline{s_l^l})}$
---	--

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

<p style="text-align: center;">R_NO_ACTION</p> $\overline{config \longrightarrow config}$	<p style="text-align: center;">R_ACTION_LIST</p> $\frac{1. (P, L, H, \overline{s_l^l}) \xrightarrow{ia} (P', L, H, \overline{s_l^l}) \quad 2. (P', L, H, \overline{s_l^l}) \xrightarrow{ia_2 \dots ia_k} (P'', L, H, \overline{s_l^l})}{(P, L, H, \overline{s_l^l}) \xrightarrow{ia \ ia_2 \dots ia_k} (P'', L, H, \overline{s_l^l})}$ <p style="text-align: center;">R_INSTALL</p> $\frac{1. RC(rn) = R \quad 2. R\_body(R) = (\overline{md_k^c}, \phi) \quad 3. md\_name(md^c) = m \quad 4. \overline{md_k^c} = \overline{mn_k^k} \quad 5. m \notin \overline{mn_k^k} \quad 6. R\_update(R, md^c, \overline{md_k^c}, \phi) = R' \quad 7. RC' = RC[rn \mapsto R']}{((RC, MH), L, H, \overline{s_l^l}) \xrightarrow{rn.\mathbf{install}(md^c)} ((RC', MH), L, H, \overline{s_l^l})}$
---	--

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.  $RC' = RC[rn \mapsto R']$

$$\frac{((RC, MH), L, H, \overline{s_i^l})}{R\_EXISTING\_INSTANCE} \xrightarrow{rn.\text{uninstall}(m)} ((RC', MH), L, H, \overline{s_i^l})$$

1.  $imp\_name(imp) = m$
2.  $find\_md(RC, rn_1, m) = (rn_2, md^c)$
3.  $RC(rn_2) = R_2$
4.  $R\_body(R_2) = (\overline{md^c_2}, \phi_2)$
5.  $mi' \notin \mathbf{dom}(MH)$
6.  $imp\_dep\_of(md^c, mi', imp) = imp\_dep$
7.  $\phi_2(md^c, imp\_dep) = mi$

$$\frac{((RC, MH), L, H, \overline{s_i^l})}{R\_NEW\_INSTANCE} \xrightarrow{mi=rn_1.get\_instance(imp)} ((RC, MH), L, H, \overline{s_i^l})$$

1.  $imp\_name(imp) = m$
2.  $find\_md(RC, rn_1, m) = (rn_2, md^c)$
3.  $RC(rn_2) = R_2$
4.  $R\_body(R_2) = (\overline{md^c_2}, \phi_2)$
5.  $mi' \notin \mathbf{dom}(MH)$
6.  $imp\_dep\_of(md^c, mi', imp) = imp\_dep'$
7.  $\phi_2(md^c, imp\_dep') = \mathbf{null}$
8.  $md^c = repl\ \mathbf{module}\ m\ \{ \overline{cld^c}\ \overline{imp_k^k}\ \overline{fq_n} \}$
9.  $((RC, MH), L, H, \overline{s_i^l}) \xrightarrow{mi=rn_2.get\_instance(imp_k)^k} ((RC', MH'), L, H, \overline{s_i^l})$
10.  $mi \notin \mathbf{dom}(MH')$
11.  $imp\_dep\_of(md^c, mi, imp) = imp\_dep$
12.  $\vdash_{mi} md^c \rightsquigarrow md$
13.  $imp\_br(imp_k) = br_k$
14.  $MH'' = MH'[mi \mapsto (md, \overline{mi_k}\ br_k^k)]$
15.  $RC'(rn_2) = R'_2$
16.  $R\_body(R'_2) = (\overline{md^c_3}, \phi_3)$
17.  $R\_update(R'_2, \overline{md^c_3}, \phi_3[md^c \mapsto imp\_dep \mapsto m]) = R''_2$
18.  $RC'' = RC'[rn_2 \mapsto R''_2]$
19.  $(RC'', MH'') \vdash_{mi} md$

$$\frac{((RC, MH), L, H, \overline{s_i^l})}{R\_NEW\_INSTANCE} \xrightarrow{mi=rn_1.get\_instance(imp)} ((RC'', MH''), L, H, \overline{s_i^l})$$

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

ADMIN\_INSTALL

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

ADMIN\_UNINSTALL

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

ADMIN\_NEW\_INSTANCE

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

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

WRC\_INSTALL

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

WRC\_UNINSTALL

$$\frac{\begin{array}{l} 1. \vdash (RC, MH) \quad 2. mi \notin \mathbf{dom}(MH) \\ 3. RC(rn) = R \quad 4. R\_body(R) = (\overline{md^c_1}, \phi) \\ 5. find\_md\_in\_mds(\overline{md^c_1}, m) = md^c \\ 6. mds\_rm(\overline{md^c_1}, md^c) = \overline{md^c_2} \\ 7. R\_update(R, \overline{md^c_2}, \phi \setminus md^c) = R' \end{array}}{(RC, MH), mi, (RC[rn \mapsto R'], MH) \in wf\_P\_change}$$

1.  $\vdash (RC, MH)$
  2.  $mi \notin \mathbf{dom}(MH)$
  3.  $RC(rn) = R$
  4.  $R\_body(R) = (\overline{md^c}, \phi)$
  5.  $\overline{mi_k}^k \subseteq \mathbf{dom}(MH)$
  6.  $md\_name(\overline{md^c}) = m$
  7.  $R\_update(R, \overline{md^c}, \phi[md^c \mapsto imp\_dep \mapsto mi]) = R'$
  8.  $RC' = RC[rn \mapsto R']$
  9.  $MH' = MH [mi \mapsto (md, \overline{mi_k}^k)]$
  10.  $(RC', MH') \vdash_{mi} md$
- 
- $((RC, MH), mi, (RC', MH')) \in wf\_P.change$