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

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 $\overline{[\tau/P]\tau'}$ means $[\tau/P]\tau'_1, [\tau/P]\tau'_2, \ldots, [\tau/P]\tau'_n$, so one possible concrete expansion is $[\tau/P]\tau'_1, [\tau/P]\tau'_2, [\tau/P]\tau'_3, [\tau/P]\tau'_4, [\tau/P]\tau'_5$. In addition, if an overbar construction expands to zero copies and there is a comma adjacent to the construction (or perhaps two, one on each side), then exactly one adjacent comma is deleted.

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:

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

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

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
I ranges over inference variables names I
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 Used in This Paper

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, in the following order:

- Every occurrence of a BNF nonterminal and 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 have the same base letter, and that base letter ranges 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.

p	$::= \overline{\delta}, e$	program (declarations plus expression)
δ	$::= \texttt{trait}\ T[\![\overline{V\ \beta}]\!] <: \{\overline{t}\} \Diamond \{\overline{t}\} \equiv \bigcup \{\overline{c}\}\ \overline{\mu}\ \texttt{end}$	trait declaration
	$\mid \texttt{trait}\ T[\![\overline{V\ \beta}]\!] <: \{\overline{t}\} \Diamond \{\overline{t}\}\ \overline{\mu}\ \texttt{end}$	trait declaration
	$\mid \texttt{object}\ O[\![\overline{\beta}]\!](\overline{x{:}\,\tau}) <: \{\overline{t}\}\ \overline{\mu}\ \texttt{end}$	object declaration
	$\mid f[\![\overline{\beta}]\!](\overline{x{:}\,\tau}){:}\,\tau = e$	function declaration
V	$::= \texttt{covariant} \mid \texttt{contravariant} \mid \texttt{invariant}$	variance
μ	$::= m[\![\overline{\varphi}]\!](\overline{x{:}\,\tau}){:}\,\tau = e$	dotted method declaration
β	$::= P <: \{\overline{\tau}\}$	simple type parameter binding
φ	$::= \{\overline{\tau}\} <: P <: \{\overline{\tau}\}$	full type parameter binding
e	$::= x$	variable reference
	$\mid z$	field 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
	$\mid e.z$	dotted field reference
	$\mid O[\![\overline{\tau}]\!](\overline{e})$	object creation
	$\mid f(\overline{e})$	function invocation
	$\mid e.m(\overline{e})$	method dotted invocation
	$\mid (e\ \texttt{match}\ x{:}\,\tau \Rightarrow e\ \texttt{else}\ e)$	match expression

τ	$::= P$	type parameter reference
	$\mid c$	constructed type
	$\mid (\overline{\tau})$	tuple type
	$\mid (\tau \to \tau)$	arrow type
	$\mid \mathrm{Any}$	special Any type
	$\mid \mathrm{Object}$	special Object type
c	$::= O[\![\overline{\tau}]\!]$	object type
	$\mid t$	trait type
t	$::= T[\![\overline{\tau}]\!]$	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
Id	$::= P$	names
	$\mid T$	
	$\mid O$	
	$\mid x$	
	$\mid z$	
	$\mid f$	
	$\mid m$	

Figure 2. Grammar for Welterweight Fortress

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

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

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 monogram 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

The grammar for Welterweight Fortress is given in Figure 2. A program consists of declarations and an expression to be evaluated i the contex of those declarations. Each declaration defines a trait, an object, or a top-level function.

A trait is similar to a Java interface, but can contain method declarations. It also has an *extends clause* $<: \{\overline{t}\}$, an *excludes clause* $\Diamond\ \{\overline{t}\}$, and optionally a *comprises clause* $\equiv\bigcup\ \{\overline{c}\}$. The extends clause indicates what other trait instances are supertypes of the trait; the excludes clause indicates what other trait instances cannot have values in common with this trait. The comprises clause, if present, indicates that no value can belong to the trait unless it also belongs to one of the comprised types.

An object is similar to a Java class. Rather than having a separately declared constructor method, it has a parameter list containing names of fields and their types; an object creation expression provides a set of arguments that are simply used to initialize the declared fields, with no further action

α	$::= P$	type parameter name
	$\mid T[\![\overline{\alpha}]\!]$	trait type
	$\mid O[\![\overline{\alpha}]\!]$	object type
	$\mid (\overline{\alpha})$	tuple type
	$\mid (\alpha \to \alpha)$	arrow type
	$\mid \mathrm{Any}$	special Any type
	$\mid \mathrm{Object}$	special Object type
	$\mid \mathrm{Bottom}$	special Bottom type
	$\mid (\alpha \cup \alpha)$	union type
	$\mid (\alpha \cap \alpha)$	intersection type
κ	$::= \alpha$	lattice type
	$\mid \Phi$	existentially quantified type
	$\mid \Psi$	universally quantified type
Φ	$::= \exists[\![\overline{\lambda}]\!]\alpha$	existentially quantified type
Ψ	$::= \forall[\![\overline{\lambda}]\!]\alpha$	universally quantified type
λ	$::= \{\overline{\alpha}\} <: P <: \{\overline{\alpha}\}$	lattice type parameter binding
ψ	$::= \delta$	program declaration
	$\mid \lambda$	lattice type parameter binding
I	$::=$ identifier	type inference variable name
var	$::= x$	variable name
	$\mid z$	field name
	$\mid$ `self`	self keyword
Δ	$::= \{\overline{\psi}\}$	type-declaration environment
Γ	$::= \overline{var : \alpha}$	variable-type environment

Figure 3. Symbols Not Used in the Concrete Syntax

taken. An object declaration also has an extends clause, indicating what trait instances are supertypes of the object type.

A top-level function has a parameter list containing names of parameters and their types, and also a return type and a body expression. Top-level functions may be overloaded; that is, more than one function declaration may have the same name f.

Method declarations appearing in traits or objects are similar in form to function declarations; however, the keyword `self` may be used within its body expression to refer to a value (the *target object*) for which the method was invoked.

Traits, objects, functions, and methods all have a (possibly empty) list of *static type parameters*. Associated with each static type parameter is (possibly empty) set of *upper bounds*; any type used to instantiate the type parameter must be a subtype of each of the declared upper bounds. Each static type parameter of a method also has a (possibly empty) set of *lower bounds*; any type used to instantiate the type parameter must be a supertype of each of the declared lower bounds. Each static type parameter of a trait has an associated *variance*, which indicates ways in which different instances of the same trait may extend or exclude each other.

Most of the forms of expression are fairly conventional. The function creation expression might be called a 'typed

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 4. Well-formed Types

Well-formed expressions and static types: $\boxed{\Delta;\Gamma \vdash e : \alpha}$

$$\frac{\Gamma = \_, var{:}\,\alpha, \overline{x{:}\,\_} \qquad var \notin \{\overline{x}\}}{\Delta;\Gamma \vdash var : \alpha} \quad \text{(T-VARIABLE)}$$

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

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

$$\frac{\Delta \vdash \overline{\tau\ \mathsf{ok}} \qquad \Delta \vdash \omega\ \mathsf{ok} \qquad \Delta;\Gamma, \overline{x{:}\,\tau} \vdash e : \omega}{\Delta;\Gamma \vdash ((\overline{x{:}\,\tau}){:}\,\omega \Rightarrow e) : ((\overline{\tau}) \to \omega)} \quad \text{(T-FUNC)}$$

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

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

$$\frac{\Delta \vdash \overline{\tau\ \mathsf{ok}} \qquad TBD}{\Delta;\Gamma \vdash O[\![\overline{\tau}]\!](\overline{e}) :} \quad \text{(T-OBJECT)}$$

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

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

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

Figure 5. Static Types of Expressions

lambda expression" in other languages. We use the symbol `@` to distinguish application of function-typed values from invocation of (possibly overloaded) top-level functions. Function invocations and method invocations each come in two forms, depending on whether static type arguments are provided explicitly or are to be inferred. The *match expression* is a kind of compromise between a conventional of `if` e_1 `then` e_2 `else` e_3 by declaring objects $\mathrm{True}[\![\ ]\!]$ and $\mathrm{False}[\![\ ]\!]$, using them as values of predicates, and then writing e_1 `match` $x'{:}\,\mathrm{True}[\![\ ]\!] \Rightarrow e_2$ `else` e_3.

The type Any is a supertype of all types. The type Object is a supertype of every constructed type, that is, every trait type and every object type.

4. Wellformedness and Static Typing

Figure 3 gives a grammar for symbols that do not appear in the concrete grammar of Figure 2 but are used in later figures.

Figure 4 gives rules for deciding whether a type is well-formed, given a set of declarations Δ.

Figure 5 gives rules for determining the static type of an expression, given a set of declarations Δ and a variable-type environment Γ.

Figure 6 gives rules for deciding whether declarations are well-formed and well-typed, given the overall set of declarations Δ, and for determining the type of a program.

Figure 7 gives rules for miscellaneous subsidary judgments.

The rules for typing and well-formedness depend on judgments about subtyping and type exclusion, which are addressed in Section 6.

5. Dynamic Evaluation Semantics

Figure 8 gives conventional grammars for redexes (a subset of expressions that are subject to evaluation reduction), values (expressions that contain no redexes), and evaluation contexts. It also defines how to compute the ilk of a value. The notation $\langle(\alpha \to \omega)\rangle$ is the type of functions that produce a value of type ω when given an argument of type α *and* furthermore go wrong ("get stuck") when given an argument that is not f type α.

Figure 9 gives rules for reducing an expression to a value. We use the conventional notation $[\overline{X/Id}]X'$ to indicate a copy of X' modified by simultaneous substitution of a copy of X_1 for each occurrence of Id_1, a copy of X_2 for each occurrence of Id_2, and so on. We use the conventional notation $E[e]$ to mean an evaluation context with its "hole" replaced by the expression e (that is, $[e/\Box]E$).

In Welterweight Fortress there are three kinds of values: objects, functions, and tuples; every object belongs to at least one constructed type, every function belongs to at least one arrow type, and every tuple belongs to at least one tuple type. No value belongs to Bottom.

Well-formed program and its static type: $\boxed{\vdash p : \alpha}$

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

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

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

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

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

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

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

Figure 6. Program typing and Well-formed Definitions

Although a value may belong to more than one type, every value v belongs to a unique type $\mathit{ilk}(v)$ (the *ilk* of the value). For every type α, if v belongs to α then $\mathit{ilk}(v)$ is a subtype of α. (This notion of *ilk* corresponds to what is sometimes called the "class" or "run-time type" of the value.[1])

The implementation significance of ilks is that it is possible to select the dynamically most specific applicable function from an overload set using only the ilks of the argument values; no other information about the arguments is needed.

If the type system is sound, then if an expression is determined by the type system to have type α, then every value computed by the expression at run time will belong to type α moreover, whenever a function whose ilk is $\langle(\alpha \to \omega)\rangle$ is applied to an argument value, then the argument value must belong to type α, and whenever a function of method is invoked, the argument values will belong to the types of the corresponding parameters.

[1] We prefer the term "ilk" to "run-time type" because the notion—and usefulness—of the most specific type to which a value belongs is not confined to run time.

Not constructed from T: $\boxed{notOfTrait(T, c)}$

$$notOfTrait(T, O[\![\_]\!]) \quad \text{(NotOfTrait-Object)}$$

$$\frac{distinct(T, T')}{notOfTrait(T, T'[\![\_]\!])} \quad \text{(NotOfTrait-Trait)}$$

Distinct names: $\boxed{distinct(\overline{Id})}$

$$\frac{\left|\{\overline{Id}\}\right| = \#(\overline{Id})}{distinct(\overline{Id})} \quad \text{(Distinct-Identifiers)}$$

Distinct declarations: $\boxed{distinct(\overline{\delta})}$

$$\frac{\left|\{\overline{\delta}\}\right| = \#(\overline{\delta})}{distinct(\overline{\delta})} \quad \text{(Distinct-Declarations)}$$

Parameters in type environment: $\boxed{parameters(\Delta)}$

$$\frac{\Delta = \{\overline{\delta}\}}{parameters(\Delta) = \bigcap \{\overline{oneParam(\delta)}\}} \quad \text{(Parameters)}$$

$$\begin{array}{l} oneParam(\{\_\} <: P <: \{\_\}) = \{P\} \\ oneParam(\texttt{trait}\ \_\ \texttt{end}) = \{\,\} \\ oneParam(\texttt{object}\ \_\ \texttt{end}) = \{\,\} \\ oneParam(f[\![\_]\!](\_)\texttt{:}\_ = \_) = \{\,\} \end{array}$$

Trait/object names in type environment: $\boxed{typeNames(\Delta)}$

$$\frac{\Delta = \{\overline{\delta}\}}{typeNames(\Delta) = \bigcap \{\overline{oneName(\delta)}\}} \quad \text{(Names)}$$

$$\begin{array}{l} oneParam(\{\_\} <: P <: \{\_\}) = \{\,\} \\ oneParam(\texttt{trait}\ T[\![\_]\!]\ \_\ \texttt{end}) = \{T\} \\ oneParam(\texttt{object}\ O[\![\_]\!]\ \_\ \texttt{end}) = \{O\} \\ oneParam(f[\![\_]\!](\_)\texttt{:}\_ = \_) = \{\,\} \end{array}$$

Immediately extended traits: $\boxed{parents(\Delta, c)}$

$$\frac{\texttt{trait}\ T[\![\overline{V\ P <: \{\tau\}}]\!] <: \{\overline{t}\}\ \_\ \texttt{end} \in \Delta}{parents(\Delta, T[\![\overline{\alpha}]\!]) = \{\overline{[\overline{\alpha}/\overline{\tau}]t}\}}$$

$$\frac{\texttt{trait}\ O[\![\overline{P <: \{\tau\}}]\!](\_) <: \{\overline{t}\}\ \_\ \texttt{end} \in \Delta}{parents(\Delta, O[\![\overline{\alpha}]\!]) = \{\overline{[\overline{\alpha}/\overline{\tau}]t}\}}$$

All extended traits: $\boxed{ancestors(\Delta, c)}$

$$ancestors(\Delta, c) = \{c\} \cup \left(\bigcup_{\alpha \in parents(\Delta, c)} ancestors(\Delta, \alpha) \right)$$

Excluded traits: $\boxed{excluded(\Delta, c)}$

$$\frac{\texttt{trait}\ T[\![\overline{V\ P <: \{\tau\}}]\!]\_\ \Diamond\ \{\overline{t}\}\ \_\ \texttt{end} \in \Delta}{excluded(\Delta, T[\![\overline{\alpha}]\!]) = \{\overline{[\overline{\alpha}/\overline{\tau}]t}\}}$$

Comprised types: $\boxed{comprised(\Delta, c)}$

$$\frac{\texttt{trait}\ T[\![\overline{V\ P <: \{\tau\}}]\!]\_\ \Diamond\ \{\_\}\ \texttt{end} \in \Delta}{comprised(\Delta, T[\![\overline{\alpha}]\!]) = \{\text{Any}\}}$$

$$\frac{\begin{array}{c}\texttt{trait}\ T[\![\overline{V\ P <: \{\tau\}}]\!]\_\ \Diamond\ \{\_\}\ \equiv \bigcup \{\overline{c}\}\ \texttt{end} \in \Delta \\ \eta = oneComprised(\underline{\Delta}, (\overline{\alpha}), (\overline{P}), c)\end{array}}{comprised(\Delta, O[\![\overline{\alpha}]\!]) = \{\overline{[\overline{\alpha}/\overline{\tau}]c}\}}$$

Figure 7. Miscellaneous Judgments

$$
\begin{array}{lll}
v ::= O[\![\overline{\alpha}]\!](\overline{v}) & & \text{object instance} \\
\quad \mid (\overline{v}) & & \text{tuple value} \\
\quad \mid ((\overline{x{:}\,\tau}){:}\,\tau \Rightarrow e) & & \text{function value} \\
E ::= \square & & \text{evaluation context} \\
\quad \mid (\overline{e}\, E\, \overline{e}) & & \\
\quad \mid \pi_i(E) & & \\
\quad \mid E\texttt{@}(\overline{e}) & & \\
\quad \mid e\texttt{@}(\overline{e}\, E\, \overline{e}) & & \\
\quad \mid E.z & & \\
\quad \mid O[\![\overline{\tau}]\!](\overline{e}\, E\, \overline{e}) & & \\
\quad \mid f[\![\overline{\tau}]\!](\overline{e}\, E\, \overline{e}) & & \\
\quad \mid f(\overline{e}\, E\, \overline{e}) & & \\
\quad \mid E.m[\![\overline{\tau}]\!](\overline{e}) & & \\
\quad \mid e.m[\![\overline{\tau}]\!](\overline{e}\, E\, \overline{e}) & & \\
\quad \mid E.m(\overline{e}) & & \\
\quad \mid e.(\overline{e}\, E\, \overline{e}) & & \\
\quad \mid (E \texttt{ match } x{:}\,\tau \Rightarrow e \texttt{ else } e) & &
\end{array}
$$

$$
\begin{array}{lll}
R ::= \pi_i(v) & & \text{redex} \\
\quad \mid (v) & & \\
\quad \mid v\texttt{@}(\overline{v}) & & \\
\quad \mid v.z & & \\
\quad \mid f[\![\overline{\tau}]\!](\overline{v}) & & \\
\quad \mid f(\overline{v}) & & \\
\quad \mid v.m[\![\overline{\tau}]\!](\overline{v}) & & \\
\quad \mid v.m(\overline{v}) & & \\
\quad \mid (v \texttt{ match } x{:}\,\tau \Rightarrow e \texttt{ else } e) & &
\end{array}
$$

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

$$
\begin{array}{l}
ilk(O[\![\overline{\alpha}]\!](\overline{v})) = O[\![\overline{\alpha}]\!] \\
ilk((\overline{v})) = (\overline{ilk(v)}) \\
ilk(((\overline{x{:}\,\tau}){:}\,\tau' \Rightarrow e)) = \langle((\overline{\tau}) \to \tau')\rangle
\end{array}
$$

Figure 8. Values, Evaluation Contexts, Redexes, and Ilks

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

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

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

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

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

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

$$\frac{\begin{array}{c}\texttt{object } O[\![\_]\!](\overline{z : \_})\ \_\ \texttt{end} \in \Delta \\ msa(\Delta, O[\![\overline{\gamma}]\!], m, (\overline{ilk(v)})) = \big\{\big(m[\![\overline{\{\_\} <: P <: \{\_\}}]\!](\overline{x{:}\,\alpha}){:}\,\rho = e, \_\big)\big\}\end{array}}{\Delta \vdash E\big[O[\![\overline{\gamma}]\!](\overline{v}).m(\overline{v'})\big] \longrightarrow E\big[[O[\![\overline{\gamma}]\!](\overline{v})/\texttt{self}]\overline{[\texttt{self}.z/z]}\overline{[v'/x]}\overline{[\tau/P]}e\big]} \qquad \text{(R-METHOD)}$$

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

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

Figure 9. Dynamic Semantics: Evaluation Rules

6. Subtyping and Type Exclusion

Figure 10 gives rules for computing a set of constraints that, if satisfied, will guarantee a (reflexive) subtype relationship between two types. (Conversely, if the constraints are not satisfiable, then a subtype relationship does *not* hold between the two types.)

Figure 11 gives rules for computing a set of constraints that, if satisfied, will guarantee an exclusion relationship between two types. (Conversely, if the constraints are not satisfiable, then an exclusion relationship does *not* hold between the two types.)

Subtyping Rules That Generate Constraints: $\boxed{\Delta \vdash \alpha <: \alpha \Leftarrow \mathcal{C}}$

Logical rules

$$\Delta \vdash \text{Bottom} <: \alpha \Leftarrow \text{true}$$

$$\Delta \vdash c <: \text{Bottom} \Leftarrow \text{false}$$

$$\Delta \vdash (\alpha \to \rho) <: \text{Bottom} \Leftarrow \text{false}$$

$$\Delta \vdash \alpha <: \text{Any} \Leftarrow \text{true}$$

$$\Delta \vdash \text{Any} <: (\alpha \to \omega) \Leftarrow \text{false}$$

$$\frac{\Delta \vdash \alpha <: \eta \Leftarrow \mathcal{C} \quad \Delta \vdash \alpha <: \chi \Leftarrow \mathcal{C}'}{\Delta \vdash \alpha <: (\eta \cap \chi) \Leftarrow \mathcal{C} \wedge \mathcal{C}'}$$

$$\frac{\Delta \vdash \alpha <: \chi \Leftarrow \mathcal{C} \quad \Delta \vdash \eta <: \chi \Leftarrow \mathcal{C}' \quad \Delta \vdash \alpha \Diamond \eta \Leftarrow \mathcal{C}''}{\Delta \vdash (\alpha \cap \eta) <: \chi \Leftarrow \mathcal{C} \vee \mathcal{C}' \vee \mathcal{C}''}$$

$$\frac{\Delta \vdash \alpha <: \eta \Leftarrow \mathcal{C} \quad \Delta \vdash \alpha <: \chi \Leftarrow \mathcal{C}'}{\Delta \vdash \alpha <: (\eta \cup \chi) \Leftarrow \mathcal{C} \vee \mathcal{C}'}$$

$$\frac{\Delta \vdash \alpha <: \chi \Leftarrow \mathcal{C} \quad \Delta \vdash \eta <: \chi \Leftarrow \mathcal{C}'}{\Delta \vdash (\alpha \cup \eta) <: \chi \Leftarrow \mathcal{C} \wedge \mathcal{C}'}$$

Inference Variables

$$\frac{I \notin \mathit{parameters}(\Delta)}{\Delta \vdash I <: I \Leftarrow \text{true}}$$

$$\frac{I \notin \mathit{parameters}(\Delta)}{\Delta \vdash I <: \alpha \Leftarrow I <: \alpha}$$

$$\frac{I \notin \mathit{parameters}(\Delta)}{\Delta \vdash \alpha <: I \Leftarrow \alpha <: I}$$

Bound Variables

$$\Delta \vdash P <: P \Leftarrow \text{true}$$

$$\frac{\{\_\} <: P <: \{\overline{\xi}\} \in \Delta}{\Delta \vdash \overline{\underline{P} <: \xi \Leftarrow \text{true}}}$$

$$\frac{\{\overline{\zeta}\} <: P <: \{\_\} \in \Delta}{\Delta \vdash \overline{\zeta <: \underline{P} \Leftarrow \text{true}}}$$

$$\frac{\Delta \vdash \alpha <: \text{Bottom} \Leftarrow \mathcal{C}}{\Delta \vdash \alpha <: P \Leftarrow \mathcal{C}}$$

Structural rules

$$\frac{\#(\overline{\alpha}) = \#(\overline{\eta}) \qquad \Delta \vdash \overline{\alpha <: \eta \Leftarrow \mathcal{C}} \quad \Delta \vdash \overline{\alpha <: \text{Bottom} \Leftarrow \mathcal{C}'}}{\Delta \vdash (\overline{\alpha}) <: (\overline{\eta}) \Leftarrow \left(\bigwedge \{\overline{\mathcal{C}}\}\right) \vee \left(\bigvee \{\overline{\mathcal{C}'}\}\right)}$$

$$\frac{\#(\overline{\alpha}) \neq 1 \qquad \Delta \vdash \overline{\alpha <: \text{Bottom} \Leftarrow \mathcal{C}}}{\Delta \vdash (\overline{\alpha}) <: \eta \Leftarrow \bigvee \{\overline{\mathcal{C}}\}}$$

$$\frac{\Delta \vdash \alpha' <: \alpha \Leftarrow \mathcal{C} \qquad \Delta \vdash \rho <: \rho' \Leftarrow \mathcal{C}'}{\Delta \vdash (\alpha \to \rho) <: (\alpha' \to \rho') \Leftarrow \mathcal{C} \wedge \mathcal{C}'}$$

$$\frac{\#(\overline{U}) \neq 1}{\Delta \vdash (\alpha \to \rho) <: (\overline{U}) \Leftarrow \text{false}}$$

$$\frac{\#(\overline{\alpha}) \neq 1}{\Delta \vdash c <: (\overline{\alpha}) \Leftarrow \text{false}}$$

$$\Delta \vdash (\alpha \to \rho) <: c \Leftarrow \text{false}$$

$$\Delta \vdash c <: (\alpha \to \rho) \Leftarrow \text{false}$$

Constructed types

$$\frac{\mathit{distinct}(T, T') \qquad \{\overline{\tau}\} = \mathit{parents}(\Delta, T[\![\overline{\alpha}]\!]) \qquad \Delta \vdash \overline{\tau <: \underline{T'[\![\overline{\eta}]\!]} \Leftarrow \mathcal{C}}}{\Delta \vdash T[\![\overline{\alpha}]\!] <: T'[\![\overline{\eta}]\!] \Leftarrow \bigvee \{\overline{\mathcal{C}}\}}$$

$$\frac{\{\overline{\tau}\} = \mathit{parents}(\Delta, O[\![\overline{\alpha}]\!]) \qquad \Delta \vdash \overline{\tau <: \underline{T'[\![\overline{\eta}]\!]} \Leftarrow \mathcal{C}}}{\Delta \vdash O[\![\overline{\alpha}]\!] <: T'[\![\overline{\eta}]\!] \Leftarrow \bigvee \{\overline{\mathcal{C}}\}}$$

$$\frac{\texttt{trait}\ T[\![\overline{V\ P <: \{\_\}}]\!]\ \_\ \texttt{end} \in \{\Delta\} \qquad \Delta \vdash \overline{\alpha\ V\ \eta \Leftarrow \mathcal{C}}}{\Delta \vdash T[\![\overline{\alpha}]\!] <: T[\![\overline{\eta}]\!] \Leftarrow \bigwedge \{\overline{\mathcal{C}}\}}$$

Variance test: $\boxed{\Delta \vdash \alpha\ V\ \alpha \Leftarrow \mathcal{C}}$

$$\frac{\Delta \vdash \alpha <: \eta \Leftarrow \mathcal{C}}{\Delta \vdash \alpha\ \texttt{covariant}\ \eta \Leftarrow \mathcal{C}}$$

$$\frac{\Delta \vdash \eta <: \alpha \Leftarrow \mathcal{C}}{\Delta \vdash \alpha\ \texttt{contravariant}\ \eta \Leftarrow \mathcal{C}}$$

$$\frac{\Delta \vdash \alpha \equiv \eta \Leftarrow \mathcal{C}}{\Delta \vdash \alpha\ \texttt{invariant}\ \eta \Leftarrow \mathcal{C}}$$

Figure 10. Algorithm for generating subtyping constraints. Apply the first rule that matches.

Exclusion Rules That Generate Constraints: $\boxed{\Delta \vdash \alpha \lozenge \alpha \Leftarrow \mathcal{C}}$

Logical rules

$$\Delta \vdash \text{Bottom} \lozenge \alpha \Leftarrow \text{true}$$

$$\frac{\Delta \vdash \alpha <: \text{Bottom} \Leftarrow \mathcal{C}}{\Delta \vdash \text{Any} \lozenge \alpha \Leftarrow \mathcal{C}}$$

$$\frac{\#(\overline{\alpha}) \neq 1}{\Delta \vdash (\overline{\alpha}) \lozenge \text{Object} \Leftarrow \mathit{true}}$$

$$\Delta \vdash (\alpha \to \rho) \lozenge \text{Object} \Leftarrow \mathit{true}$$

$$\frac{\Delta \vdash \alpha \lozenge \chi \Leftarrow \mathcal{C} \qquad \Delta \vdash \eta \lozenge \chi \Leftarrow \mathcal{C}' \qquad \Delta \vdash (\alpha \cap \eta) <: \text{Bottom} \Leftarrow \mathcal{C}''}{\Delta \vdash (\alpha \cap \eta) \lozenge \chi \Leftarrow \mathcal{C} \vee \mathcal{C}' \vee \mathcal{C}''}$$

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

Inference Variables

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

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

Bound Variables

$$\frac{\{\_\} <: P <: \{\overline{\xi}\} \in \Delta \qquad \Delta \vdash \overline{\xi \lozenge \alpha \Leftarrow \mathcal{C}}}{\Delta \vdash P \lozenge \alpha \Leftarrow \bigvee \{\overline{\mathcal{C}}\}}$$

Structural rules

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

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

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

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

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

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

Constructed types

$$\frac{\Delta \vdash c \lozenge_{\text{x}} c' \Leftarrow \mathcal{C}_x \qquad \Delta \vdash c \lozenge_{\text{c}} c' \Leftarrow \mathcal{C}_c \qquad \Delta \vdash c \lozenge_{\text{o}} c' \Leftarrow \mathcal{C}_o \qquad \Delta \vdash c \lozenge_{\text{m}} c' \Leftarrow \mathcal{C}_m}{\Delta \vdash c \lozenge c' \Leftarrow \mathcal{C}_x \vee \mathcal{C}_c \vee \mathcal{C}_o \vee \mathcal{C}_m}$$

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

$$\frac{\Delta \vdash c \rhd_{\text{c}} c' \Leftarrow \mathcal{C} \qquad \Delta \vdash c' \rhd_{\text{c}} c \Leftarrow \mathcal{C}'}{\Delta \vdash c \lozenge_{\text{c}} c' \Leftarrow \mathcal{C} \vee \mathcal{C}'}$$

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

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

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

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

$$\frac{\mathit{distinct}(T, T') \qquad \{\overline{\chi}\} = \mathit{ancestors}(\Delta, T[\![\overline{\alpha}]\!]) \qquad \{\overline{\omega}\} = \bigcup\{\overline{\mathit{excluded}(\Delta, \chi)}\} \qquad \Delta \vdash \overline{T'[\![\overline{\eta}]\!] <: \omega \Leftarrow \mathcal{C}}}{\Delta \vdash T[\![\overline{\alpha}]\!] \rhd_{\text{x}} T'[\![\overline{\eta}]\!] \Leftarrow \bigvee\{\overline{\mathcal{C}}\}}$$

$$\frac{\mathit{distinct}(T, T') \qquad \{\overline{\omega}\} = \mathit{comprised}(\Delta, T[\![\overline{\alpha}]\!]) \qquad \Delta \vdash \overline{T'[\![\overline{\eta}]\!] \lozenge \omega \Leftarrow \mathcal{C}}}{\Delta \vdash T[\![\overline{\alpha}]\!] \rhd_{\text{c}} T'[\![\overline{\eta}]\!] \Leftarrow \bigwedge\{\overline{\mathcal{C}}\}}$$

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

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

$$\frac{\{\overline{\chi}\} = \mathit{ancestors}(\Delta, T[\![\overline{\alpha}]\!]) \qquad \{\overline{\omega}\} = \mathit{ancestors}(\Delta, T'[\![\overline{\eta}]\!]) \qquad \Delta \vdash \overline{\overline{\chi \lozenge_{\text{m}} \underline{\omega} \Leftarrow \mathcal{C}}}}{\Delta \vdash T[\![\overline{\alpha}]\!] \lozenge_{\text{m}} T'[\![\overline{\eta}]\!] \Leftarrow \bigvee\left\{\overline{\overline{\mathcal{C}}}\right\}}$$

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

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

Figure 11. Algorithm for generating exclusion constraints. Each rule is symmetric; apply the first one that matches.

Constrained type equivalence: $\boxed{\Delta \vdash \alpha \equiv \alpha \Leftarrow \mathcal{C}}$

$$\frac{\Delta \vdash \alpha <: \eta \Leftarrow \mathcal{C} \qquad \Delta \vdash \eta <: \alpha \Leftarrow \mathcal{C}'}{\Delta \vdash \alpha <: \eta \Leftarrow \mathcal{C} \wedge \mathcal{C}'}$$

Unconditional subtyping: $\boxed{\Delta \vdash \alpha <: \alpha}$

$$\frac{\Delta \vdash \alpha <: \eta \Leftarrow \text{true}}{\Delta \vdash \alpha <: \eta}$$

Unconditional exclusion: $\boxed{\Delta \vdash \alpha \lozenge \alpha}$

$$\frac{\Delta \vdash \alpha \lozenge \eta \Leftarrow \text{true}}{\Delta \vdash \alpha \lozenge \eta}$$

Unconditional type equivalence: $\boxed{\Delta \vdash \alpha \equiv \alpha}$

$$\frac{\Delta \vdash \alpha \equiv \eta \Leftarrow \text{true}}{\Delta \vdash \alpha \equiv \eta}$$

Unconditional type not-equivalence: $\boxed{\Delta \vdash \alpha \not\equiv \alpha}$

$$\frac{\Delta \vdash \alpha \not\equiv \eta \Leftarrow \text{true}}{\Delta \vdash \alpha \not\equiv \eta}$$

Conditional type equivalence: $\boxed{\Delta \vdash \alpha \equiv \alpha \Leftarrow \mathcal{C}}$

$$\frac{\Delta \vdash \alpha <: \eta \Leftarrow \mathcal{C} \qquad \Delta \vdash \eta <: \alpha \Leftarrow \mathcal{C}'}{\Delta \vdash \alpha \equiv \eta \Leftarrow \mathcal{C} \wedge \mathcal{C}'}$$

Conditional type non-equivalence: $\boxed{\Delta \vdash \alpha \not\equiv \alpha \Leftarrow \mathcal{C}}$

$$\frac{\Delta \vdash \alpha \not<: \eta \Leftarrow \mathcal{C} \qquad \Delta \vdash \eta \not<: \alpha \Leftarrow \mathcal{C}'}{\Delta \vdash \alpha \not\equiv \eta \Leftarrow \mathcal{C} \vee \mathcal{C}'}$$

Converse type non-equivalence: $\boxed{\Delta \vdash \alpha \not\equiv \alpha \Rightarrow \mathcal{C}}$

$$\frac{\Delta \vdash \alpha \equiv \eta \Leftarrow \mathcal{C}}{\Delta \vdash \alpha \not\equiv \eta \Rightarrow \neg(\mathcal{C})}$$

Figure 12. Miscellaneous Typing Judgments

Figure 12 gives rules that define type equivalence in terms of subtyping and define unconditional versions of the subtyping, type exclusion, and type equivalence relationships.

Figure 13 gives rules for deciding whether one existentially quantified type is a subtype of another, and whether one universally quantified type is a subtype of another.

Existential subtyping: $\boxed{\Delta \vdash \Phi \lesssim \Phi \quad \Delta \vdash \Phi \leq \Phi}$

$$\frac{\begin{array}{c}\{\overline{P}\} \cap (FV(\eta) \cup FV(\overline{\zeta'}) \cup FV(\overline{\xi'})) = \emptyset \\ \Delta' = \Delta, \overline{\{\zeta\} <: P <: \{\xi\}} \qquad \Delta' \vdash \overline{[\overline{\chi}/\overline{Q}]\zeta' <: \chi} \\ \Delta' \vdash \alpha <: [\overline{\chi}/\overline{Q}]\eta \qquad \Delta' \vdash \overline{\chi <: [\overline{\chi}/\overline{Q}]\xi'}\end{array}}{\Delta \vdash \exists[\![\overline{\{\zeta\} <: P <: \{\xi\}}]\!]\alpha \lesssim \exists[\![\{\overline{\zeta'}\} <: Q <: \{\overline{\xi'}\}]\!]\eta}$$

$$\frac{\Delta \vdash \Phi \xrightarrow{\equiv} \Phi' \text{ using } \_ \qquad \Delta \vdash \Phi' \lesssim \Phi''}{\Delta \vdash \Phi \leq \Phi''}$$

Universal subtyping: $\boxed{\Delta \vdash \Psi \lesssim \Psi \quad \Delta \vdash \Psi \leq \Psi}$

$$\frac{\begin{array}{c}\{\overline{Q}\} \cap (FV(\alpha) \cup FV(\overline{\zeta}) \cup FV(\overline{\xi})) = \emptyset \\ \Delta' = \Delta, \overline{\{\zeta'\} <: Q <: \{\xi'\}} \qquad \Delta' \vdash \overline{[\overline{\chi}/\overline{P}]\{\zeta\} <: \chi} \\ \Delta' \vdash [\overline{\chi}/\overline{P}]\alpha <: \eta \qquad \Delta' \vdash \overline{\chi <: [\overline{\chi}/\overline{P}]\{\xi\}}\end{array}}{\Delta \vdash \forall[\![\overline{\{\zeta\} <: P <: \{\xi\}}]\!]\alpha \lesssim \forall[\![\overline{\{\zeta'\} <: Q <: \{\xi'\}}]\!]\eta}$$

$$\frac{\Delta \vdash \Psi \lesssim \Psi'' \qquad \Delta \vdash \Psi' \xrightarrow{\equiv} \Psi'' \text{ using } \_}{\Delta \vdash \Psi \leq \Psi'}$$

Existential Reduction: $\boxed{\Delta \vdash \Phi \xrightarrow{\equiv} \Phi \text{ using } \sigma}$

$$\frac{\Delta \vdash \alpha \not\equiv \text{Bottom} \Rightarrow \mathcal{C} \qquad \mathit{toConstraint}(\lambda) = \mathcal{C}' \qquad \Delta \vdash \mathit{unify}(\mathcal{C} \wedge \mathcal{C}') = (\sigma, \mathcal{C}'') \qquad \mathit{toBounds}(\mathcal{C}'') = \lambda'}{\Delta \vdash \exists[\![\overline{\lambda}]\!]\alpha \xrightarrow{\equiv} \exists[\![\overline{\lambda'}]\!]\sigma(\alpha) \text{ using } \sigma}$$

$$\frac{\text{otherwise}}{\Delta \vdash \exists[\![\overline{\lambda}]\!]\alpha \xrightarrow{\equiv} \exists[\![\overline{\lambda}]\!]\alpha \text{ using } [/]}$$

Universal Reduction: $\boxed{\Delta \vdash \Psi \xrightarrow{\equiv} \Psi \text{ using } \sigma}$

$$\frac{\Delta \vdash \exists[\![\overline{\lambda}]\!]\alpha \xrightarrow{\equiv} \exists[\![\overline{\lambda'}]\!]\alpha' \text{ using } \sigma}{\Delta \vdash \forall[\![\overline{\lambda}]\!](\alpha \to T\omega) \xrightarrow{\equiv} \forall[\![\overline{\lambda'}]\!](\alpha' \to \sigma(\omega)) \text{ using } \sigma}$$

Figure 13. Typing Rules for Quantified Types. Note that alpha-renaming of type variables may be necessary to apply these rules.

7. Run-time Type Solving

8. Examples

9. Related Work

10. 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.