Welterweight Fortress DRAFT

David Chase
Oracle Labs
david.r.chase@oracle.com

Justin Hilburn
Oracle Labs
justin.hilburn@oracle.com

Victor Luchangco
Oracle Labs
victor.luchangco@oracle.com

Karl Naden
Oracle Labs
karl.naden@oracle.com

Sukyoung Ryu
KAIST
sryu.cs@kaist.ac.kr

Guy L. Steele Jr.
Oracle Labs
guy.steele@oracle.com

John Tristan
Oracle Labs
jean.baptiste.tristan@oracle.com

Abstract

Fortress [1]

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features—classes and objects, inheritance, modules, packages, polymorphism

General Terms Languages

Keywords object-oriented programming, multiple dispatch, symmetric dispatch, multiple inheritance, overloading, ilks, run-time types, static types, components, modularity, meet rule, methods, multimethods, separate compilation, Fortress

1. Introduction

OOPSLA '11 October 22–27, 2011, Portland, Oregon, USA.

2. Notation

We use the term *monogram* to refer to a single letter (Latin or Greek) that, rather than being used for decorative purposes, is itself possibly "decorated" with one or more prime marks and/or a sequence of one or more subscripts. Examples of monograms are x, β, e', α_2, and $\tau'_{15\,27}$.

We write $\overline{x}$ as shorthand for a possibly empty comma-separated sequence $x_1, x_2, \ldots, x_n$ for some freely chosen nonnegative integer n; thus $\overline{x}$ may expand to " " or "x_1" or "x_1, x_2" or "x_1, x_2, x_3" or "x_1, x_2, x_3, x_4" and so on. More generally, for any expression, that same expression with an overbar is shorthand for a possibly empty comma-separated sequence of copies of that expression with two transformations applied to each copy: (a) any subexpression that is underlined one or more times is replaced by a copy of that subexpression with one underline removed, and (b) any subexpression that is a monogram that is not underlined is replaced by a copy of that monogram with an additional subscript i appended, where i is the number of the copy (starting from the left with 1). Thus $\overline{\underline{[\tau/P]}\tau'}$ means $[\tau/P]\tau'_1, [\tau/P]\tau'_2, \ldots, [\tau/P]\tau'_n$, so one possible concrete expansion is $[\tau/P]\tau'_1, [\tau/P]\tau'_2, [\tau/P]\tau'_3, [\tau/P]\tau'_4, [\tau/P]\tau'_5$. If overbar constructions are nested, they are expanded outermost first. Therefore the shorthand $f\,[\![\,\overline{P <: \{\overline{\tau}\}}\,]\!]$ means $f\,[\![\,P_1 <: \{\overline{\tau_1}\}, P_2 <: \{\overline{\tau_2}\}, \ldots, P_n <: \{\overline{\tau_n}\}\,]\!]$, which in turn means:

$$
\begin{aligned}
f\,[\![\,&P_1 <: \{\tau_{1\,1}, \tau_{1\,2}, \ldots, \tau_{1\,n_1}\},\\
&P_2 <: \{\tau_{2\,1}, \tau_{2\,2}, \ldots, \tau_{2\,n_2}\},\\
&\ldots,\\
&P_n <: \{\tau_{m\,1}, \tau_{m\,2}, \ldots, \tau_{m\,n_m}\}\,]\!]
\end{aligned}
$$

so one possible concrete expansion is:

$$
\begin{aligned}
f\,[\![\,&P_1 <: \{\tau_{1\,1}, \tau_{1\,2}, \tau_{1\,3}\},\\
&P_2 <: \{\tau_{2\,1}\},\\
&P_3 <: \{\,\},\\
&P_4 <: \{\tau_{4\,1}, \tau_{4\,2}, \tau_{4\,3}, \tau_{4\,4}, \tau_{4\,5}, \tau_{4\,6}\}\,]\!]
\end{aligned}
$$

Note the use of whitespace between subscripts so that $\tau_{4\,12}$ is clearly different from $\tau_{41\,2}$.

The function # returns an integer saying how many arguments it was given; thus $\#(\overline{x})$ tells the length of the sequence into which $\overline{x}$ has expanded.

After all occurrences of the overbar construction have been expanded, three other shorthand substitutions take place:

- Every monogram whose base letter has been described as *ranging over* a BNF nonterminal of a specified grammar is replaced by a token sequence generated by the grammar from that nonterminal.
- Each occurrence of the symbol "_" is replaced by any sequence of tokens that is correctly balanced within respect to parentheses, braces, and brackets of all kinds, such that every comma or semicolon in the sequence is contained within at least one matched pair of parentheses, braces, or brackets. (This is used as a "don't care" indication when asking whether any of a set of constructs matches a certain syntactic pattern.)
- Each occurrence of the symbol "•" is deleted. (This symbol is used as an explicit indication that an empty sequence of symbols is intended.)

When any of these shorthands is used in an overall context (such as a BNF rule, inference rule, axiom, or expository sentence or paragraph), it is as if there were an infinite number of instantiations of that context, one for each possible expansion of the shorthand. Three consistency constraints must be obeyed in performing the substitutions for any single such context:

- If the same monogram (with identical decorations) has an additional subscript attached to it by more than one overbar construction, then all such overbar constructions are constrained to produce the same number of copies in any given instantiation of the rule; otherwise the choices for the number of copies produced by each overbar construction is free and independent.
- If the base letter of a monogram ranges over a BNF nonterminal, then multiple identical occurrences of the monogram must be replaced by identical copies of a single generated token sequence.
- If two distinct monograms each have base letters that range over a BNF nonterminal that expands to simply "identifier", then they must be replaced with different identifiers.

The last two constraints rely on metavariable declarations such as those in Figure 1. A declaration such as "e ranges over expressions e" means "monograms with base letter e expand into expressions generated by BNF nonterminal e"; this may seem redundant, but only because by convention we frequently use a single-letter identifier as a BNF nonterminal and then go on to use that same single-letter identifier as a base letter for monograms. A declaration such as "$\alpha, \gamma, \rho, \chi, \eta$ range over lattice types α" means "monograms with base letter α or γ or ρ or χ or η expand into expressions generated by BNF nonterminal α" which is more clearly not a redundant statement.

As an additional convenience using these shorthands, we adopt these conventions:

- If a judgment has several comma-separated expressions to the right of the turnstile "$\vdash$", it is as if there were several distinct judgments, one containing each of the expressions to the right of the turnstile. Thus the judgment $\Gamma \vdash \tau_1 <: \tau_1', \tau_2 <: \tau_2', \tau_3 <: \tau_3'$ means the same as three separately written judgments:

$$\Gamma \vdash \tau_1 <: \tau_1' \qquad \Gamma \vdash \tau_2 <: \tau_2' \qquad \Gamma \vdash \tau_3 <: \tau_3'$$

- If a judgment has nothing to the right of the turnstile, it is as if there were no judgment written at all.
- If an inference rule has several comma-separated expressions or judgments as consequents, it is as if there were several distinct inference rules, one containing each of the consequents.
- If an inference rule has no consequents, it is as if there were no inference rule written at all.

As an extreme (but useful) example of the application of these conventions, consider this axiom:

$$\Delta \vdash \overline{\underline{E}\big[\pi(\overline{\underline{v}})\big] \longrightarrow \underline{E}[v]}$$

In order to apply this axiom to a particular case, we may freely choose to expand the largest overbar construction to produce, say, two copies:

$$\Delta \vdash E\big[\pi_1(\overline{v})\big] \longrightarrow E[v_1], E\big[\pi_2(\overline{v})\big] \longrightarrow E[v_2]$$

Note that both the π symbol and the second occurrence of v receive subscripts in each copy, but the underlines (which are removed as part of the expansion process) prevent the first occurrence of v (which happens to have a second overbar) and the two occurrences of E from receiving subscripts. Now we expand the remaining overbars, but because they will attach subscripts to the symbol v, and v has already had subscripts attached by the larger overbar, we must choose the same number of copies (two) for each of these overbars:

$$\Delta \vdash E\big[\pi_1(v_1, v_2)\big] \longrightarrow E[v_1], E\big[\pi_2(v_1, v_2)\big] \longrightarrow E[v_2]$$

Then this judgment with two comma-separated expressions to the right of the turnstile is understood to mean two distinct judgments:

$$\Delta \vdash E\big[\pi_1(v_1, v_2)\big] \longrightarrow E[v_1]$$
$$\Delta \vdash E\big[\pi_2(v_1, v_2)\big] \longrightarrow E[v_2]$$

A final note: we sometimes use parentheses or braces or brackets of different sizes within an expression purely to enhance readability; the size of such a symbol does not affect its meaning in the formalism.

3. Grammar

p	$::=$	$\overline{\delta}, e$	program (declarations plus expression)
δ	$::=$	$\texttt{trait}\ T\llbracket\overline{V\ \beta}\rrbracket <: \{\overline{t}\} \lozenge \{\overline{t}\} \equiv\bigcup \{\overline{c}\}\ \overline{\mu}\ \texttt{end}$	trait declaration
	$\mid$	$\texttt{object}\ O\llbracket\overline{\beta}\rrbracket(\overline{z{:}\,\tau}) <: \{\overline{t}\}\ \overline{\mu}\ \texttt{end}$	object declaration
	$\mid$	$f\llbracket\overline{\beta}\rrbracket(\overline{x{:}\,\tau}){:}\,\tau = e$	function declaration
V	$::=$	$\texttt{covariant} \mid \texttt{contravariant} \mid \texttt{invariant}$	variance
μ	$::=$	$m\llbracket\overline{\varphi}\rrbracket(\overline{x{:}\,\tau}){:}\,\tau = e$	method declaration
β	$::=$	$P <: \{\overline{\tau}\}$	simple type parameter binding
φ	$::=$	$\{\overline{\tau}\} <: P <: \{\overline{\tau}\}$	full type parameter binding
e	$::=$	x	variable reference
	$\mid$	$\texttt{self}$	self reference
	$\mid$	$(\overline{e})$	tuple creation
	$\mid$	$\pi_i(e)$	tuple projection
	$\mid$	$((\overline{x{:}\,\tau}){:}\,\tau \Rightarrow e)$	function creation
	$\mid$	$e\texttt{@}(\overline{e})$	function application
	$\mid$	$e.z$	field reference
	$\mid$	$O\llbracket\overline{\tau}\rrbracket(\overline{e})$	object creation
	$\mid$	$f\llbracket\overline{\tau}\rrbracket(\overline{e})$	function invocation with static arguments
	$\mid$	$f(\overline{e})$	function invocation, no static arguments
	$\mid$	$e.m\llbracket\overline{\tau}\rrbracket(\overline{e})$	method invocation with static arguments
	$\mid$	$e.m(\overline{e})$	method invocation, no static arguments
	$\mid$	$(e\ \texttt{match}\ x{:}\,\tau \Rightarrow e\ \texttt{else}\ e)$	match expression

τ	$::=$	P	type parameter reference
	$\mid$	c	constructed type
	$\mid$	$(\overline{\tau})$	tuple type
	$\mid$	$(\tau \rightarrow \tau)$	arrow type
	$\mid$	Any	special Any type
	$\mid$	Object	special Object type
c	$::=$	$O\llbracket\overline{\tau}\rrbracket$	object type
	$\mid$	t	trait type
t	$::=$	$T\llbracket\overline{\tau}\rrbracket$	trait type
P	$::=$	identifier	type parameter name
T	$::=$	identifier	generic trait name
O	$::=$	identifier	generic object name
x	$::=$	identifier	variable name
z	$::=$	identifier	field name
f	$::=$	identifier	function name
m	$::=$	identifier	method name
i	$::=$	integer	fixed integer

Figure 2. Grammar for Welterweight Fortress

δ ranges over top-level declarations δ
V ranges over variances V
μ ranges over method declarations μ
β ranges over simple type parameter bindings β
φ ranges over full type parameter bindings φ
e ranges over expressions e
t ranges over trait types t
c ranges over constructed types c
P, Q, S range over type parameter names P
T, A, B range over trait names T
O ranges over object names O
x, y range over variable names x
z ranges over field names z
f ranges over function names f
m ranges over method names m
τ, ζ, ξ, ω range over types τ
$\alpha, \gamma, \rho, \chi, \eta$ range over lattice types α
κ ranges over quantified types κ
λ ranges over lattice type parameter bindings λ
ψ ranges over general type environment entries ψ
v ranges over values v
E ranges over evaluation contexts E
R ranges over redexes R

π and σ do not range over any BNF nonterminal

Figure 1. Metavariables

α	$::=$	P	type parameter name
	$\mid$	$T\llbracket\overline{\alpha}\rrbracket$	trait type
	$\mid$	$O\llbracket\overline{\alpha}\rrbracket$	object type
	$\mid$	$(\overline{\alpha})$	tuple type
	$\mid$	$(\alpha \rightarrow \alpha)$	arrow type
	$\mid$	Any	special Any type
	$\mid$	Object	special Object type
	$\mid$	Bottom	special Bottom type
	$\mid$	$(\alpha \cup \alpha)$	union type
	$\mid$	$(\alpha \cap \alpha)$	intersection type
κ	$::=$	α	lattice type
	$\mid$	$\exists\llbracket\overline{\lambda}\rrbracket\alpha$	existentially quantified type
	$\mid$	$\forall\llbracket\overline{\lambda}\rrbracket\alpha$	universally quantified type
λ	$::=$	$\{\overline{\alpha}\} <: P <: \{\overline{\alpha}\}$	lattice type parameter binding
ψ	$::=$	δ	program declaration
	$\mid$	λ	lattice type parameter binding
Δ	$::=$	$\overline{\psi}$	type-declaration environment
Γ	$::=$	$\overline{x{:}\,\alpha}$	variable-type environment

Figure 3. Symbols Not Used in the Concrete Syntax

Evaluation rules: $\boxed{\Delta \vdash E[R] \longrightarrow E[e]}$

$$\Delta \vdash \overline{\underline{E}\big[\pi(\underline{\overline{v}})\big] \longrightarrow \underline{E}[v]} \qquad \text{(R-PROJECT-TUPLE)}$$

$$\Delta \vdash (v) \longrightarrow v \qquad \text{(R-SINGLETON-TUPLE)}$$

$$\Delta \vdash E\big[((\overline{x\colon\tau})\colon\tau \Rightarrow e)\texttt{@}(\overline{v})\big] \longrightarrow E\big[[\overline{v/x}]e\big] \qquad \text{(R-APPLY-FUNCTION)}$$

$$\frac{\texttt{object}\ O[\![\_]\!](\overline{z\colon\_})\ \_\ \texttt{end} \in \{\Delta\}}{\Delta \vdash \overline{\underline{E}\big[\,\underline{O[\![\overline{\alpha}]\!](\overline{v})}.z\big] \longrightarrow \underline{E}[v]}} \qquad \text{(R-FIELD-ACCESS)}$$

$$\frac{msa(\Delta, f[\![\overline{\tau}]\!], (\overline{ilk(v)})) = \big\{\big(f[\![\overline{P <\colon \{\_\}}]\!](\overline{x\colon\alpha})\colon\rho = e, \_\big)\big\}}{\Delta \vdash E\big[f[\![\overline{\tau}]\!](\overline{v})\big] \longrightarrow E\big[[\overline{v/x}][\overline{\tau/P}]e\big]} \qquad \text{(R-FUNCTION-STATIC-ARGS)}$$

$$\frac{msa(\Delta, f, (\overline{ilk(v)})) = \big\{\big(f[\![\overline{P <\colon \{\_\}}]\!](\overline{x\colon\alpha})\colon\rho = e, \_\big)\big\}}{\Delta \vdash E\big[f(\overline{v})\big] \longrightarrow E\big[[\overline{v/x}][\overline{\tau/P}]e\big]} \qquad \text{(R-FUNCTION-NO-STATIC-ARGS)}$$

$$\frac{msa(\Delta, O[\![\overline{\gamma}]\!], m[\![\overline{\tau}]\!], (\overline{ilk(v)})) = \big\{\big(m[\![\overline{\{\_\} <\colon P <\colon \{\_\}}]\!](\overline{x\colon\alpha})\colon\rho = e, \_\big)\big\}}{\Delta \vdash E\big[O[\![\overline{\gamma}]\!](\overline{v}).m[\![\overline{\tau}]\!](\overline{v'})\big] \longrightarrow E\big[[O[\![\overline{\gamma}]\!](\overline{v})/\texttt{self}][\overline{v'/x}][\overline{\tau/P}]e\big]} \qquad \text{(R-METHOD-STATIC-ARGS)}$$

$$\frac{msa(\Delta, O[\![\overline{\gamma}]\!], m, (\overline{ilk(v)})) = \big\{\big(m[\![\overline{\{\_\} <\colon P <\colon \{\_\}}]\!](\overline{x\colon\alpha})\colon\rho = e, \_\big)\big\}}{\Delta \vdash E\big[O[\![\overline{\gamma}]\!](\overline{v}).m(\overline{v'})\big] \longrightarrow E\big[[O[\![\overline{\gamma}]\!](\overline{v})/\texttt{self}][\overline{v'/x}][\overline{\tau/P}]e\big]} \qquad \text{(R-METHOD-NO-STATIC-ARGS)}$$

$$\frac{\Delta \vdash ilk(v) <\colon \tau}{\Delta \vdash E\big[(v\ \texttt{match}\ x\colon\tau \Rightarrow e\ \texttt{else}\ e')\big] \longrightarrow E\big[[v/x]e\big]} \qquad \text{(R-MATCH-SUCCEEDS)}$$

$$\frac{\Delta \vdash ilk(v) \not<\colon \tau}{\Delta \vdash E\big[(v\ \texttt{match}\ x\colon\tau \Rightarrow e\ \texttt{else}\ e')\big] \longrightarrow E[e']} \qquad \text{(R-MATCH-FAILS)}$$

Figure 5. Dynamic Semantics: Evaluation Rules

Program typing: $\boxed{\vdash p : \alpha}$

$$\frac{p = \overline{\delta}, e \qquad \overline{\delta} \vdash \overline{\delta\ \mathsf{ok}} \qquad \overline{\delta}; \bullet \vdash e : \alpha}{\vdash p : \alpha} \quad \text{(T-PROGRAM)}$$

Well-formed declarations: $\boxed{\Delta \vdash \delta\ \mathsf{ok}}$

$$\frac{\begin{array}{c}\Delta' = \Delta, \overline{\{\,\} <: P <: \{\overline{\xi}\}} \\ \Delta' \vdash \overline{\xi\ \mathsf{ok}} \qquad \Delta' \vdash \overline{A[\![\overline{\tau}]\!]\ \mathsf{ok}} \qquad \Delta' \vdash \overline{t\ \mathsf{ok}} \qquad \Delta' \vdash \overline{c\ \mathsf{ok}} \qquad \Delta' \vdash \overline{c <: T[\![\overline{P}]\!]} \\ distinct(T, \overline{A}) \qquad \overline{notOfTrait(\underline{T}, t)} \qquad \overline{notOfTrait(\underline{T}, c)} \qquad \Delta'; \mathtt{self}\colon T[\![\overline{P}]\!] \vdash \overline{\mu\ \mathsf{ok}}\end{array}}{\Delta \vdash \mathtt{trait}\ T[\![\overline{V\ P <: \{\overline{\xi}\}}]\!] <: \{\overline{A[\![\overline{\tau}]\!]}\} \lozenge \{\overline{t}\} \equiv\bigcup \{\overline{c}\}\ \overline{\mu}\ \mathsf{end}\ \mathsf{ok}} \quad \text{(D-TRAIT)}$$

$$\frac{\begin{array}{c}\Delta' = \Delta, \overline{\{\,\} <: P <: \{\overline{\xi}\}} \\ \Delta' \vdash \overline{\xi\ \mathsf{ok}} \qquad \Delta' \vdash \overline{\tau\ \mathsf{ok}} \qquad \Delta' \vdash \overline{t\ \mathsf{ok}} \qquad \Delta'; \mathtt{self}\colon T[\![\overline{P}]\!] \vdash \overline{\mu\ \mathsf{ok}}\end{array}}{\Delta \vdash \mathtt{object}\ O[\![\overline{P <: \{\overline{\xi}\}}]\!](\overline{x\colon\tau}) <: \{\overline{t}\}\ \overline{\mu}\ \mathsf{end}\ \mathsf{ok}} \quad \text{(D-OBJECT)}$$

$$\frac{\begin{array}{c}\Delta' = \Delta, \overline{\{\,\} <: P <: \{\overline{\xi}\}} \\ \Delta' \vdash \overline{\xi\ \mathsf{ok}} \qquad \Delta' \vdash \overline{\tau\ \mathsf{ok}} \qquad \Delta' \vdash \omega\ \mathsf{ok} \qquad \Delta'; \overline{x\colon\tau} \vdash e : \rho \qquad \Delta' \vdash \rho <: \omega\end{array}}{\Delta \vdash f[\![\overline{P <: \{\overline{\xi}\}}]\!](\overline{x\colon\tau})\colon \omega = e\ \mathsf{ok}} \quad \text{(D-FUNCTION)}$$

Well-formed methods: $\boxed{\Delta; \Gamma \vdash \mu\ \mathsf{ok}}$

$$\frac{\begin{array}{c}\Delta' = \Delta, \overline{\{\overline{\zeta}\} <: P <: \{\overline{\xi}\}} \\ \Delta' \vdash \overline{\zeta\ \mathsf{ok}} \qquad \Delta' \vdash \overline{\xi\ \mathsf{ok}} \qquad \Delta' \vdash \overline{\tau\ \mathsf{ok}} \qquad \Delta' \vdash \omega\ \mathsf{ok} \qquad \Delta'; \Gamma, \overline{x\colon\tau} \vdash e : \rho \qquad \Delta' \vdash \rho <: \omega\end{array}}{\Delta \vdash m[\![\overline{\{\overline{\zeta}\} <: P <: \{\overline{\xi}\}}]\!](\overline{x\colon\tau})\colon \omega = e\ \mathsf{ok}} \quad \text{(D-METHOD)}$$

Figure 9. Program typing and Well-formed Definitions

Subtyping (part 2 of 2): $\boxed{\Delta \vdash \kappa <: \kappa}$

$$\frac{\begin{array}{c}\mathtt{trait}\ T[\![\overline{V\ P <: \{\_\}}, \mathtt{covariant}\ Q <: \{\_\}, \overline{V'\ S <: \{\_\}}]\!]\ \_\ \mathsf{end} \in \{\Delta\} \\ \Delta \vdash \gamma <: \gamma' \qquad \#(\overline{\alpha}) = \#(\overline{P}) \qquad \#(\overline{\eta}) = \#(\overline{S}) \qquad \Delta \vdash T[\![\overline{\alpha}, \gamma, \overline{\eta}]\!]\ \mathsf{ok} \qquad \Delta \vdash T[\![\overline{\alpha}, \gamma', \overline{\eta}]\!]\ \mathsf{ok}\end{array}}{\Delta \vdash T[\![\overline{\alpha}, \gamma, \overline{\eta}]\!] <: T[\![\overline{\alpha}, \gamma', \overline{\eta}]\!]} \quad \text{(S-COVARIANT)}$$

$$\frac{\begin{array}{c}\mathtt{trait}\ T[\![\overline{V\ P <: \{\_\}}, \mathtt{contravariant}\ Q <: \{\_\}, \overline{V'\ S <: \{\_\}}]\!]\ \_\ \mathsf{end} \in \{\Delta\} \\ \Delta \vdash \gamma' <: \gamma \qquad \#(\overline{\alpha}) = \#(\overline{P}) \qquad \#(\overline{\eta}) = \#(\overline{S}) \qquad \Delta \vdash T[\![\overline{\alpha}, \gamma, \overline{\eta}]\!]\ \mathsf{ok} \qquad \Delta \vdash T[\![\overline{\alpha}, \gamma', \overline{\eta}]\!]\ \mathsf{ok}\end{array}}{\Delta \vdash T[\![\overline{\alpha}, \gamma, \overline{\eta}]\!] <: T[\![\overline{\alpha}, \gamma', \overline{\eta}]\!]} \quad \text{(S-CONTRAVARIANT)}$$

$$\frac{\mathtt{trait}\ T[\![\overline{V\ P <: \{\_\}}]\!] <: \{\overline{t}\}\ \_\ \mathsf{end} \in \{\Delta\} \qquad \Delta \vdash T[\![\overline{\alpha}]\!]\ \mathsf{ok}}{\Delta \vdash T[\![\overline{\alpha}]\!] <: [\overline{\alpha/P}]t} \quad \text{(S-TRAIT-EXTENDS)}$$

$$\frac{\mathtt{object}\ O[\![\overline{P <: \{\_\}}]\!](\_) <: \{\overline{t}\}\ \_\ \mathsf{end} \in \{\Delta\} \qquad \Delta \vdash O[\![\overline{\alpha}]\!]\ \mathsf{ok}}{\Delta \vdash O[\![\overline{\alpha}]\!] <: [\overline{\alpha/P}]t} \quad \text{(S-OBJECT-EXTENDS)}$$

Figure 12. Subtyping (part 2 of 2)

$$
\begin{array}{lll}
v & ::= O\llbracket\overline{\alpha}\rrbracket(\overline{v}) & \text{object instance} \\
& \mid\ (\overline{v}) & \text{tuple value} \\
& \mid\ ((\overline{x\colon\tau})\colon \tau \Rightarrow e) & \text{function value} \\
E & ::= \Box & \text{evaluation context} \\
& \mid\ (\overline{e}\,E\,\overline{e}) & \\
& \mid\ \pi_i(E) & \\
& \mid\ E\texttt{@}(\overline{e}) & \\
& \mid\ e\texttt{@}(\overline{e}\,E\,\overline{e}) & \\
& \mid\ E.z & \\
& \mid\ O\llbracket\overline{\tau}\rrbracket(\overline{e}\,E\,\overline{e}) & \\
& \mid\ f\llbracket\overline{\tau}\rrbracket(\overline{e}\,E\,\overline{e}) & \\
& \mid\ f(\overline{e}\,E\,\overline{e}) & \\
& \mid\ E.m\llbracket\overline{\tau}\rrbracket(\overline{e}) & \\
& \mid\ e.m\llbracket\overline{\tau}\rrbracket(\overline{e}\,E\,\overline{e}) & \\
& \mid\ E.m(\overline{e}) & \\
& \mid\ e.(\overline{e}\,E\,\overline{e}) & \\
& \mid\ (E \texttt{ match } x\colon\tau \Rightarrow e \texttt{ else } e) & \\
R & ::= \pi_i(v) & \text{redex} \\
& \mid\ (v) & \\
& \mid\ v\texttt{@}(\overline{v}) & \\
& \mid\ v.z & \\
& \mid\ f\llbracket\overline{\tau}\rrbracket(\overline{v}) & \\
& \mid\ f(\overline{v}) & \\
& \mid\ v.m\llbracket\overline{\tau}\rrbracket(\overline{v}) & \\
& \mid\ v.m(\overline{v}) & \\
& \mid\ (v \texttt{ match } x\colon\tau \Rightarrow e \texttt{ else } e) &
\end{array}
$$

Ilk of a value: $\boxed{\mathit{ilk}(v) = \alpha}$

$$\mathit{ilk}(O\llbracket\overline{\alpha}\rrbracket(\overline{v})) = O\llbracket\overline{\alpha}\rrbracket$$
$$\mathit{ilk}((\overline{v})) = (\overline{\mathit{ilk}(v)})$$
$$\mathit{ilk}(((\overline{x\colon\tau})\colon\tau' \Rightarrow e)) = ((\overline{\tau}) \to \tau')$$

Figure 4. Values, Evaluation Contexts, Redexes, and Ilks

Domain of environment: $\boxed{\mathit{dom}(\Gamma) = \{\overline{x}\}}$

$$\mathit{dom}(\overline{x\colon\_}) = \{\overline{x}\} \qquad \text{(E-DOMAIN)}$$

Lookup in variable-type environment: $\boxed{\mathit{lookup}(\Gamma, x) = \alpha}$

$$\frac{\Gamma = \_, x\colon\alpha, \overline{x'\colon\_} \qquad x \notin \{\overline{x'}\}}{\mathit{lookup}(\Gamma, x) = \alpha} \qquad \text{(E-LOOKUP)}$$

Figure 6. Functions on Variable-type Enviroments

Static types of expressions: $\boxed{\Delta; \Gamma \vdash e : \alpha}$

$$\frac{x \in \mathit{dom}(\Gamma)}{\Delta; \Gamma \vdash x : \mathit{lookup}(\Gamma, x)} \qquad \text{(T-VARIABLE)}$$

$$\frac{\texttt{self} \in \mathit{dom}(\Gamma)}{\Delta; \Gamma \vdash \texttt{self} : \mathit{lookup}(\Gamma, \texttt{self})} \qquad \text{(T-SELF)}$$

$$\frac{\Delta; \Gamma \vdash \overline{e : \alpha}}{\Delta; \Gamma \vdash (\overline{e}) : (\overline{\alpha})} \qquad \text{(T-TUPLE)}$$

$$\frac{\Delta; \Gamma \vdash e : (\overline{\alpha})}{\Delta; \Gamma \vdash \overline{\pi(\underline{e}) : \alpha}} \qquad \text{(T-PROJECT)}$$

$$\Delta; \Gamma \vdash ((\overline{x\colon\tau})\colon\omega \Rightarrow e) : ((\overline{\tau}) \to \omega) \qquad \text{(T-FUNC)}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash e : ((\overline{\alpha}) \to \rho) \\ \Delta; \Gamma \vdash (\overline{e'}) : (\overline{\chi}) \qquad \Delta \vdash \overline{\chi <: \alpha}\end{array}}{\Delta; \Gamma \vdash e\texttt{@}(\overline{e'}) : \rho} \qquad \text{(T-APPLY)}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash e : O\llbracket\overline{\alpha}\rrbracket \\ \texttt{object } O\llbracket\overline{P <: \{\_\}}\rrbracket(\overline{z\colon\tau})\ \_ \texttt{ end} \in \{\Delta\}\end{array}}{\Delta; \Gamma \vdash \underline{\underline{e}.z : \left[\overline{\alpha/P}\right]\tau}} \qquad \text{(T-FIELD)}$$

$$\frac{TBD}{\Delta; \Gamma \vdash O\llbracket\overline{\tau}\rrbracket(\overline{e}) :} \qquad \text{(T-OBJECT)}$$

$$\frac{TBD}{\Delta; \Gamma \vdash f\llbracket\overline{\tau}\rrbracket(\overline{e}) :} \qquad \text{(T-FUNC-SA)}$$

$$\frac{TBD}{\Delta; \Gamma \vdash f(\overline{e}) :} \qquad \text{(T-FUNC-NSA)}$$

$$\frac{TBD}{\Delta; \Gamma \vdash e.m\llbracket\overline{\tau}\rrbracket(\overline{e}) :} \qquad \text{(T-METHOD-SA)}$$

$$\frac{TBD}{\Delta; \Gamma \vdash e.m(\overline{e}) :} \qquad \text{(T-METHOD-NSA)}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash e : \alpha \\ \Delta; \Gamma, x\colon(\alpha \cap \tau) \vdash e' : \eta \qquad \Delta; \Gamma \vdash e'' : \chi\end{array}}{\Delta; \Gamma \vdash (e \texttt{ match } x\colon\tau \Rightarrow e' \texttt{ else } e'') : (\eta \cup \chi)} \qquad \text{(T-MATCH)}$$

Figure 7. Static Types of Expressions

Well-formed types: $\boxed{\Delta \vdash \kappa \ \mathsf{ok}}$

$$\frac{\{\_\} <: P <: \{\_\} \in \{\Delta\}}{\Delta \vdash P \ \mathsf{ok}} \quad \text{(W-PARAM)}$$

$$\frac{\begin{array}{c}\texttt{trait}\ T[\![\overline{V\ P <: \{\overline{\xi}\}}]\!]\ \_\ \texttt{end} \in \{\Delta\} \\ \#(\overline{\alpha}) = \#(\overline{P}) \qquad \Delta \vdash \overline{\alpha\ \mathsf{ok}} \\ \Delta \vdash \overline{\alpha <: \left[\overline{\alpha/P}\right]\xi}\end{array}}{\Delta \vdash T[\![\overline{\alpha}]\!]\ \mathsf{ok}} \quad \text{(W-TRAIT)}$$

$$\frac{\begin{array}{c}\texttt{object}\ O[\![\overline{P <: \{\overline{\xi}\}}]\!]\ \_\ \texttt{end} \in \{\Delta\} \\ \#(\overline{\alpha}) = \#(\overline{P}) \qquad \Delta \vdash \overline{\alpha\ \mathsf{ok}} \\ \Delta \vdash \overline{\alpha <: \left[\overline{\alpha/P}\right]\xi}\end{array}}{\Delta \vdash O[\![\overline{\alpha}]\!]\ \mathsf{ok}} \quad \text{(W-OBJECT)}$$

$$\frac{\Delta \vdash \overline{\alpha\ \mathsf{ok}}}{\Delta \vdash (\overline{\alpha})\ \mathsf{ok}} \quad \text{(W-TUPLE)}$$

$$\frac{\Delta \vdash \alpha\ \mathsf{ok} \qquad \Delta \vdash \rho\ \mathsf{ok}}{\Delta \vdash (\alpha \rightarrow \rho)\ \mathsf{ok}} \quad \text{(W-ARROW)}$$

$$\Delta \vdash \mathrm{Any}\ \mathsf{ok} \quad \text{(W-ANY-TYPE)}$$

$$\Delta \vdash \mathrm{Object}\ \mathsf{ok} \quad \text{(W-OBJECT-TYPE)}$$

$$\Delta \vdash \mathrm{Bottom}\ \mathsf{ok} \quad \text{(W-BOTTOM-TYPE)}$$

$$\frac{\Delta \vdash \alpha\ \mathsf{ok} \qquad \Delta \vdash \gamma\ \mathsf{ok}}{\Delta \vdash (\alpha \cup \gamma)\ \mathsf{ok}} \quad \text{(W-UNION)}$$

$$\frac{\Delta \vdash \alpha\ \mathsf{ok} \qquad \Delta \vdash \gamma\ \mathsf{ok}}{\Delta \vdash (\alpha \cap \gamma)\ \mathsf{ok}} \quad \text{(W-INTERSECTION)}$$

$$\frac{\begin{array}{c}\Delta \vdash \overline{\chi\ \mathsf{ok}} \qquad \Delta \vdash \overline{\eta\ \mathsf{ok}} \\ \Delta, \overline{\{\overline{\chi}\} <: P <: \{\overline{\eta}\}} \vdash \alpha\ \mathsf{ok}\end{array}}{\Delta \vdash \exists[\![\overline{\{\overline{\chi}\} <: P <: \{\overline{\eta}\}}]\!]\alpha\ \mathsf{ok}} \quad \text{(W-EXISTS)}$$

$$\frac{\begin{array}{c}\Delta \vdash \overline{\chi\ \mathsf{ok}} \qquad \Delta \vdash \overline{\eta\ \mathsf{ok}} \\ \Delta, \overline{\{\overline{\chi}\} <: P <: \{\overline{\eta}\}} \vdash \alpha\ \mathsf{ok}\end{array}}{\Delta \vdash \forall[\![\overline{\{\overline{\chi}\} <: P <: \{\overline{\eta}\}}]\!]\alpha\ \mathsf{ok}} \quad \text{(W-FORALL)}$$

Figure 8. Well-formed Types

Not constructed from a given trait: $\boxed{\mathit{notOfTrait}(T, c)}$

$$\mathit{notOfTrait}(T, O[\![\_]\!]) \quad \text{(NOTOFTRAIT-OBJECT)}$$

$$\frac{\mathit{distinct}(T, T')}{\mathit{notOfTrait}(T, T'[\![\_]\!])} \quad \text{(NOTOFTRAIT-TRAIT)}$$

Distinct trait names: $\boxed{\mathit{distinct}(\overline{T})}$

$$\frac{\left|\{\overline{T}\}\right| = \#(\overline{T})}{\mathit{distinct}(\overline{T})} \quad \text{(DISTINCT-TRAIT)}$$

Figure 10. Miscellaneous Judgments

Subtyping (part 1 of 2): $\boxed{\Delta \vdash \kappa <: \kappa}$

$$\Delta \vdash \kappa <: \kappa \quad \text{(S-REFLEXIVE)}$$

$$\frac{\Delta \vdash \kappa <: \kappa' \qquad \Delta \vdash \kappa' <: \kappa''}{\Delta \vdash \kappa <: \kappa''} \quad \text{(S-TRANSITIVE)}$$

$$\frac{\Delta \vdash \kappa \equiv \kappa'}{\Delta \vdash \kappa <: \kappa'} \quad \text{(S-EQUIVALENT)}$$

$$\frac{\{\_\} <: P <: \{\overline{\alpha}\} \in \{\Delta\}}{\Delta \vdash \overline{P <: \alpha}} \quad \text{(S-PARAMETER-SUB)}$$

$$\frac{\{\overline{\alpha}\} <: P <: \{\_\} \in \{\Delta\}}{\Delta \vdash \overline{\alpha <: P}} \quad \text{(S-PARAMETER-SUPER)}$$

$$\Delta \vdash \kappa <: \mathrm{Any} \quad \text{(S-ANY)}$$

$$\Delta \vdash \mathrm{Bottom} <: \kappa \quad \text{(S-BOTTOM)}$$

$$\Delta \vdash c <: \mathrm{Object} \quad \text{(S-OBJECT)}$$

$$\frac{\#(\overline{\alpha}) = \#(\overline{\eta}) \qquad \Delta \vdash \overline{\alpha <: \eta}}{\Delta \vdash (\overline{\alpha}) <: (\overline{\eta})} \quad \text{(S-TUPLE)}$$

$$\frac{\Delta \vdash \alpha' <: \alpha \qquad \Delta \vdash \rho <: \rho'}{\Delta \vdash (\alpha \to \rho) <: (\alpha' \to \rho')} \quad \text{(S-ARROW)}$$

$$\Delta \vdash \alpha <: (\alpha \cup \eta) \quad \text{(S-UNION-SUPER)}$$

$$\Delta \vdash (\alpha \cap \eta) <: \alpha \quad \text{(S-INTERSECTION-SUB)}$$

$$\frac{\Delta \vdash (\alpha \cup \eta) <: \chi}{\Delta \vdash \alpha <: \chi} \quad \text{(S-UNION-SUB)}$$

$$\frac{\Delta \vdash \chi <: (\alpha \cap \eta)}{\Delta \vdash \chi <: \alpha} \quad \text{(S-INTERSECTION-SUPER)}$$

Figure 11. Subtyping (part 1 of 2)

Type equivalence: $\boxed{\Delta \vdash \kappa \equiv \kappa}$

$$\frac{\Delta \vdash \kappa \equiv \kappa'}{\Delta \vdash \kappa' \equiv \kappa} \quad \text{(E-SYMMETRIC)}$$

$$\Delta \vdash (\kappa) \equiv \kappa \quad \text{(E-SINGLETON-TUPLE)}$$

$$\frac{\kappa' \in \{\overline{\kappa}\} \qquad \Delta \vdash \kappa' <: \mathrm{Bottom}}{\Delta \vdash (\overline{\kappa}) \equiv \mathrm{Bottom}} \quad \text{(E-TUPLE-BOTTOM)}$$

$$\Delta \vdash (\alpha \cup \eta) \equiv (\eta \cup \alpha) \quad \text{(U-COMMUTATIVE)}$$

$$\Delta \vdash (\alpha \cup (\eta \cup \chi)) \equiv ((\alpha \cup \eta) \cup \chi) \quad \text{(U-ASSOCIATIVE)}$$

$$\frac{\Delta \vdash \alpha <: \eta}{\Delta \vdash (\alpha \cup \eta) \equiv \eta} \quad \text{(U-ABSORPTION)}$$

$$\Delta \vdash (\alpha \cap \eta) \equiv (\eta \cap \alpha) \quad \text{(I-COMMUTATIVE)}$$

$$\Delta \vdash (\alpha \cap (\eta \cap \chi)) \equiv ((\alpha \cap \eta) \cap \chi) \quad \text{(I-ASSOCIATIVE)}$$

$$\frac{\Delta \vdash \alpha <: \eta}{\Delta \vdash (\alpha \cap \eta) \equiv \alpha} \quad \text{(I-ABSORPTION)}$$

Figure 13. Type equivalence

Type exclusion: $\boxed{\Delta \vdash \alpha \equiv \alpha}$

Figure 14. Type Exclusion

4. Wellformedness

5. Examples

6. Related Work

7. Conclusion and Discussion

Acknowledgments

References

[1] Eric Allen, David Chase, Joe Hallett, Victor Luchangco, Jan-Willem Maessen, Sukyoung Ryu, Guy L. Steele Jr., and Sam Tobin-Hochstadt. The Fortress Language Specification Version 1.0, March 2008.

Exclusion: $\boxed{\Delta \vdash \alpha \Diamond \alpha \Leftarrow \mathcal{C}}$

Logical rules

$$\Delta \vdash \mathrm{Bottom} \Diamond \alpha \Leftarrow \text{true}$$

$$\frac{\Delta \vdash \alpha <: \mathrm{Bottom} \Leftarrow \mathcal{C}}{\Delta \vdash \mathrm{Any} \Diamond \alpha \Leftarrow \mathcal{C}}$$

$$\frac{\#(\overline{\alpha}) \neq 1}{\Delta \vdash (\overline{\alpha}) \Diamond \mathrm{Object} \Leftarrow true}$$

$$\Delta \vdash (\alpha \to \rho) \Diamond \mathrm{Object} \Leftarrow true$$

$$\frac{\begin{array}{c}\Delta \vdash \alpha \Diamond \chi \Leftarrow \mathcal{C} \qquad \Delta \vdash \eta \Diamond \chi \Leftarrow \mathcal{C}' \\ \Delta \vdash (\alpha \cap \eta) <: \mathrm{Bottom} \Leftarrow \mathcal{C}''\end{array}}{\Delta \vdash (\alpha \cap \eta) \Diamond \chi \Leftarrow \mathcal{C} \vee \mathcal{C}' \vee \mathcal{C}''}$$

$$\frac{\Delta \vdash \alpha \Diamond \chi \Leftarrow \mathcal{C} \qquad \Delta \vdash \eta \Diamond \chi \Leftarrow \mathcal{C}'}{\Delta \vdash (\alpha \cup \eta) \Diamond \chi \Leftarrow \mathcal{C} \wedge \mathcal{C}'}$$

Inference Variables

$$\frac{I \notin parameters(\Delta)}{\Delta \vdash I \Diamond I \Leftarrow \text{false}}$$

$$\frac{I \notin parameters(\Delta)}{\Delta \vdash I \Diamond \alpha \Leftarrow I \Diamond \alpha}$$

Bound Variables

$$\frac{\begin{array}{c}\{\_\} <: P <: \{\overline{\xi}\} \in \Delta \\ \Delta \vdash \overline{\xi \Diamond \alpha \Leftarrow \mathcal{C}}\end{array}}{\Delta \vdash P \Diamond \alpha \Leftarrow \bigvee \{\overline{\mathcal{C}}\}}$$

Structural rules

$$\frac{\#(\overline{\alpha}) = \#(\overline{\eta}) \qquad \Delta \vdash \overline{\alpha \Diamond T \Leftarrow \mathcal{C}}}{\Delta \vdash (\overline{\alpha}) \Diamond (\overline{\eta}) \Leftarrow \bigvee\{\mathcal{C}\}}$$

$$\frac{\#(\overline{\alpha}) \neq \#(\overline{\eta})}{\Delta \vdash (\overline{\alpha}) \Diamond (\overline{\eta}) \Leftarrow \text{true}}$$

$$\frac{\#(\overline{\eta}) \neq 1}{\Delta \vdash c \Diamond (\overline{\eta}) \Leftarrow \text{true}}$$

$$\frac{\#(\overline{\eta}) \neq 1}{\Delta \vdash (\alpha \to \rho) \Diamond (\overline{\eta}) \Leftarrow \text{true}}$$

$$\Delta \vdash (\alpha \to \rho) \Diamond (\alpha' \to \rho') \Leftarrow \text{false}$$

$$\Delta \vdash c \Diamond (\alpha \to \rho) \Leftarrow \text{true}$$

Constructed types

$$\frac{\begin{array}{c}\Delta \vdash c \Diamond_{\mathrm{x}} c' \Leftarrow \mathcal{C}_x \qquad \Delta \vdash c \Diamond_{\mathrm{c}} c' \Leftarrow \mathcal{C}_c \\ \Delta \vdash c \Diamond_{\mathrm{o}} c' \Leftarrow \mathcal{C}_o \qquad \Delta \vdash c \Diamond_{\mathrm{m}} c' \Leftarrow \mathcal{C}_m\end{array}}{\Delta \vdash c \Diamond c' \Leftarrow \mathcal{C}_x \vee \mathcal{C}_c \vee \mathcal{C}_o \vee \mathcal{C}_m}$$

$$\frac{\Delta \vdash c \rhd_{\mathrm{x}} c' \Leftarrow \mathcal{C} \qquad \Delta \vdash c' \rhd_{\mathrm{x}} c \Leftarrow \mathcal{C}'}{\Delta \vdash c \Diamond_{\mathrm{x}} c' \Leftarrow \mathcal{C} \vee \mathcal{C}'}$$

$$\frac{\Delta \vdash c \rhd_{\mathrm{c}} c' \Leftarrow \mathcal{C} \qquad \Delta \vdash c' \rhd_{\mathrm{c}} c \Leftarrow \mathcal{C}'}{\Delta \vdash c \Diamond_{\mathrm{c}} c' \Leftarrow \mathcal{C} \vee \mathcal{C}'}$$

$$\frac{\Delta \vdash c \rhd_{\mathrm{o}} c' \Leftarrow \mathcal{C} \qquad \Delta \vdash c' \rhd_{\mathrm{o}} c \Leftarrow \mathcal{C}'}{\Delta \vdash c \Diamond_{\mathrm{o}} c' \Leftarrow \mathcal{C} \vee \mathcal{C}'}$$

$$\Delta \vdash T[\![\overline{\alpha}]\!] \rhd_{\mathrm{x}} T[\![\overline{\eta}]\!] \Leftarrow \text{false}$$

$$\Delta \vdash T[\![\overline{\alpha}]\!] \rhd_{\mathrm{c}} T[\![\overline{\eta}]\!] \Leftarrow \text{false}$$

$$\Delta \vdash T[\![\overline{\alpha}]\!] \rhd_{\mathrm{o}} T[\![\overline{\eta}]\!] \Leftarrow \text{false}$$

$$\frac{\begin{array}{c}distinct(T, T') \qquad \{\overline{\chi}\} = \mathit{ancestors}(T[\![\overline{\alpha}]\!]) \\ \{\overline{\omega}\} = \bigcup\{\overline{\chi.\mathit{excludes}}\} \qquad \Delta \vdash \overline{\underline{T'[\![\overline{\eta}]\!]} <: \omega \Leftarrow \mathcal{C}}\end{array}}{\Delta \vdash T[\![\overline{\alpha}]\!] \rhd_{\mathrm{x}} T'[\![\overline{\eta}]\!] \Leftarrow \bigvee \{\overline{\mathcal{C}}\}}$$

$$\frac{\begin{array}{c}distinct(T, T') \\ \{\overline{\omega}\} = \bigcup\{\overline{T[\![\overline{\alpha}]\!].\mathit{comprises}}\} \qquad \Delta \vdash \overline{\underline{T'[\![\overline{\eta}]\!]} \Diamond \omega \Leftarrow \mathcal{C}}\end{array}}{\Delta \vdash T[\![\overline{\alpha}]\!] \rhd_{\mathrm{c}} T'[\![\overline{\eta}]\!] \Leftarrow \bigwedge \{\overline{\mathcal{C}}\}}$$

$$\frac{\Delta \vdash O[\![\overline{\alpha}]\!] \not<: c \Leftarrow \mathcal{C}}{\Delta \vdash O[\![\overline{\alpha}]\!] \rhd_{\mathrm{o}} c \Leftarrow \mathcal{C}}$$

$$\Delta \vdash T[\![\overline{\alpha}]\!] \rhd_{\mathrm{c}} T'[\![\overline{\eta}]\!] \Leftarrow \text{false}$$

$$\frac{\begin{array}{c}\{\overline{\chi}\} = \mathit{ancestors}(T[\![\overline{\alpha}]\!]) \qquad \{\overline{\omega}\} = \mathit{ancestors}(T'[\![\overline{\eta}]\!]) \\ \Delta \vdash \overline{\overline{\chi \diamondcdot_{\mathrm{m}} \underline{\omega} \Leftarrow \mathcal{C}}}\end{array}}{\Delta \vdash T[\![\overline{\alpha}]\!] \Diamond_{\mathrm{m}} T'[\![\overline{\eta}]\!] \Leftarrow \bigvee \left\{\overline{\overline{\mathcal{C}}}\right\}}$$

$$\frac{\Delta \vdash \overline{\alpha \not\equiv \eta \Leftarrow \mathcal{C}}}{\Delta \vdash T[\![\overline{\alpha}]\!] \diamondcdot_{\mathrm{m}} T[\![\overline{\eta}]\!] \Leftarrow \bigvee \{\overline{\mathcal{C}}\}}$$

$$\frac{distinct(T, T')}{\Delta \vdash T[\![\overline{\alpha}]\!] \diamondcdot_{\mathrm{m}} T'[\![\overline{\eta}]\!] \Leftarrow \text{false}}$$

Figure 15. Algorithm for generating exclusion constraints. Each rule is symmetric; apply the first one that matches.