\[ mf \quad ::= \quad \text{superpackage } mn \{ \text{member } pn_j; \text{import } mn_k; \text{export } fqn_l; \} \quad \text{module file def.} \\
\]

\[ mn \quad ::= \quad \text{core } m \quad | \quad m \quad \text{module name def.} \]

\[ fqn \quad ::= \quad pn.dcl \quad \text{fully-qualified name def.} \]

\[ SRC \quad ::= \quad cld^c \quad \text{source files (}cld^c\text{) def.} \]

\[ cld^c \quad ::= \quad cld_1^c .. cld_k^c \quad | \quad cld_1^c # cld_k^c \]

\[ cld^c \quad ::= \quad pd \ am \ class \ dcl \ extends \ cl \{ \text{fd meth def}_c^c \} \quad \text{class, compile-time code def.} \]

\[ pd \quad ::= \quad \text{package } pn; \quad \text{package declaration (}pn\text{) def.} \]

\[ am \quad ::= \quad \text{public} \quad \text{access modifier def.} \]

\[ C, \ cl \quad ::= \quad \text{Object} \quad | \quad fqn \quad \text{class name def.} \]

\[ \overline{fd} \quad ::= \quad [ ] \quad | \quad fd_1 .. fd_k \quad \text{field declarations (}fd\text{ list) def.} \]

\[ fd \quad ::= \quad cl \ f; \quad \text{field declaration def.} \]

\[ \overline{\text{meth def}_c^c} \quad ::= \quad \text{meth def}_1^c .. \text{meth def}_k^c \quad \text{method def.'s, compile-time code def.} \]
\[
\begin{align*}
\text{meth}_\text{def} & ::= \text{meth}_\text{sig}\{\text{meth}_\text{body}\} \\
\text{meth}_\text{sig} & ::= \text{cl meth(}\overline{\text{vd}}\text{)} \\
\overline{\text{vd}} & ::= \text{vd}_1..\text{vd}_k \\
\text{vd} & ::= \text{cl var} \\
\text{meth}_\text{body} & ::= s_1^x..s_k^x \text{return } y; \\
s^x & ::= \{s_1^x..s_k^x\} \\
& | \text{var } = x; \\
& | \text{var } = x.f; \\
& | x.f = y; \\
& | \text{if } (x==y)s_1^x \text{ else } s_2^x \\
& | \text{var } = \text{new } cl(); \\
& | \text{var } = x.meth(\overline{y}); \\
T\text{Var}, x, y & ::= \text{var} \\
& | \text{this} \\
\overline{x}, \overline{y} & ::= x_1..x_k \\
P & ::= (R.C, MH) \\
R.C & ::= [] \text{ empty repository context} \\
& | R.C[\text{rn } \mapsto R] \text{ rn maps to } R \text{ in } R.C \\
rn & ::= \text{bootstrap}_r \text{ bootstrap} \\
& | \text{r} \text{ standard} \\
R & ::= \text{bootstrap repository } \{\overline{md}; \phi\} \text{ bootstrap} \\
& | \text{repository } r \text{ child of } \{\text{rn}\{\overline{md}; \phi\}\} \text{ standard} \\
\overline{md} & ::= md_1^x..md_k^x \text{ def.} \\
& | md^x#\overline{md} \text{ cons} \\
md & ::= \text{module def.'s, compile-time code } (md^x \text{ list}) \\
& | \text{module definition}
\end{align*}
\]
\[ \text{module } mn \{ \text{cld } m \} \quad \text{def.} \]

\[ m \quad ::= \quad m_1 .. m_k \quad \text{M def.} \]

\[ \text{fqn} \quad ::= \quad fqn_1 .. fqn_k \quad \text{M def.} \]

\[ \phi \quad ::= \quad [] \quad \text{M empty repository's cache} \]
\[ \phi \{ m \} \quad \text{M map } m \text{ to } \phi \]
\[ \phi \setminus m \quad \text{M remove mapping for } m \]

\[ \text{MH} \quad ::= \quad [] \quad \text{M empty module hierarchy} \]
\[ [ mi \mapsto (md, m) ] \quad \text{M maps } mi \text{ to the given def. and imports} \]
\[ \text{MH}_1 .. \text{MH}_k \quad \text{M composes many} \]

\[ md \quad ::= \quad \text{module } mn \{ \text{cld } m \} \quad \text{def.} \]

\[ \text{cld} \quad ::= \quad [] \quad \text{M empty} \]
\[ \text{cld}_1 .. \text{cld}_k \quad \text{M def.} \]

\[ \text{cld} \quad ::= \quad \text{pd am class dcl extends cl} \{ \text{fd meth_def} \} \quad \text{def.} \]

\[ \text{meth_def} \quad ::= \quad [] \quad \text{M empty} \]
\[ \text{meth_def}_1 .. \text{meth_def}_k \quad \text{M def.} \]

\[ \text{meth_def} \quad ::= \quad \text{methSig} \{ \text{meth_body} \} \quad \text{M def.} \]

\[ \text{meth_body} \quad ::= \quad s_1 .. s_k \quad \text{return } y; \quad \text{M def.} \]

\[ s \quad ::= \quad \{ \pi^k \} \quad \text{M block} \]
\[ \text{var} = x; \quad \text{M variable assignment} \]
\[ \text{var} = x.f; \quad \text{M field read} \]
\[ x.f = y; \quad \text{M field write} \]
\[ \text{if} \ (x == y) s \text{ else } s' \quad \text{M conditional branch} \]
\[ \text{var} = \text{new } ctx \ cl(); \quad \text{M object creation} \]
\[ \text{var} = x.meth(y); \quad \text{M method call} \]

\[ ctx \quad ::= \quad mi.pn \quad \text{M def.} \]

\[ \overline{mi} \quad ::= \quad \text{module instance identifiers } (mi \text{ list}) \]
| $m\ddot{d}c_{opt}$ | ::= | module def., compile-time code option ($m\ddot{d}c$ option)
| | | $null$ | M | none
| | $m\ddot{d}c$ | M | some

| $\vec{f}$ | ::= | fields ($f$ list)
| | | $[]$ | M | empty
| | $f_1 .. f_k$ | M | def.
| | $\vec{f};\vec{f'}$ | M | append

| $\vec{f}_{opt}$ | ::= | fields option ($\vec{f}$ option)
| | | $null$ | M | none
| | $\vec{f}$ | M | some

| $\overline{meth}$ | ::= | method names ($meth$ list)
| | | $[]$ | M | empty
| | $meth_1 .. meth_k$ | M | def.
| | $meth;meth$ | M | append

| $meth_{\text{def}}_{opt}$ | ::= | method def. option ($meth_{\text{def}}$ option)
| | | $null$ | M | none
| | $meth_{\text{def}}$ | M | some

| ctx$meth_{\text{def}}_{opt}$ | ::= | method def. in context option ($(ctx \times meth_{\text{def}}$) option)
| | | $null$ | M | none
| | $(ctx,meth_{\text{def}})$ | M | some

| cld_{opt} | ::= | class def. option ($cld$ list)
| | | $null$ | M | none
| | $cld$ | M | some

| ctx$cld$ | ::= | 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.
| | $\overline{ctxcld}@[\overline{ctxcld}]$ | M | rev cons

| ctx$cld_{opt}$ | ::= | 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.
<table>
<thead>
<tr>
<th>$mi_{opt}$</th>
<th>::= module instance option ($mi$ option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>M none</td>
</tr>
<tr>
<td>$mi$</td>
<td>M some</td>
</tr>
<tr>
<td>$\phi(mdc)$</td>
<td>M module instance lookup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R_{opt}$</th>
<th>::= repository option ($R$ option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>M none</td>
</tr>
<tr>
<td>$R$</td>
<td>M some</td>
</tr>
<tr>
<td>$RC(rn)$</td>
<td>M repository lookup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$mhv$</th>
<th>::= module hierarchy value ($md \times m$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(md, m)$</td>
<td>M def.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$mhv_{opt}$</th>
<th>::= module hierarchy value option ($mhv$ option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>M none</td>
</tr>
<tr>
<td>$mhv$</td>
<td>M some</td>
</tr>
<tr>
<td>$MH(mi)$</td>
<td>M lookup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$rnmd_{opt}$</th>
<th>::= module def., compile-time code lookup value ($rnmd$ option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>M none</td>
</tr>
<tr>
<td>$(rn, md)$</td>
<td>M some</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$Type, \tau$</th>
<th>::= type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>supertype of all types</td>
</tr>
<tr>
<td>$ctx.dcl$</td>
<td>class identifier</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\tau_{opt}$</th>
<th>::= result of type lookup ($\tau$ option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>M none</td>
</tr>
<tr>
<td>$\tau$</td>
<td>M some</td>
</tr>
<tr>
<td>$\Gamma(x)$</td>
<td>M static type lookup</td>
</tr>
<tr>
<td>$H(oid)$</td>
<td>M dynamic type lookup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\tau_{\perp}$</th>
<th>::= result of type lookup that can abort</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{opt}$</td>
<td>result of type lookup</td>
</tr>
<tr>
<td>$\perp$</td>
<td>failed to find a type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\tau$</th>
<th>::= types ($\tau$ list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1 .. \tau_k$</td>
<td>M def.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\pi$</th>
<th>::= method type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau \to \tau$</td>
<td>def.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\Gamma$</th>
<th>::= type environment ($x \to \tau$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x_1 \mapsto \tau_1 .. x_k \mapsto \tau_k]$</td>
<td>M type mappings</td>
</tr>
<tr>
<td>$\Gamma[x \mapsto \tau]$</td>
<td>M $\Gamma$ with $x \mapsto \tau$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>::= variable mapping ($x \mapsto x$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x_1 \mapsto y_1 .. x_k \mapsto y_k]$</td>
<td>M variable mappings</td>
</tr>
<tr>
<td>$\theta[x \mapsto y]$</td>
<td>M $\theta$ with $x \mapsto y$</td>
</tr>
</tbody>
</table>

| $Val, v, w$ | ::= value |
null
oid
null value
object identifier

\( v \)  
result of value lookup (v option)

\( L(x) \)  
lookup value of local variable

\( H(oid, f) \)  
lookup value of field

\( [\] \)  
variable state (\( x \to v \))

\( [x \to v] \)  
empty variable state

\( L[x_1 \to v_1 \ldots x_k \to v_k] \)  
\( L \) with \( x \to v \)

\( H(oid, f) \to v \)  
heap (\( oid \to (\tau \times (f \to v)) \))

\( [\] \)  
empty heap

\( H[oid \to (\tau, f_1 \to v_1 \ldots f_k \to v_k)] \)  
\( H \) with new \( oid \) of type \( \tau \)

\( H[(oid, f) \to v] \)  
\( H \) with \( (oid, f) \to v \)

config

\( (P, L, H, s) \) normal configuration

\( (P, L, H, Exception) \) exception occurred

Exception

\( NPE \) null-pointer exception

a

\( rn. install(md^c); \) install

\( rn. uninstall(m); \) uninstall

\( rn. initialise(m); \) initialise

ia

\( rn. install(md^c) \) install

\( rn. uninstall(m) \) uninstall

\( m = rn. get_instance(m) \) initialise

\( \overline{ia} \)

\( ia_1 \ldots ia_k \) internal actions

\( nn \)

\( 0 \) zero

\( 1 \) one

\( nn + nn' \) plus

\( nn - nn' \) minus

\( \text{size}(\text{dom} RC) \) size of domain of \( RC \)

\( \text{R\_name}(R) = rn \) extract repository’s name

\( \text{R\_name\_bootstrap} \) 

\( \text{R\_name\_standard} \) 

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

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

\( \text{R\_body}(R) = (md^c, \phi) \) extract repository’s contents
**R_body_bootstrap**

\[ R_{\text{body bootstrap repository}} \{\text{md}^C; \phi\} = (\text{md}^C, \phi) \]

\[ R_{\text{update}}(R, \text{md}^C, \phi) = R' \] – update a repository with given contents

\[ R_{\text{update bootstrap repository}} \{\text{md}^C_1; \phi_1\}, \text{md}^C_2, \phi_2 = \text{bootstrap repository} \{\text{md}^C_2; \phi_2\} \]

\[ R_{\text{update standard repository}} \text{r child of } \text{rn} \{\text{md}^C; \phi\} = (\text{md}^C, \phi) \]

**R_body_standard**

\[ \text{mds rm}(\text{md}^C_1, \text{md}^C) = \overline{\text{md}^C_2} \] – remove a module def. from a list

\[
\begin{align*}
\text{MDS_RM_EMPT} & : \quad \text{mds rm}(\text{md}^C) = \overline{\text{md}^C} \\
\text{MDS_RM_CONS_TRUE} & : \quad \begin{cases} 
1. \text{mds rm}(\overline{\text{md}^C_1}, \text{md}^C) = \overline{\text{md}^C} \\
\text{mds rm}(\text{md}^C_1 \# \overline{\text{md}^C_1}, \text{md}^C) = \overline{\text{md}^C_2}
\end{cases} \\
\text{MDS_RM_CONS_FALSE} & : \quad \begin{cases} 
1. \text{md}^C_1 \neq \text{md}^C \\
2. \text{mds rm}(\overline{\text{md}^C_1}, \text{md}^C) = \overline{\text{md}^C_2} \\
\text{mds rm}(\text{md}^C_1 \# \overline{\text{md}^C_1}, \text{md}^C) = \overline{\text{md}^C_1 \# \text{md}^C_2}
\end{cases}
\end{align*}
\]

\[ \text{md.name}(\text{md}^C) = \text{mn} \] – get name of module def.

\[ \text{md.name}(\overline{\text{md}^C}) = \text{mn} \]

\[ \text{full.name}(\text{cl}d) = \text{fq}n \] – extract the full name from a class

**full.name**

\[
\begin{align*}
\text{full.name.package } \text{pn}; \text{am class } \text{dcl } \text{extends } \text{cl}\{\text{fd meth def}\} & = \text{pn.dcl} \\
\text{package.name}(\text{cl}d) & = \text{pn} \quad \text{– extract the package name of a class}
\end{align*}
\]

**PACKAGE_NAME**

\[
\begin{align*}
\text{package.name.package } \text{pn}; \text{am class } \text{dcl } \text{extends } \text{cl}\{\text{fd meth def}\} & = \text{pn} \\
\text{class.name}(\text{cl}d) & = \text{dcl} \quad \text{– extract the class name from a class}
\end{align*}
\]

**CLASS_NAME**

\[
\begin{align*}
\text{class.name}(\text{dcl} \text{am class } \text{dcl } \text{extends } \text{cl}\{\text{fd meth def}\}) & = \text{dcl} \\
\text{superclass.name}(\text{cl}d) & = \text{cl} \quad \text{– extract the superclass name from a class}
\end{align*}
\]

**SUPERCLASS_NAME**

\[
\begin{align*}
\text{superclass.name}(\text{dcl} \text{am class } \text{dcl } \text{extends } \text{cl}\{\text{fd meth def}\}) & = \text{cl} \\
\text{class.fields}(\text{cl}d) & = \text{fd} \quad \text{– extract class fields from a class}
\end{align*}
\]

**CLASS_FIELDS**

\[
\begin{align*}
\text{class.fields}(\text{dcl} \text{am class } \text{dcl } \text{extends } \text{cl}\{\text{fd meth def}\}) & = \text{fd}
\end{align*}
\]
**CLASS_METHODS**

\[ \text{class\_methods}(\text{cld}) = \text{meth\_def} \] extract class methods from a class

\[ \text{class\_methods}(\text{pd\ am\ class\ del\ extends\ cl}\{\text{\_def\ meth\_def}\}) = \text{meth\_def} \]

\[ \text{method\_name}(\text{meth\_def}) = \text{meth} \] extract the method name from a method definition

\[ \text{method\_name}(\text{cl\ meth\(\text{\_d}\)\{\text{meth\_body}\}}) = \text{meth} \]

\[ \text{distinct\_fqsns}(\text{cld}) \] fully-qualified names are distinct

\[ \text{DF\_DEF} \]

\[ \begin{align*}
1. & \quad \text{\_full\_name}(\text{cld}) = \text{fqn}_k \\
2. & \quad \text{\_\_distinct}(\text{fqn}_k) \\
\end{align*} \]

\[ \text{distinct\_fqsns}(cld, k) \]

\[ \text{find\_md\_in\_mds}(\text{md}^k, \text{mn}) = \text{md}^\text{opt} \] module definition lookup in a list

\[ \text{FMIM\_EMPTY} \quad \text{FMIM\_CONS\_TRUE} \]

\[ \begin{align*}
\text{find\_md\_in\_mds}(& \text{cld}, \text{mn}) = \text{null} \\
\text{find\_md\_in\_mds}(& \text{md}^k \text{md}^2 \ldots \text{md}^\text{opt}, \text{mn}) = \text{md}^\text{cld} \\
\text{FMIM\_CONS\_FALSE} \]

\[ \begin{align*}
1. & \quad \text{\_md} = \text{module\ mn}\{\text{\_cld}\ \text{fqn}^\text{k}\} \\
2. & \quad \text{\_fqn} = \text{null'} \\
3. & \quad \text{\_mn} \neq \text{mn'} \\
\text{find\_md\_in\_mds}(& \text{md}^k \text{md}^2 \ldots \text{md}^\text{opt}, \text{mn}) = \text{md}^\text{opt} \\
\text{FMIM\_BOOTSTRAP} \]

\[ \text{find\_md\_rec}(\text{RC}, \text{rn}_1, \text{mn}, \text{nn}) = \text{rnm} \text{md}^\text{opt} \] module def. lookup (recursive part)

\[ \text{FMIM\_NULL} \quad \text{FMIM\_BOOTSTRAP\_NULL} \quad \text{FMIM\_STANDARD\_FAIL} \]

\[ \begin{align*}
1. & \quad \text{\_RC}(\text{rn}) = \text{null} \\
\text{find\_md\_rec}(\text{RC}, \text{rn}, \text{mn}, \text{nn}) = \text{null} \\
\text{FMIM\_STANDARD\_SELF} \]

\[ \begin{align*}
1. & \quad \text{\_RC}(\text{rn}) = \text{bootstrap\ repository\ \{\text{md}^\text{cld}\}; \phi} \\
2. & \quad \text{\_find\_md\_in\_mds}(\text{md}^\text{cld}, \text{mn}) = \text{null} \\
\text{find\_md\_rec}(\text{RC}, \text{rn}, \text{mn}, \text{nn}) = \text{null} \\
\text{FMIM\_STANDARD\_NULL} \]

\[ \begin{align*}
1. & \quad \text{\_RC}(\text{rn}) = \text{repository\ r\ child\ of\ \text{rn}\{\text{md}^\text{cld}\}; \phi} \\
2. & \quad \text{\_size}(\text{dom\ RC}) > \text{nn} \\
3. & \quad \text{\_find\_md\_rec}(\text{RC}, \text{rn}_2, \text{mn}, \text{nn} + 1) = (\text{rn}_3, \text{md}^\text{cld}) \\
\text{find\_md\_rec}(\text{RC}, \text{rn}_1, \text{mn}, \text{nn}) = (\text{rn}_3, \text{md}^\text{cld}) \\
\end{align*} \]

\[ \text{FMR\_NULL} \]

\[ \begin{align*}
1. & \quad \text{RC}(\text{rn}) = \text{null} \\
\text{find\_md\_rec}(\text{RC}, \text{rn}, \text{mn}, \text{nn}) = \text{null} \\
\text{FMR\_BOOTSTRAP\_NULL} \]

\[ \begin{align*}
1. & \quad \text{RC}(\text{rn}) = \text{bootstrap\ repository\ \{\text{md}^\text{cld}\}; \phi} \\
2. & \quad \text{find\_md\_in\_mds}(\text{md}^\text{cld}, \text{mn}) = \text{null} \\
\text{find\_md\_rec}(\text{RC}, \text{rn}, \text{mn}, \text{nn}) = \text{null} \\
\text{FMR\_STANDARD\_FAIL} \]

\[ \begin{align*}
1. & \quad \text{RC}(\text{rn}) = \text{repository\ r\ child\ of\ \text{rn}\{\text{md}^\text{cld}\}; \phi} \\
2. & \quad \text{size}(\text{dom\ RC}) > \text{nn} \\
3. & \quad \text{find\_md\_rec}(\text{RC}, \text{rn}_2, \text{mn}, \text{nn} + 1) = \text{null} \\
4. & \quad \text{find\_md\_in\_mds}(\text{md}^\text{cld}, \text{mn}) = \text{null} \\
\text{find\_md\_rec}(\text{RC}, \text{rn}_1, \text{mn}, \text{nn}) = \text{null} \\
\text{FMR\_STANDARD\_NULL} \]

\[ \begin{align*}
1. & \quad \text{RC}(\text{rn}) = \text{repository\ r\ child\ of\ \text{rn}\{\text{md}^\text{cld}\}; \phi} \\
2. & \quad \text{size}(\text{dom\ RC}) > \text{nn} \\
3. & \quad \text{find\_md\_rec}(\text{RC}, \text{rn}_2, \text{mn}, \text{nn} + 1) = \text{null} \\
4. & \quad \text{find\_md\_in\_mds}(\text{md}^\text{cld}, \text{mn}) = \text{null} \\
\text{find\_md\_rec}(\text{RC}, \text{rn}_1, \text{mn}, \text{nn}) = \text{null} \]
\[ \text{find\_md}(RC, \text{rn}, \text{mn}) = \text{rnm\_md\_opt} \] – module def. lookup

\[
\text{FM\_DEF} \\
1. \text{find\_md\_rec}(RC, \text{rn}, \text{mn}, 0) = \text{rnm\_md\_opt} \\
\text{find\_md}(RC, \text{rn}, \text{mn}) = \text{rnm\_md\_opt}
\]

\[ \text{find\_cld\_in\_module(cld, fqn)} = \text{cld\_opt} \] – class lookup in an import

\[
\text{FCIM\_EMPTY} \\
\text{FCIM\_NULL} \\
\text{find\_cld\_in\_module([], fqn)} = \text{null} \\
\text{find\_cld\_in\_module(cld\_2 \ldots cld\_k, fqn)} = \text{null} \\
\text{FCIM\_CONS\_TRUE} \\
1. \text{distinct\_fqs}(cld\_2 \ldots cld\_k) \\
2. cld = \text{package pn; public class dcl extends c1, d2} \\
\text{find\_cld\_in\_module(cld\_2 \ldots cld\_k, pn.dcl)} = \text{cld}\_opt
\]

\[ \text{find\_cld\_in\_core(P, fqn)} = \text{ctx\_cld\_opt} \] – class lookup in the core library module

\[
\text{FCIC\_NO\_REP\_EX} \\
\text{FCIC\_NOT\_BOOTSTRAP\_EX} \\
1. \text{RC(bootstrap, r)} = \text{null} \\
\text{find\_cld\_in\_core((RC, MH), fqn)} = \text{null} \\
\text{find\_cld\_in\_core((RC, MH), fqn)} = \text{null} \\
\text{FCIC\_NO\_CORE\_EX} \\
1. \text{RC(bootstrap, r)} = \text{repository rl child of rn\{md\_r; \phi\}} \\
\text{find\_cld\_in\_core((RC, MH), fqn)} = \text{null} \\
\text{FCIC\_NO\_CORE\_MI\_EX} \\
1. \text{RC(bootstrap, r)} = \text{bootstrap repository \{md\_r; \phi\}} \\
2. \text{find\_md\_in\_mds(md\_r, core, rn)} = \text{null} \\
\text{find\_cld\_in\_core((RC, MH), fqn)} = \text{null} \\
\text{FCIC\_NO\_MDM\_EX} \\
1. \text{RC(bootstrap, r)} = \text{bootstrap repository \{md\_r; \phi\}} \\
2. \text{find\_md\_in\_mds(md\_r, core, rn)} = \text{md}\_r \\
3. (md\_r) = \text{null} \\
\text{find\_cld\_in\_core((RC, MH), fqn)} = \text{null} \\
\text{FCIC\_FALSE} \\
1. \text{RC(bootstrap, r)} = \text{bootstrap repository \{md\_r; \phi\}} \\
2. \text{find\_md\_in\_mds(md\_r, core, rn)} = \text{md}\_r \\
3. (md\_r) = \text{null} \\
4. MH(mn) = \text{null} \\
\text{find\_cld\_in\_module(cld, fqn)} = \text{null} \\
\text{find\_cld\_in\_core((RC, MH), fqn)} = \text{null} \]
FCIC_TRUE

1. $RC(bootstrap_r) = \texttt{bootstrap repository}\{md; \phi\}$
2. $\text{find}_\text{md}_\text{in}_\text{mds}(md, core, m) = md$
3. $\phi(md) = mi$
4. $MH(mi) = (\text{module}\ mn \{\text{cld} \ fqn\}, mi)$
5. $\text{find}_\text{cld}_\text{in}_\text{module}(\text{cld} \ fqn) = \text{cld}$
6. $\text{package}_\text{name}(\text{cld}) = pn$

$\text{find}_\text{cld}_\text{in}_\text{core}(RC, MH, fqn) = (mi, pn, \text{cld})$

$(MH, m, mi, nn) \in \text{reachable}$ — there are $nn$ module instances reachable from $mi$ in $MH$

REACHABLE_EMPTY

1. $MH(mi) = (md, mi)$
2. $(MH, m, mi, nn) \in \text{reachable}$
3. $(MH, m, mi, nn) \in \text{reachable}$

$(MH, m, mi, nn) \in \text{reachable}$

REACHABLE_CONS

1. $\forall mi, mi \subseteq \text{dom}(MH) \rightarrow (\exists nn. (MH, mi, nn) \in \text{reachable})$
2. $\forall mi \in \text{dom}(MH). \exists md, mi. MH(mi) = (md, mi) \land mi \subseteq \text{dom}(MH)$

acyclic_mH

$\text{find}_\text{cld}_\text{in}_\text{imports}(MH, m, m, fqn) = \text{ctxcld}_\text{opt}$ — class lookup in imports

FCI_NULL

$\text{find}_\text{cld}_\text{in}_\text{imports}(MH, m, m, fqn) = \text{null}$

1. $\neg (\text{acyclic_mH} \land mi \in \text{dom}(MH) \land mi \subseteq \text{dom}(MH))$

FCI_SKIP

1. $\text{acyclic_mH} \land \text{dom}(MH) \land mi \subseteq \text{dom}(MH)$
2. $md = \text{module}\ mn \{\text{cld} \ fqn\} \land fqn \notin fqn$
3. $\text{find}_\text{cld}_\text{in}_\text{imports}(MH, mi, m, fqn) = \text{ctxcld}_\text{opt}$

FCI_REC

1. $\text{acyclic_mH} \land \text{dom}(MH) \land mi \subseteq \text{dom}(MH)$
2. $md = \text{module}\ mn \{\text{cld} \ fqn\} \land fqn \in fqn$
3. $\text{find}_\text{cld}_\text{in}_\text{imports}(MH, mi, m, fqn) = \text{ctxcld}$

FCI_SELF

1. $\text{acyclic_mH} \land \text{dom}(MH) \land mi \subseteq \text{dom}(MH)$
2. $md = \text{module}\ mn \{\text{cld} \ fqn\} \land fqn \in fqn$
3. $\text{find}_\text{cld}_\text{in}_\text{imports}(MH, mi, m, fqn) = \text{null}$
4. $\text{find}_\text{cld}_\text{in}_\text{module}(\text{cld} \ fqn) = \text{cld} \land \text{package}_\text{name}(\text{cld}) = pn$

FCI_NEXT

1. $\text{acyclic_mH} \land \text{dom}(MH) \land mi \subseteq \text{dom}(MH)$
2. $md = \text{module}\ mn \{\text{cld} \ fqn\} \land fqn \in fqn$
3. $\text{find}_\text{cld}_\text{in}_\text{imports}(MH, mi, m, fqn) = \text{null}$
4. $\text{find}_\text{cld}_\text{in}_\text{module}(\text{cld} \ fqn) = \text{null}$
5. $\text{find}_\text{cld}_\text{in}_\text{imports}(MH, mi, m, fqn) = \text{ctxcld}_\text{opt}$
find_cld_in_set f(cld, pn, fqn) = cld_opt — class lookup in the same module

FCIS_EMPTY

find_cld_in_set f(cld, pn, fqn) = FCIS_NULL

1. distinct_fqns(cld cld2 .. cldk)

1. distinct_fqns(cld cld2 .. cldk)

2. cld = package pn'; am class dcl extends cl{fd meth_def}

2. cld = package pn''; am class dcl' extends cl{fd meth_def}

3. pn = pn' ∨ am = public

3. (pn ≠ pn' ∨ am ≠ public) ∨ pn' ≠ pn'' ∨ dcl ≠ dcl'

find_cld_in_set f(cld cld2 .. cldk, pn, pn'.dcl) = cld_opt

find_cld_in_set f(cld cld2 .. cldk, pn, pn'.dcl) = cld_opt

find_cld(P, cxt, fqn) = cxtcld — class lookup

FC_CORE

find_cld_in_core(P, fqn) = cxtcld

1. find_cld_in_core(P, fqn) = cxtcld

FC_NULL

find_cld((RC, MH), mi, pn, fqn) = null

FC IMPORTS

find_cld_in Imports(MH, mi, fqn) = cxtcld

1. find_cld_in_core((RC, MH), fqn) = null

2. MH(mi) = null

FC SELF

find_cld((RC, MH), mi, pn, fqn) = null

find_cld_in Imports(MH, mi, fqn) = null

find_cld_in_set f(cld, pn, fqn) = null

find_cld((RC, MH), mi, pn, fqn) = (mi, pn', cld)

find_cld((RC, MH), fqn) = null

find_cld(P, cxt, cld) = τ_opt — type lookup

FT OBJ

find_type(P, cxt, Object) = Object

1. find_cld(P, cxt, fqn) = null

FT NULL

find_type(P, cxt, fqn) = null

1. find_cld(P, cxt, pn, dcl) = (ctz', cld)

FT DCL

find_type(P, cxt, pn, dcl) = ctz'.dcl

find_type(P, cxt, fqn) = null

find_type(P, cxt, cl, mn) ∈ path_length — get the length of the inheritance path for cl

PL FQN

(P, cxt, Object, 0) ∈ path_length

1. find_cld(P, cxt, fqn) = (ctz', cld)

2. superclass_name(cld) = cl

3. (P, ctz', cl, mn) ∈ path_length

(P, cxt, fqn, mn + 1) ∈ path_length
acyclic_clds_{mi}^P$ – the class inheritance hierarchy in $P$ is acyclic (starting at $mi$)

\[\text{ACM_DEF}\]

1. $\forall pn \text{ fqn.} \ (\exists ctx' \text{ cld.} find_cld(P, mi, pn, fqn) = (ctx', cld) \rightarrow \exists nn. (P, mi, pn, fqn, nn) \in \text{path_length})$

acymcldsP – the class inheritance hierarchy in $P$ is acyclic

\[\text{AC_DEF}\]

1. $\forall mi. \text{acyclic_clds}_{mi}^P$

\[\text{find_path_rec}(P, ctx, cl, ctxcld) = ctxcld_{opt}\] – class path lookup (recursive part)

\[\text{FPR_OBJ}\]

\[\text{find_path_rec}(P, ctx, Object, ctxcld) = ctxcld\]

\[\text{FPR_FQN}\]

\[\text{find_path_rec}(P, ctx, fqn, ctxcld) = ctxcld_{opt}\]

\[\text{FPR_NULL}\]

\[\text{find_path_rec}(P, ctx, fqn) = \text{null}\]

\[\text{find_path}(P, ctx, cl) = ctxcld_{opt}\] – class path lookup with a class name

\[\text{FP_DEF}\]

1. $\text{find_path_rec}(P, ctx, cl, [\]) = ctxcld_{opt}$

\[\text{find_path}(P, ctx, cl) = ctxcld_{opt}\]

\[\text{FPTY_OBJ}\]

\[\text{find_path}(P, Object) = []\]

\[\text{FPTY_DCL}\]

\[\text{find_path}(P, mi, pn, dcl) = ctxcld_{opt}\]

\[\text{fields_in_path}(ctxcld) = \overline{f}\] – fields lookup in a class path

\[\text{FIP_EMPTY}\]

1. class\.fields(cl) = $\overline{f_j \overline{f_i}}$

\[\text{files_in_path}([\]) = []\]

\[\text{FIP_CONS}\]

3. $\overline{f} = \overline{f_i} \overline{f_j}$

\[\text{fields_in_path}(ctx, cl, ctxcld_2 .. ctxcld_k) = \overline{f}\]

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

\[\text{FIELDS_NONE}\]

1. $\text{find_path}(P, \tau) = \text{null}$

\[\text{fields}(P, \tau) = \text{null}\]

\[\text{FIELDS_SOME}\]

1. $\text{find_path}(P, \tau) = ctxcld$

\[\text{fields}(P, \tau) = \overline{f}\]
methods_in_path(\emptyset) = \text{meth} - \text{method names lookup in a path}

\text{MIP}_\text{CONS}

\text{MIP}_\text{EMPTY}

1. \text{class}_\text{methods}(\emptyset) = \text{meth}_\text{def}_1
2. \text{meth}_\text{def}_1 = \text{cl}_\text{meth}(\emptyset) \{ \text{meth}_\text{body}_1 \}
3. \text{methods}_\text{in}_\text{path}(\emptyset) = \text{meth}
4. \text{meth} = \text{meth}_1 \cdot \text{meth}

\text{methods}_\text{in}_\text{path}(\emptyset) = \text{meth}

\text{methods}(P, \tau) = \text{meth} - \text{method names lookup in a type}

\text{METHODS}_\text{METHODS}

\text{FTIF}_\text{EMPTY}

\text{FTIF}_\text{CONS}_\text{BOT}

\text{FTIF}_\text{CONS}_\text{TRUE}

\text{FTIF}_\text{CONS}_\text{FALSE}

\text{FTIF}_\text{EMPTY}

\text{FTIF}_\text{CONS}_\text{BOT}

\text{FTIF}_\text{CONS}_\text{TRUE}

\text{FTIF}_\text{CONS}_\text{FALSE}

\text{ftype}_\text{in}_\text{fds}(P, \text{ctxt}, \vec{f}_d, f) = \tau_{\text{opt}} - \text{field type lookup in a list}

\text{ftype}_\text{in}_\text{path}(_{\text{cxtcl}d, f}) = \tau_{\text{opt}} - \text{field type lookup in a path}

\text{ftype}(P, \tau, f) = \tau' - \text{field type lookup}

\text{FMDIL}_\text{EMPTY}

\text{FMDIL}_\text{CONS}_\text{TRUE}

\text{find}_\text{meth}_\text{def}_\text{in}_\text{list}(\text{meth}_\text{def}, \text{meth}) = \text{meth}_\text{def}_\text{opt} - \text{meth. def. lookup (list)}
\[
\begin{align*}
\text{FMDIL\_CONS\_FALSE} & \\
1. \text{meth\_def} = & \text{cl\_meth}\text{'(}\text{\textbackslash{}d}\{\text{meth\_body}\} & \text{2. meth} \neq & \text{meth'} \\
3. \text{find\_meth\_def\_in\_list(meth\_def}_2 \ldots \text{meth\_def}_k, \text{meth})} = & & \text{meth\_def\_opt} \\
\end{align*}
\]

\[
\begin{align*}
\text{find\_meth\_def\_in\_path(\text{ctx\_cld}, \text{meth})} = & \text{ctx\_meth\_def\_opt} & \text{meth\_def\_lookup (path)}
\end{align*}
\]

\[
\begin{align*}
\text{FMDIP\_EMPTY} & \\
\text{find\_meth\_def\_in\_path()} = & \text{null} & \text{FMDIP\_CONS\_TRUE} \\
\end{align*}
\]

\[
\begin{align*}
\text{find\_meth\_def\_in\_list(meth\_def}_2, \ldots , & \text{meth\_def}_k, \text{meth})} = & \text{meth\_def} \\
\text{find\_meth\_def\_in\_path(\text{ctx\_cld}_2 \ldots \text{ctx\_cld}_k, & \text{meth})} = & (\text{ctx\_meth}, \text{meth}) \\
\text{FMDIP\_CONS\_FALSE} & \\
\end{align*}
\]

\[
\begin{align*}
\text{find\_meth\_def\_in\_path(\text{ctx\_cld}_2 \ldots \text{ctx\_cld}_k, & \text{meth})} = \text{ctx\_meth\_def\_opt} \\
\text{find\_meth\_def\_in\_path(\text{ctx\_cld}_2 \ldots \text{ctx\_cld}_k, & \text{meth})} = \text{ctx\_meth\_def\_opt} \\
\end{align*}
\]

\[
\begin{align*}
\text{find\_meth\_def(P, \tau, \text{meth})} = & \text{ctx\_meth\_def\_opt} & \text{method\_def\_lookup in a type}
\end{align*}
\]

\[
\begin{align*}
\text{FMD\_NULL} & \\
1. \text{find\_path}(P, \tau) = & \text{null} & \text{FMD\_OPT} \\
\text{find\_meth\_def(P, \tau, \text{meth})} = & \text{null} & 1. \text{find\_path}(P, \tau) = & \text{ctx\_cld} \\
\text{find\_meth\_def(P, \tau, \text{meth})} = & \text{ctx\_meth\_def\_opt} & 2. \text{find\_meth\_def\_in\_path(ctx\_cld, \text{meth})} = & \text{ctx\_meth\_def\_opt} \\
\end{align*}
\]

\[
\begin{align*}
\text{mtype}(P, \tau, \text{meth}) = & \pi & \text{method\_type\_lookup}
\end{align*}
\]

\[
\begin{align*}
\text{mtype}(P, \tau, \text{meth}) = & \pi & \text{MTYPE} \\
1. \text{find\_meth\_def}(P, \tau, \text{meth}) = & (\text{ctx\_meth}_k, \text{meth}) \\
2. \text{meth\_def} = & \text{cl\_meth}(\text{\textbackslash{}t}_k \text{var}_k \ldots \{\text{meth\_body}\}) \\
3. \text{find\_type}(P, \text{ctx\_cld}_k, \text{cl}) = & \tau' \\
4. \text{find\_type}(P, \text{ctx\_cld}_k, \text{cl}) = & \tau_k \\
5. \pi = & \tau_k \rightarrow \tau' \\
\end{align*}
\]

\[
\begin{align*}
P \vdash & \tau \triangleq \tau' & \text{subtyping} \\
\text{STY\_OBJ} & \\
1. \text{find\_path}(P, \tau) = & \text{ctx\_cld} \\
\frac{}{P \vdash \tau \triangleq \text{Object}} & \text{STY\_DCL} \\
2. \text{find\_cld}(P, \text{mi}'\cdot\text{pn}', \text{pn}'\cdot\text{dcl}') = & \text{ctx\_cld} \\
3. \text{ctx\_cld} \in & \text{ctx\_cld} \\
\frac{}{P \vdash \tau \triangleq \text{mi}'\cdot\text{pn}'\cdot\text{dcl}'} & \\
\end{align*}
\]

\[
\begin{align*}
P \vdash & \tau \triangleq \tau' & \text{normal, multiple subtyping} \\
\text{STY\_MANY} & \\
1. \tau = & \tau_k^k \\
2. \tau' = & \tau_k^k \\
3. P \vdash & \tau_k \triangleq \tau_k' \\
\frac{}{P \vdash \tau \triangleq \tau'} & \\
\end{align*}
\]
\[
P \vdash \tau_{\text{opt}} \prec \tau_{\text{opt}}' \quad \text{option subtyping}
\]

**STY_OPTION**

1. \( \tau_{\text{opt}} = \tau \)
2. \( \tau_{\text{opt}}' = \tau' \)
3. \( P \vdash \tau \prec \tau' \)

\[
P, H \vdash \nu_{\text{opt}} \prec \tau_{\text{opt}} \quad \text{well-formed value}
\]

**WF_NULL**

1. \( \tau_{\text{opt}} = \tau \)

\[
P, H \vdash \text{null} \prec \tau_{\text{opt}}
\]

**WF_OBJECT**

1. \( P \vdash H (\text{oid}) \prec \tau_{\text{opt}} \)

\[
P, H \vdash \text{oid} \prec \tau_{\text{opt}}
\]

\[
P, \Gamma, H \vdash \text{L} \quad \text{well-formed variable state}
\]

**WF_VARSTATE**

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

\[
P, \Gamma, H \vdash \text{L}
\]

\[
P \vdash H \quad \text{well-formed heap}
\]

**WF_HEAP**

1. \( \text{finite}(\text{dom}(H)) \)

\[
\exists \tau. H (\text{oid}) = \tau \land \exists f. \text{fields}(P, \tau) = f \land \\
\forall f \in f. \exists \tau'. (P, H \vdash H (\text{oid}, f) \prec \tau')
\]

\[
P \vdash H
\]

\[
\Gamma \vdash \text{config} \quad \text{well-formed configuration}
\]

**WF_ALL**

1. \( \vdash P \)
2. \( P \vdash H \)
3. \( P, \Gamma, H \vdash \text{L} \)
4. \( P, \Gamma \vdash s_k \)

\[
\Gamma \vdash (P, L, H, \text{Exception}) \quad P, \Gamma \vdash s
\]

\[
P, \Gamma \vdash s \quad \text{well-formed statement}
\]

**WF_BLOCK**

1. \( P, \Gamma \vdash s_k \)

\[
P, \Gamma \vdash \{ \, s_k \}
\]

**WF_VAR_ASSIGN**

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

\[
P, \Gamma \vdash \text{var} = x.f;
\]

**WF_FIELD_READ**

1. \( P \vdash \Gamma(x) \prec \Gamma(\text{var}) \)
2. \( P \vdash \tau \prec \Gamma(\text{var}) \)

\[
P, \Gamma \vdash \text{var} = \text{new}_{\text{ctx}}(\text{cl});
\]

**WF_FIELD_WRITE**

1. \( P \vdash \Gamma(x) \prec \Gamma(\text{var}) \)
2. \( P \vdash \tau \prec \Gamma(\text{var}) \)
3. \( P, \Gamma \vdash \text{new}_{\text{ctx}}(\text{cl}) \)

\[
P, \Gamma \vdash \text{var} = x.\text{meth}(\text{g});
\]

**WF_IF**

1. \( P \vdash \Gamma(x) \prec \Gamma(y) \lor \Gamma(\text{g}) \prec \Gamma(\text{g}) \)

\[
P, \Gamma \vdash \text{if}(x == y) s_1 \text{ else } s_2
\]

**WF_JF**

1. \( P \vdash \Gamma(x) \prec \Gamma(\text{var}) \lor \Gamma(y) \prec \Gamma(\text{var}) \)
2. \( P \vdash \tau \prec \Gamma(\text{var}) \)

\[
P, \Gamma \vdash \text{var} = \text{new}_{\text{ctx}}(\text{cl});
\]

**WF_MCALL**

1. \( \text{y} = y_k \)
2. \( \Gamma(z) = \tau \)
3. \( \text{mtype}(P, \tau, \text{meth}) = \pi_k \to \tau' \)
4. \( P \vdash \Gamma(y_k) \prec \tau_k \)
5. \( P \vdash \tau' \prec \Gamma(\text{var}) \)

\[
P, \Gamma \vdash \text{var} = x.\text{meth}(\text{g});
\]
\[
P \vdash \text{meth}_k \rightarrow \text{method} \quad \text{well-formed method in } \tau
\]

**WF_METHOD**

1. \(\text{distinct}(\text{var}_k^k)\)
2. \(\text{find_type}(P, \text{ctx}, \text{cl}_k) = \tau_k^k\)
3. \(\Gamma = [\text{var}_k^i \rightarrow \tau_k^i\{\text{this} \rightarrow \text{ctx.dcl}\}\]
4. \(P, \Gamma \vdash s_i\)
5. \(\text{find_type}(P, \text{ctx}, \text{cl}) = \tau\)
6. \(P \vdash \Gamma(y) < \tau\)

\[
P \vdash_{\text{ctx.dcl}} \text{cl} \text{meth}_{\text{cl} \text{var}_k^k}\{\overline{\tau_i}\ \text{return} \ y; \}
\]

**WF_CLASS**

\[
P \vdash_{\text{ctx}} (\text{cl}, \text{ct}, \text{fd}, \text{meth}_k) \quad \text{well-formed class in } \text{ctx} \quad \text{(generic rule)}
\]

1. \(\text{find_type}(P, \text{ctx}, \text{cl}) = \tau\)
2. \(\text{ctx.dcl} \neq \tau\)
3. \(\text{distinct}(\overline{\text{ct}_j^j})\)
4. \(\text{fields}(P, \tau) = \overline{\text{ct}_j^j} \perp \overline{\text{ct}}\)
5. \(\text{find_type}(P, \text{ctx}, \text{cl}_j^j) = \tau_j^j\)
6. \(P \vdash_{\text{ctx.dcl}} \text{meth}_k^k\)
7. \(\text{method_name}(\text{meth}_k^k) = \text{meth}_k^k\)
8. \(\text{distinct}(\overline{\text{meth}_k^k})\)
9. \(\text{methods}(P, \tau) = \overline{\text{meth}_l^l}\)
10. \(\text{mtype}(P, \text{ctx.dcl}, \text{meth}_l^l) = \overline{\pi_l^l}\)
11. \(\text{mtype}(P, \tau, \text{meth}_l^l) = \overline{\pi_l^l}\)
12. \(\text{meth}_l^l \in \text{meth}_k^k \rightarrow \pi_l^l = \pi_l^l\)

\[
P \vdash_{\text{ctx}} (\text{cl}, \text{ct}, \text{ct}_j^j, \text{meth}_k^k)
\]

**WF_PACKAGE**

\[
P \vdash_{\text{mi}} \text{cld} \quad \text{well-formed class in } \text{mi}
\]

1. \(\text{full_name}(\text{cld}_k^k) = \text{fqn}_k^k\)
2. \(\text{distinct}(\overline{\text{fqn}_k^k})\)
3. \(P \vdash_{\text{mi}} \text{cld}_k^k\)
4. \(\text{acyclic_clds}_m P\)

\[
P \vdash_{\text{mi}} \text{module} \{\text{cl}_k^k \overline{\text{fqn}}\}
\]

**WF_MODULE**

\[
MH \vdash \phi \quad \text{well-formed module instance cache}
\]

**WF_RMIS**

1. \(\text{ran}(\phi) \subseteq \text{dom}(MH)\)

\[
MH \vdash \phi
\]

**WF_BOOTSTRAP_REP**

**WF_NORMAL_REP**

1. \(\text{find_mds_in_mds}(\text{md}^\phi, \text{core_m}) = \text{md}^\phi\)
2. \(MH \vdash \phi\)

\[
(RC, MH) \vdash \text{bootstrap repository} \{\text{md}^\phi; \phi\}
\]

1. \(r \neq \text{rn}\)
2. \(\text{rn} \in \text{dom}(RC)\)
3. \(MH \vdash \phi\)

\[
(RC, MH) \vdash \text{repository} \ r \ \text{child of} \ \text{rn} \{\text{md}^\phi; \phi\}
\]
\( \text{MH} \vdash \text{RC} \) – well-formed repository context

\[
\begin{align*}
1. \forall \text{rn} \in \text{dom}(\text{RC}). \exists R. & \quad (\text{RC}(\text{rn}) = R \land \text{R.name}(R) = \text{rn} \land \\
2. \text{bootstrap.r} \in \text{dom}(\text{RC}) & \quad \text{(RC, MH)} \vdash R
\end{align*}
\]

\( \text{RC} \vdash \text{MH} \) – well-formed module hierarchy

\[
\begin{align*}
1. \text{acyclic.mh} & \quad \text{MH} \\
2. \forall \text{mi} \in \text{dom}(\text{MH}). \exists \text{md mi.} & \quad (\text{RC}, \text{MH}) \vdash \text{mi md}
\end{align*}
\]

\( \vdash P \) – well-formed program

\[
\begin{align*}
1. \text{MH} & \vdash \text{RC} \\
2. \text{RC} & \vdash \text{MH} \\
3. \text{acyclic.clds}(\text{RC}, \text{MH}) & \quad \vdash (\text{RC}, \text{MH})
\end{align*}
\]

\( \text{find}\_\text{pkg.clds}(\text{cld}_1, \text{pn}) = \text{cld}_2 \) –

\[
\begin{align*}
\text{find}\_\text{pkg.clds}(\text{pn}) = & \quad \text{FPC._\text{CONS_TRUE}} \\
1. \text{cld}_\text{c} = \text{package} \text{ pn; am class dcl extends cl}\{\text{fdmeth\_def}_\text{c}\} & \quad \text{FPC._\text{CONS_FALSE}} \\
2. \text{pn} \notin \text{pn} & \\
3. \text{find}\_\text{pkg.clds}(\text{cld}_2, \text{pn}) = \text{cld}_\text{c} & \quad \text{find}\_\text{pkg.clds}(\text{cld}_2, \text{pn}) = \text{cld}_\text{c} \\
\end{align*}
\]

\( \text{SRC} \vdash \text{mf} \rightsquigarrow \text{md}_\text{c} \) – packaging a module file to a module def., compile-time code

\[
\begin{align*}
1. \text{find}\_\text{pkg.clds}(\text{cld}_1, \text{pn}_j) = \text{cld}_2 & \quad \text{PCG._\text{MF}} \\
\text{cld}_1 \vdash \text{superpackage} \text{ mn}\{\text{member} \text{ pn}_j; \text{import} \text{ m}_k; \text{export} \text{ fqn}_l; \} & \rightsquigarrow \text{module} \text{ mn}\{\text{cld}_2 \text{ m}_k \text{ fqn}_l; \}
\end{align*}
\]

\( \text{ctx} s \rightsquigarrow s \) – context insertion for a statement

\[
\begin{align*}
\text{CL\_\text{S\_\text{BLOCK}}:} & \quad \text{CL\_\text{S\_\text{VAR\_ASSIGN}}:} \quad \text{CL\_\text{S\_\text{FIELD\_READ}}:} \quad \text{CL\_\text{S\_\text{FIELD\_WRITE}}:} \\
\text{ctx} \text{ s}_k \rightsquigarrow s_k & \quad \text{ctx} \text{ var} = x; \rightsquigarrow \text{var} = x; & \quad \text{ctx} \text{ var} = x.f; \rightsquigarrow \text{var} = x.f; & \quad \text{ctx} x.f = y; \rightsquigarrow x.f = y; \\
\text{ctx} \text{ s}_k \rightsquigarrow s_1 & \quad \text{ctx} \text{ s}_1 \rightsquigarrow s_1 & \quad \text{ctx} \text{ if} (x == y) s_1 \text{ else } s_2 \rightsquigarrow \text{ctx} \text{ if} (x == y)s_1 \text{ else } s_2 & \quad \text{ctx} \text{ var} = x.\text{meth}(y_k); \rightsquigarrow \text{var} = x.\text{meth}(y_k); \\
\text{ctx} \text{ new} \text{ cl}(); & \quad \text{ctx} = \text{new} \text{ cl}();
\end{align*}
\]
\[ \vdash_{\text{ctx}} \text{meth}_i \text{def}_i^c \rightsquigarrow \text{meth}_i \text{def} \]  – context insertion for method def.’s

\[
\begin{align*}
\vdash_{\text{ctx}} & s^c \rightsquigarrow s^k \\
\vdash_{\text{ctx}} & \text{cl}\{\text{meth}_i \text{def}_i^c k\} \rightsquigarrow \text{cl}\{\text{meth}_i \text{def}_i^c k\}
\end{align*}
\]

\[ \vdash_{\text{mi}} \text{cld}_i^c \rightsquigarrow \text{cld} \]  – context insertion for class def.’s

\[
\begin{align*}
\vdash_{\text{mi}} & \text{cld}_i^c = \text{package } p_n; \text{ am class } dcl \text{ extends } \text{cl}\{\text{fd}_i \text{meth}_i \text{def}_i^c k\}
\end{align*}
\]

\[ \vdash_{\text{mi}} \text{md}_i^c \rightsquigarrow \text{md} \]  – module def. translation

\[
\begin{align*}
\vdash_{\text{mi}} & \text{module} mn\{\text{cld}_i^c k \text{ fqn}\} \rightsquigarrow \text{module} mn\{\text{cld}_i^c k \text{ fqn}\}
\end{align*}
\]

\[ \theta \vdash s \rightsquigarrow s’ \]  – variable translation for a statement

\[
\begin{align*}
\text{TR}_S\_\text{BLOCK} \quad &1. \; \theta \vdash s^k \rightsquigarrow s^k \\
\text{TR}_S\_\text{FIELD}\_\text{READ} \quad &1. \; \theta(\text{var}) = \text{var’} \\
&2. \; \theta(x) = x’ \\
&3. \; \theta(\text{y}) = y’ \\
&4. \; \theta \vdash s_2 \rightsquigarrow s_2’ \\
&5. \; \theta \vdash x.f = y; \rightsquigarrow x’.f = y’;
\end{align*}
\]

\[
\begin{align*}
\text{TR}_S\_\text{NEW} \quad &1. \; \theta(\text{var}) = \text{var’} \\
&2. \; \theta(x) = x’ \\
&3. \; \theta(\text{y}) = y’ \\
&4. \; \theta \vdash x.f = y; \rightsquigarrow x’.f = y’;
\end{align*}
\]

\[
\begin{align*}
\text{TR}_S\_\text{IF} \quad &1. \; \theta(\text{var}) = \text{var’} \\
&2. \; \theta(x) = x’ \\
&3. \; \theta(\text{y}) = y’ \\
&4. \; \theta \vdash s_2 \rightsquigarrow s_2’ \\
\end{align*}
\]

\[
\begin{align*}
\text{TR}_S\_\text{MCALL} \quad &1. \; \theta(\text{var}) = \text{var’} \\
&2. \; \theta(x) = x’ \\
&3. \; \theta(\text{y}) = y’ \\
&4. \; \theta \vdash x.f = y; \rightsquigarrow x’.f = y’;
\end{align*}
\]

\[ \text{config} \rightarrow \text{config’} \]  – reduction of a statement

\[
\begin{align*}
\text{R}_\text{BLOCK} \quad &1. \; L(x) = v \\
&2. \; L(y) = w \\
&3. \; v = w \\
&4. \; \text{fields } (P, \tau) = f_k^c \\
&5. \; \text{oid} \notin \text{dom } (H) \\
&6. \; H’ = H[\text{oid} \mapsto (\tau, f_k \mapsto \text{null}^k)]
\end{align*}
\]

\[
\begin{align*}
\text{R}_\text{NEW} \quad &1. \; L(x) = v \\
&2. \; L(y) = w \\
&3. \; v \neq w \\
&4. \; \text{if } (x = y) s_1 \text{ else } s_2 s_1^f &\rightarrow (P, L, H, s_2 s_1^f)
\end{align*}
\]

\[
\begin{align*}
\text{R}_\text{FIELD}\_\text{READ}\_\text{NPE} \quad &1. \; L(x) = \text{null} \\
&2. \; H(\text{oid}, f) = v \\
&3. \; \text{oid} \notin \text{dom } (H) \\
&4. \; H’ = H[\text{oid} \mapsto (\tau, f_k \mapsto \text{null}^k)]
\end{align*}
\]

\[
\begin{align*}
\text{R}_\text{FIELD}\_\text{WRITE}\_\text{NPE} \quad &1. \; L(x) = \text{null} \\
&2. \; L(y) = v \\
&3. \; \text{oid} \notin \text{dom } (H) \\
&4. \; H’ = H[\text{oid} \mapsto (\tau, f_k \mapsto \text{null}^k)]
\end{align*}
\]

\[
\begin{align*}
\text{R}_\text{FIELD}\_\text{READ}\_\text{TRUE} \quad &1. \; L(x) = v \\
&2. \; L(y) = w \\
&3. \; v \neq w \\
&4. \; \text{if } (x = y) s_1 \text{ else } s_2 s_1^f &\rightarrow (P, L, H, s_2 s_1^f)
\end{align*}
\]

\[
\begin{align*}
\text{R}_\text{FIELD}\_\text{WRITE}\_\text{TRUE} \quad &1. \; L(x) = \text{null} \\
&2. \; L(y) = v \\
&3. \; \text{oid} \notin \text{dom } (H) \\
&4. \; H’ = H[\text{oid} \mapsto (\tau, f_k \mapsto \text{null}^k)]
\end{align*}
\]

\[
\begin{align*}
\text{R}_\text{NEW} \quad &1. \; \text{if } (x = y) s_1 \text{ else } s_2 s_1^f &\rightarrow (P, L, H, s_2 s_1^f)
\end{align*}
\]
\[ L(x) = \textbf{null} \]

\[ (P, I, H, \text{var} = x.\text{meth}(\text{var}^k); \pi^1) \rightarrow (P, I, H, \text{NPE}) \]

1. \( L(x) = \text{oid} \)  
2. \( H(\text{oid}) = \tau \)
3. \( \text{find\_meth\_def}(P, \tau, \text{meth}) = (\text{ctx}, \text{cl\_meth}(\text{ctx\_var}^k)\{s_j^j, \text{return} \ y; \}) \)
4. \( \text{var}^k \downarrow \text{dom} (I) \)  
5. \( \text{distinct} (\text{var}^k) \)  
6. \( x' \notin \text{dom} (I) \)
7. \( x' \notin \text{var}^k \)
8. \( L(y_k) = v_k \)
9. \( L' = L[\text{var}^k \mapsto v_k] \quad \{x' \mapsto \text{oid} \} \)
10. \( \theta = [\text{var}_k \mapsto \text{var}^k] \quad \{\text{this} \mapsto x' \} \)  
11. \( \theta \vdash s_j^j \leadsto s_j^{\pi^3} \)
12. \( \theta(y) = y' \)

\[ (P, I, H, \text{var} = x.\text{meth}(\text{var}^k); \pi^1) \rightarrow (P, I', H, s_j^{\pi^3} \text{var} = y'; \pi^1) \]

\[ \text{config} \rightarrow \text{config}' \quad \text{— reduction of internal actions} \]

**R\_MCALL\_NPE**

\[ \text{R\_MCALL} \]

**R\_ACTION\_LIST**

**R\_NO\_ACTION**

**R\_INSTALL**

**R\_UNINSTALL**

**R\_EXISTING\_INSTANCE**

**R\_NEW\_INSTANCE**
\[ \text{config} \xrightarrow{a} \text{config} \] – reduction of an administrator action

**ADMIN_INSTALL**

1. \( ((RC, MH), L, H, \overline{s}_{1}) \xrightarrow{\text{rn. install}(md)} ((RC', MH), L, H, \overline{s}_{1}) \)
2. \( ((RC, MH), L, H, \overline{s}_{1}) \xrightarrow{\text{rn. install}(md)} ((RC', MH), L, H, \overline{s}_{1}) \)

**ADMIN_UNINSTALL**

1. \( ((RC, MH), L, H, \overline{s}_{1}) \xrightarrow{\text{rn. uninstall}(m)} ((RC', MH), L, H, \overline{s}_{1}) \)
2. \( ((RC, MH), L, H, \overline{s}_{1}) \xrightarrow{\text{rn. uninstall}(m)} ((RC', MH), L, H, \overline{s}_{1}) \)

**ADMIN_NEW_INSTANCE**

1. \( ((RC, MH), L, H, \overline{s}_{1}) \xrightarrow{\text{mi uninistalled}(m)} ((RC', MH'), L, H, \overline{s}_{1}) \)
2. \( ((RC, MH), L, H, \overline{s}_{1}) \xrightarrow{\text{mi initialisation}(m)} ((RC', MH'), L, H, \overline{s}_{1}) \)

\[(P, m, P') \in w_P \text{ change} \] – well-formed program change (proof related)

**WRC_INSTALL**

1. \( \vdash (RC, MH) \)
2. \( m \notin \text{dom}(MH) \)
3. \( RC(rn) = R \)
4. \( R.body([R] = (md, \phi)) \)
5. \( md \_\_name(md) = m \)
6. \( R\_\_update(R, md, \#md, \phi) = R' \)

\( ((RC, MH), m, (RC[\overline{rn} \rightarrow R'_1, MH]) \in w_P \text{ change} \)

**WRC_UNINSTALL**

1. \( \vdash (RC, MH) \)
2. \( m \notin \text{dom}(MH) \)
3. \( RC(rn) = R \)
4. \( R.body([R] = (md, \phi)) \)
5. \( find\_\_md\_in\_mds(md, [R] = m) = md \)
6. \( mds \_\_rmd([md], md) = md \)
7. \( R\_\_update(R, md, \phi \setminus md) = R' \)

\( ((RC, MH), m, (RC[\overline{rn} \rightarrow R'_1, MH]) \in w_P \text{ change} \)

**WRC_NEW_INSTANCE**

1. \( \vdash (RC, MH) \)
2. \( m \notin \text{dom}(MH) \)
3. \( RC(rn) = R \)
4. \( R.body([R] = (md, \phi)) \)
5. \( m \subseteq \text{dom}(MH) \)
6. \( md \_\_name(md) = m \)
7. \( R\_\_update(R, md, \phi \setminus m, md) = R' \)
8. \( RC' = RC[\overline{rn} \rightarrow R'_1] \)
9. \( MH' = MH [m \rightarrow (md, \overline{m})] \)
10. \( (RC', MH') \vdash m, md \)

\( ((RC, MH), m, (RC', MH') \in w_P \text{ change} \)