Notations

$a, b \in \mathsf{Ter}$	terms
$d \in D, e \in E$	iterators over some finite sets
$f, g, h \in \mathsf{F}$	field names
$i, j, k \in \mathbb{N}$	indices (usually $i < j < k$)
$l \in \mathsf{Loc}$	locations
$m, m_d, m_e \in \mathsf{M}$	method names
$u, v, w \in \mathsf{Val}$	values
$x, y, z \in \mathsf{Var}$	variables
C	evaluation context
F, G, H	functionals
S	store
α, β, τ	semantic types
σ, γ	substitutions (γ is "ground")
Ψ	store typing
Γ	type environment

Functional Object Calculus

Syntax

a, b	$::=$	x	variable
	$\mid$	$[m_d = \varsigma(x_d)b_d]_{d \in D}$	object creation
	$\mid$	$a.m$	method invocation (field selection)
	$\mid$	$a.m := \varsigma(x)b$	method update (field update)
	$\mid$	$\Lambda.a$	type abstraction
	$\mid$	$a\,[]$	type application
	$\mid$	pack a	creating existential package
	$\mid$	open a as x in b	opening package

Values

$$v \in \mathsf{Val} ::= [m_d = \varsigma(x_d)b_d]_{d \in D} \mid \Lambda.a \mid \text{pack } v$$

Free Variables

$$\begin{array}{l} FV : \mathsf{Ter} \to \mathcal{P}_{fin}(\mathsf{Var}) \\ FV(x) = \{x\} \\ FV([m_d = \varsigma(x_d)b_d]_{d \in D}) = \bigcup_{d \in D}(FV(b_d) \setminus \{x_d\}) \\ FV(a.m) = FV(a) \\ FV(a.m := \varsigma(x)b) = FV(a) \cup (FV(b) \setminus \{x\}) \\ FV(\Lambda.a) = FV(a) \\ FV(a\,[]) = FV(a) \\ FV(\text{pack } a) = FV(a) \\ FV(\text{open } a \text{ as } x \text{ in } b) = FV(a) \cup (FV(b) \setminus \{x\}) \end{array}$$

Closed Values

$$\mathsf{CVal} = \{v \in \mathsf{Val} \mid FV(v) = \emptyset\}$$

Applying Ground Substitutions

$\mathsf{sub} : \mathsf{Ter} \to (\mathsf{Var} \rightharpoonup_{\mathit{fin}} \mathsf{Val}) \to \mathsf{Ter}$
Notation: $\mathrm{sub}\ a\ \sigma = \sigma(a)$
$\sigma(x) = \sigma(x)$
$\sigma([m_d = \varsigma(x_d)b_d]_{d\in D}) = [m_d = \varsigma(x_d)\sigma[x_d \mapsto](b_d)]_{d\in D}$
$\sigma(a.m) = \sigma(a).m$
$\sigma(a.m := \varsigma(x)b) = (\sigma(a)).m := \varsigma(x)\sigma[x \mapsto](b)$
$\sigma(\Lambda.a) = \Lambda.\sigma(a)$
$\sigma(a\,[]) = \sigma(a)\,[]$
$\sigma(\text{pack } a) = \text{pack } \sigma(a)$
$\sigma(\text{open } a \text{ as } x \text{ in } b) = \text{open } \sigma(a) \text{ as } x \text{ in } \sigma[x \mapsto](b)$

Small-step Operational Semantics

Reduction Contexts

$$C[\bullet] = \bullet \mid C.m \mid C.m := \varsigma(x)b \mid C\,[] \mid \text{pack } C \mid \text{open } C \text{ as } x \text{ in } b$$

Reduction

$$(\textsc{Red-Sel})\quad \frac{v = [m_d = \varsigma(x_d)b_d]_{d\in D} \quad e \in D}{v.m_e \to [x_e \mapsto v](b_e)}$$

$$(\textsc{Red-Upd})\quad \frac{e \in D \quad v = [m_e = \varsigma(x)b, m_d = \varsigma(x_d)b_d]_{d\in D\setminus\{e\}}}{[m_d = \varsigma(x_d)b_d]_{d\in D}.m_e := \varsigma(x)b \to v}$$

$$(\textsc{Red-TApp})\ (\Lambda.a)\,[] \to a$$

$$(\textsc{Red-Open})\ \text{open (pack } v) \text{ as } x \text{ in } b \to [x \mapsto v](b)$$

$$(\textsc{Red-Ctx})\quad \frac{a \to b}{C[a] \to C[b]}$$

Properties

- Reduction is deterministic.

Notation

$\mathit{irred}(a) = \neg\exists b.a \to b$

Step-indexed Semantics

Safety

$\mathsf{Safe}_k = \{a \mid \forall j < k.\ a \rightarrow^j b \Rightarrow b \in \mathsf{Val} \vee \neg \mathit{irred}(b)\}$

$\mathsf{Safe} = \bigcap_{k \geq 0} \mathsf{Safe}_k$

Types

Sets of index-value pairs closed under descending index.

$\mathsf{Type} = \{\tau \subseteq \mathbb{N} \times \mathsf{Val} \mid \forall k \geq 0.\ \forall j \leq k.\ \forall v \in \mathsf{Val}.\ \langle k, v\rangle \in \tau \Rightarrow \langle j, v\rangle \in \tau\}$

Expr:$_k$Type

$a :_k \tau \ :\Leftrightarrow\ FV(a) = \emptyset \ \wedge\ \forall j < k.\ (a \rightarrow^j b \ \wedge\ \mathit{irred}(b)) \Rightarrow \langle k - j, b\rangle \in \tau$

Substitution:$_k$Environment

$\Gamma : \mathsf{Var} \rightharpoonup_{\mathit{fin}} \mathsf{Type}$ (type environment)

$\sigma : \mathsf{Var} \rightharpoonup_{\mathit{fin}} \mathsf{CVal}$ (ground substitution)

$\sigma :_k \Gamma \ :\Leftrightarrow\ \forall x \in \mathit{Dom}(\Gamma).\ \sigma(x) :_k \Gamma(x)$

Semantic Type Judgement

$\Gamma \models_k a : \alpha \ :\Leftrightarrow\ FV(a) \subseteq \mathit{Dom}(\Gamma) \ \wedge\ \forall \sigma :_k \Gamma.\ \sigma(a) :_k \alpha$

$\Gamma \models a : \alpha \ :\Leftrightarrow\ \forall k \geq 0.\ \Gamma \models_k a : \alpha$

$\models a : \alpha \ :\Leftrightarrow\ \emptyset \models a : \alpha$

Properties

1. $\forall j \leq k.\ a :_k \tau \Rightarrow a :_j \tau$
2. $\forall v \in \mathsf{Val}.\ v :_k \tau \Leftrightarrow \langle k, v\rangle \in \tau$
3. $a :_k \tau \Rightarrow a \in \mathsf{Safe}_k$
4. $\models a : \alpha \Rightarrow a \in \mathsf{Safe}$ (type soundness)

Approximation

$\lfloor \tau \rfloor_k = \{\langle j, v\rangle \mid j < k\}$

$F : \mathsf{Type} \rightarrow \mathsf{Type}$ *contractive* if $\forall \tau \in \mathsf{Type},\ \forall k \geq 0$ we have:

$$\lfloor F(\tau) \rfloor_{k+1} = \lfloor F(\lfloor \tau \rfloor_k) \rfloor_{k+1}$$

Semantic Types

$\bot = \emptyset$

$\top = \{\langle k, v\rangle \mid k \in \mathbb{N}, v \in \mathsf{Val}\}$

$[m_d : \tau_d]_{d\in D} = \{\langle k, v\rangle \mid v = [m_e = \varsigma(x_e)b_e]_{e\in E}, D \subseteq E, \alpha = [m_d : \tau_d]_{d\in D},$
$\forall d \in D.\, \forall j < k.\, ([x_d \mapsto v](b_d) :_j \tau_d \ \wedge\ \forall \varsigma(x)b.\, \langle j, \varsigma(x)b\rangle \in \alpha \to \tau_d$
$\Rightarrow \langle j, [m_d = \varsigma(x)b, m_e = \varsigma(x_e)b_e]_{e\in E\setminus\{d\}}\rangle \in \alpha)\}$

where $\alpha \to \tau = \{\langle j, \varsigma(x)b\rangle \mid \forall i < j.\, \langle i, v\rangle \in \alpha \Rightarrow [x \mapsto v](b) :_i \tau\}$ [1]

$\mu F = \{\langle k, v\rangle \mid \langle k, v\rangle \in F^{k+1}(\bot)\}$ [2]

$\forall_\alpha F = \{\langle k, \Lambda.a\rangle \mid \forall j < k.\, \forall \tau.\, \lfloor\tau\rfloor_j \in \mathsf{Type} \ \wedge\ \lfloor\tau\rfloor_j \subseteq \lfloor\alpha\rfloor_j \Rightarrow a :_j F(\tau)\}$

$\exists_\alpha F = \{\langle k, \mathrm{pack}\ v\rangle \mid \exists \tau \in \mathsf{Type}.\, \lfloor\tau\rfloor_k \subseteq \lfloor\alpha\rfloor_k \ \wedge\ \forall j < k.\, \langle j, k\rangle \in F(\tau)\}$

[1] $\alpha \to \tau$ is just a set, and not a type, because methods are not values (not even terms) in the functional object calculus.

[2] F is a function from types to types (functional)

Imperative Object Calculus

Syntax

a, b	$::=$	x	variables
	$\mid$	$\{m_d = l_d\}_{d \in D}$	object value
	$\mid$	$[m_d = \varsigma(x_d) b_d]_{d \in D}$	object creation
	$\mid$	clone a	object cloning
	$\mid$	$a.m$	method invocation
	$\mid$	$a.m := \varsigma(x)b$	method update
	$\mid$	$\lambda x.\, b$	procedures
	$\mid$	$a\ b$	procedure application
	$\mid$	$\Lambda.a$	type abstraction
	$\mid$	$a\,[]$	type application
	$\mid$	pack a	creating existential package
	$\mid$	open a as x in b	opening package

Examples

1. Factorial

$$[fac = \varsigma(y)\lambda n.\, \text{if } n = 0 \text{ then } 1 \text{ else } n \times y.fac(n-1)]$$

2. Euclid's GCD Algorithm

$$\begin{aligned}[gcd = \varsigma(y)\lambda x.\, \lambda z.\, &\text{if } x < z \text{ then } y.gcd\ x\ (z - x)\\ &\text{else if } z < x \text{ then } y.gcd\ (x - z)\ z \text{ else } x]\end{aligned}$$

3. Factorial through field (could be used for explaining what higher-order store is, even though there are simpler examples to show this)

$$\begin{aligned}&\text{let } x\ =\ [f = \varsigma(y)\lambda n.\, n] \text{ in}\\ &\text{let } z\ =\ [f = x, fac = \varsigma(y)\lambda n.\, \text{if } n = 0 \text{ then } 1\\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{else } n \times y.f.fac(n-1)] \text{ in}\\ &z.f := z\end{aligned}$$

Values

$$v \in \mathsf{Val} ::= \{m_d = l_d\}_{d \in D} \mid \lambda x.\, b \mid \Lambda.a \mid \text{pack } v$$

$$\mathsf{CVal} = \{v \in \mathsf{Val} \mid FV(v) = \emptyset\}$$

Programs

$$\mathsf{Prog} = \{v \in \mathsf{CVal} \mid \mathit{Labels}(v) = \emptyset\}$$

Small-step Operational Semantics

Reduction Contexts

$$C[\bullet] = \bullet \mid C.m \mid C.m := \varsigma(x)b \mid C\ b \mid v\ C \mid C\ [] \mid \text{pack } C \mid \text{open } C \text{ as } x \text{ in } b$$

Reduction

$$\text{(Red-Obj)} \quad \frac{\forall d \in D.\ l_d \notin Dom(S)}{\langle S, [m_d = \varsigma(x_d)b_d]_{d\in D}\rangle \to \langle S[l_d \mapsto \lambda x_d.\, b_d]_{d\in D}, \{m_d = l_d\}_{d\in D}\rangle}$$

$$\text{(Red-Clone)} \quad \frac{\forall d \in D.\ l'_d \notin Dom(S)}{\langle S, \text{clone } \{m_d = l_d\}_{d\in D}\rangle \to \langle S[l'_d \mapsto S(l_d)]_{d\in D}, \{m_d = l'_d\}_{d\in D}\rangle}$$

$$\text{(Red-Sel}^3\text{)} \quad \frac{e \in D}{\langle S, \{m_d = l_d\}_{d\in D}.m_e\rangle \to \langle S, S(l_e)\ \{m_d = l_d\}_{d\in D}\rangle}$$

$$\text{(Red-Upd)} \quad \frac{e \in D}{\langle S, \{m_d = l_d\}_{d\in D}.m_e := \varsigma(x)b\rangle \to \langle S[l_e \mapsto \lambda x.\, b], \{m_d = l_d\}_{d\in D}\rangle}$$

$$\text{(Red-Beta)} \quad \langle S, (\lambda x.\, b)\ v\rangle \to \langle S, [x \mapsto v](b)\rangle$$

$$\text{(Red-Ctx)} \quad \frac{\langle S, a\rangle \to \langle S', b\rangle}{\langle S, C[a]\rangle \to \langle S', C[b]\rangle}$$

Examples

TODO: Redo examples from ioc.tex

[3] This rule is not standard (basically a selection step is always succeeded by a beta reduction), but it allows us to reuse Ahmed's model, which is nice.

Step-indexed Semantics[4]

Safety

$$\mathsf{Safe}_k = \{(S,a) \mid \forall j < k.\ \forall S', b.\ \langle S,a\rangle \rightarrow^j \langle S',b\rangle \Rightarrow b \in \mathsf{Val} \vee \neg \mathit{irred}(S',b)\}$$

$$\mathsf{Safe} = \bigcap_{k\geq 0} \mathsf{Safe}_k$$

Circular "definition"

$$\mathsf{PreType} = \mathcal{P}(\mathbb{N} \times \mathsf{StoreType} \times \mathsf{Val})$$

$$\mathsf{StoreType} = \mathsf{Loc} \rightharpoonup_{\mathit{fin}} \mathsf{PreType}$$

Approximation

$$\lfloor \tau \rfloor_k = \{\langle j, \Psi, v\rangle \in \tau \mid j < k\}$$

$$\lfloor \Psi \rfloor_k = \lambda l \in \mathsf{Loc}.\ \lfloor \Psi(l) \rfloor_k$$

Stratification

Invariant For all types τ, $\lfloor \tau \rfloor_{k+1}$ cannot depend on any type beyond approximation k. This invariant assures the well-foundness of the whole construction.

$$\mathsf{PreType}_0 = \{\emptyset\}$$

$$\mathsf{PreType}_{k+1} = \{\tau \mid \forall \langle j, \Psi, v\rangle \in \tau.\ j \leq k\ \wedge\ \Psi \in \mathsf{StoreType}_j\}$$

$$\mathsf{StoreType}_k = \{\Psi \mid \forall l \in \mathit{Dom}(\Psi).\ \Psi(l) \in \mathsf{Type}_k\}$$

$$\mathsf{PreType} = \{\tau \mid \forall k \geq 0.\ \lfloor \tau \rfloor_k \in \mathsf{PreType}_k\}$$

$$\mathsf{StoryType} = \{\Psi \mid \forall k \geq 0.\ \lfloor \Psi \rfloor_k \in \mathsf{StoreType}_k\}$$

State Extension

$$(k,\Psi) \sqsubseteq (j,\Psi') \ :\Leftrightarrow\ j \leq k\ \wedge\ \forall l \in \mathit{Dom}(\Psi).\ \lfloor \Psi' \rfloor_j(l) = \lfloor \Psi \rfloor_j(l)$$

Types

PreTypes closed under state extension.

$$\begin{aligned}\mathsf{Type} = \{&\tau \in \mathsf{PreType} \mid \forall k, j \geq 0.\ \forall \Psi, \Psi'.\ \forall v \in \mathsf{Val}.\\ &((k,\Psi) \sqsubseteq (j,\Psi')\ \wedge\ \langle k,\Psi,v\rangle \in \tau) \Rightarrow \langle j,\Psi',v\rangle \in \tau\}\end{aligned}$$

Well-typed Store $(S :_k \Psi)$

$$\begin{aligned}(S :_k \Psi\ :\Leftrightarrow\ &\mathit{Dom}(\Psi) \subseteq \mathit{Dom}(S)\ \wedge\ \forall j < k.\\ &\forall l \in \mathit{Dom}(\Psi).\ \langle j, \lfloor \Psi \rfloor_j, S(l)\rangle \in \lfloor \Psi \rfloor_k(l)\end{aligned}$$

[4] All the definitions are the same as in Ahmed's thesis, modulo small notational differences.

Expr:$_{k,\Psi}$Type

$a :_{k,\Psi} \tau \;:\Leftrightarrow\; FV(a) = \emptyset \;\wedge\; \forall j < k, S, S', b.\; (S :_k \Psi \;\wedge\; \langle S, a\rangle \to^j \langle S', b\rangle \;\wedge\; \mathit{irred}(S', b)) \Rightarrow \exists \Psi'.\; (k, \Psi) \sqsubseteq (k-j, \Psi') \;\wedge\; S' :_{k-j} \Psi' \;\wedge\; \langle k-j, \Psi', b\rangle \in \tau$

Value Env.:$_{k,\Psi}$Type Env.

$\Gamma : \mathsf{Var} \rightharpoonup_{\mathit{fin}} \mathsf{Type}$ (type environment)

$\gamma : \mathsf{Var} \rightharpoonup_{\mathit{fin}} \mathsf{CVal}$ (value environment = closed-value substitution)

$\gamma :_{k,\Psi} \Gamma \;:\Leftrightarrow\; \forall x \in \mathit{Dom}(\Gamma).\; \gamma(x) :_{k,\Psi} \Gamma(x)$

Semantic Type Judgement

Let a such that $\mathit{Labels}(a) = \emptyset$.

$\Gamma \models_k a : \alpha \;:\Leftrightarrow\; FV(a) \subseteq \mathit{Dom}(\Gamma) \;\wedge\; \forall \Psi.\; \forall \gamma :_{k,\Psi} \Gamma.\; \gamma(a) :_{k,\Psi} \alpha$

$\Gamma \models a : \alpha \;:\Leftrightarrow\; \forall k \geq 0.\; \Gamma \models_k a : \alpha$

$\models a : \alpha \;:\Leftrightarrow\; \emptyset \models a : \alpha$

Properties

1. $j \leq k \Rightarrow (k, \Psi) \sqsubseteq (j, \lfloor \Psi \rfloor_j)$
2. $(j \leq k \;\wedge\; a :_{k,\Psi} \alpha) \Rightarrow a :_{j, \lfloor \Psi \rfloor_j} \alpha$
3. $\forall v \in \mathsf{Val}.\; v :_{k,\Psi} \tau \Rightarrow \langle k, \Psi, v\rangle \in \tau$
4. $\models a : \alpha \Rightarrow \forall S.\; (S, a) \in \mathsf{Safe}$ (type soundness)

Semantic Types

$\bot = \emptyset$

$\top = \{\langle k, \Psi, v\rangle \mid k \in \mathbb{N}, \Psi \in \mathsf{StoreType}, v \in \mathsf{Val}\}$

$\alpha \to \beta = \{\langle k, \Psi, \lambda x.\, b\rangle \mid \forall j < k.\; \forall \Psi, v \in \mathsf{Val}.$
$((k, \Psi) \sqsubseteq (j, \Psi') \;\wedge\; \langle j, \Psi', v\rangle \in \alpha) \Rightarrow [x \mapsto v](b) :_{j,\Psi'} \tau\}$

$\alpha = [m_d : \tau_d]_{d \in D} = \{\langle k, \Psi, \{m_e = l_e\}_{e \in E}\rangle \mid D \subseteq E \;\wedge$
$\exists \alpha'. \lfloor \alpha' \rfloor_k \in \mathsf{Type} \;\wedge\; \lfloor \alpha' \rfloor_k \subseteq \lfloor \alpha \rfloor_k \;\wedge\; \forall j < k.$
$\langle j, \Psi, \{m_e = l_e\}_{e \in E}\rangle \in \alpha' \;\wedge\; \forall d \in D.\; \lfloor \Psi \rfloor_k(l_d) = \lfloor \alpha' \to \tau_d \rfloor_k\}$

$\mu F = \{\langle k, \Psi, v\rangle \mid \langle k, \Psi, v\rangle \in F^{k+1}(\bot)\}$

$\forall_\alpha F = \{\langle k, \Psi, \Lambda.a\rangle \mid \forall j, \Psi'.\; \forall \tau.\; (k, \Psi) \sqsubseteq (j, \Psi') \;\wedge\; \lfloor \tau \rfloor_j \in \mathsf{Type}$
$\wedge\; \lfloor \tau \rfloor_j \subseteq \lfloor \alpha \rfloor_j \Rightarrow \forall i < j.\; a :_{i, \lfloor \Psi' \rfloor_i} F(\tau)\}$ (where F is non-expansive)

$\exists_\alpha F = \{\langle k, \Psi, \mathrm{pack}\; v\rangle \mid \exists \tau. \lfloor \tau \rfloor_k \in \mathsf{Type} \;\wedge\; \lfloor \tau \rfloor_k \subseteq \lfloor \alpha \rfloor_k$
$\wedge\; \forall j < k.\; \langle j, \lfloor \Psi \rfloor_j, v\rangle \in F(\tau)\}$ (where F is non-expansive)

Semantic Typing Rules[5][6]

(Var) $\Gamma \models x : \Gamma(x)$

Procedures Types

(Lam) $$\frac{\Gamma[x : \alpha] \models b : \beta}{\Gamma \models \lambda x.\, b : \alpha \to \beta}$$

(App) $$\frac{\Gamma \models a : \beta{\to}\alpha \quad \Gamma \models b : \beta}{\Gamma \models a\ b : \alpha}$$

Object Types

Let $\alpha = [m_d : \tau_d]_{d \in D}$

(Obj) $$\frac{\forall d \in D.\, \Gamma[x_d : \alpha] \models b_d : \tau_d}{\Gamma \models [m_d = \varsigma(x_d) b_d]_{d \in D} : \alpha}$$

(Sel) $$\frac{\Gamma \models a : \alpha \quad e \in D}{\Gamma \models a.m_e : \tau_e}$$

(Upd) $$\frac{\Gamma \models a : \alpha \quad e \in D \quad \Gamma[x : \alpha] \models b : \tau_e}{\Gamma \models a.m_e := \varsigma(x) b : \alpha}$$

(Clone) $$\frac{\Gamma \models a : \alpha}{\Gamma \models \text{clone } a : \alpha}$$

Polymorphic Types

(TAbs) $$\frac{\forall \tau \in \mathsf{Type}.\, \tau \subseteq \alpha \Rightarrow \Gamma \models a : F(\tau)}{\Gamma \models \Lambda.a : \forall_\alpha F}$$

(TApp) $$\frac{\Gamma \models a : \forall_\alpha F \quad \tau \in \mathsf{Type} \quad \tau \subseteq \alpha}{\Gamma \models a\,[] : F(\tau)}$$

(Pack) $$\frac{\exists \tau \in \mathsf{Type}.\, \tau \subseteq \alpha \ \wedge\ \Gamma \models a : F(\tau)}{\Gamma \models \text{pack } a : \exists_\alpha F}$$

[5] Common for both calculi

[6] Proved as derived lemmata from the definitions.

$$(\textsc{Open})\ \frac{\Gamma \models a : \exists_\alpha F \quad \forall \tau \in \mathsf{Type}.\ \tau \subseteq \alpha \Rightarrow \Gamma[x : F(\tau)] \models b : \beta}{\Gamma \models \text{open } a \text{ as } x \text{ in } b : \beta}$$

Recursive Types

$$(\textsc{Unfold})\ \frac{\Gamma \models a : \mu F \quad F \text{ contractive}}{\Gamma \models a : F(\mu F)}$$

$$(\textsc{Fold})\ \frac{\Gamma \models a : F(\mu F) \quad F \text{ contractive}}{\Gamma \models a : \mu F}$$

Subtyping

$$(\textsc{SubSub})\ \frac{\Gamma \models a : \alpha \quad \alpha \subseteq \beta}{\Gamma \models a : \beta}$$

$$(\textsc{SubRefl})\ \alpha \subseteq \alpha \qquad (\textsc{SubTrans})\ \frac{\alpha \subseteq \tau \quad \tau \subseteq \beta}{\alpha \subseteq \beta}$$

$$(\textsc{SubTop})\ \alpha \subseteq \top \qquad (\textsc{SubBot})\ \bot \subseteq \alpha$$

$$(\textsc{SubArrow})\ \frac{\alpha' \subseteq \alpha' \quad \beta \subseteq \beta'}{\alpha \to \beta \subseteq \alpha' \to \beta'}$$

$$(\textsc{SubObj})\ \frac{E \subseteq D}{[m_d : \tau_d]_{d \in D} \subseteq [m_e : \tau_e]_{e \in E}}{}^{7}$$

$$(\textsc{SubRec})\ \frac{\forall \alpha, \beta \in \mathsf{Type}.\ \alpha \subseteq \beta \Rightarrow F(\alpha) \subseteq G(\beta)}{\mu F \subseteq \mu G}$$

$$(\textsc{SubUniv})\ \frac{\beta \subseteq \alpha \quad \forall \tau \in \mathsf{Type}.\ \tau \subseteq \beta \Rightarrow F(\tau) \subseteq G(\tau)}{\forall_\alpha F \subseteq \forall_\beta G}$$

$$(\textsc{SubExist})\ \frac{\alpha \subseteq \beta \quad \forall \tau \in \mathsf{Type}.\ \tau \subseteq \alpha \Rightarrow F(\tau) \subseteq G(\tau)}{\exists_\alpha F \subseteq \exists_\beta G}$$

[7] Still only in width without variance annotations.

Self Quantifier The self quantifier allows recursive types with proper subtyping, and is used together with object types in order to define self types.

$$\varsigma F = \mu(\lambda\alpha \in \mathsf{Type}.\, \exists_\alpha F)$$

$$(\textsc{Wrap})\ \frac{\exists\tau \in \mathsf{Type}.\, \tau \subseteq \varsigma F \ \wedge\ \Gamma \models a : F(\tau)}{\Gamma \models \text{pack } a : \varsigma F}$$

$$(\textsc{Use})\ \frac{\Gamma \models a : \varsigma F \quad \forall\tau \in \mathsf{Type}.\, \tau \subseteq \varsigma F \Rightarrow \Gamma[x : F(\tau)] \models b : \beta}{\Gamma \models \text{open } a \text{ as } x \text{ in } b : \beta}$$

$$(\textsc{SubSelf})\ \frac{\forall\tau \in \mathsf{Type}.\, F(\tau) \subseteq G(\tau)}{\varsigma F \subseteq \varsigma G}$$

Self Types Self types are recursive object types with proper subtyping[8]. Self types have the form $\varsigma(\lambda\tau \in \mathsf{Type}.\, [m_d : F_d(\tau)]_{d\in D})$. They have the restriction that the recursion only happens covariantly, which we formalize by requiring all F_d to be monotonic. Without this additional condition the rule for selection would be unsound, while the rules for object creation and selection have this condition built in their usual preconditions.

$$\forall d \in D.\, \forall\alpha, \beta \in \mathsf{Type}.\, \alpha \subseteq \beta \Rightarrow F_d(\alpha) \subseteq F_d(\beta)$$

Self object creation: $\text{pack } [m_d = \varsigma(x_d)b_d]_{d\in D}$

Selection: $a_\circ m_d \ \equiv\ \text{open } a \text{ as } x \text{ in } x.m_d$

Update: $a_\circ m_d := \varsigma(x)b \ \equiv\ \text{open } a \text{ as } x \text{ in pack } (x.m_d := \varsigma(x)b)$

Let $\alpha = \varsigma(\lambda\tau \in \mathsf{Type}.\, [m_d : F_d(\tau)]_{d\in D})$

$$(\textsc{SelfObj})\ \frac{\exists\tau \in \mathsf{Type}.\, \tau \subseteq \alpha \ \wedge\ \forall d \in D.\, \Gamma[x_d : \tau] \models b_d : F_d(\tau)}{\Gamma \models \text{pack } [m_d = \varsigma(x_d)b_d]_{d\in D} : \alpha}$$

$$(\textsc{SelfSel})\ \frac{\Gamma \models a : \alpha \quad e \in D}{\Gamma \models a_\circ m_e : F_e(\alpha)}$$

$$(\textsc{SelfUpd})\ \frac{\Gamma \models a : \alpha \quad e \in D \quad \forall\tau \in \mathsf{Type}.\, \tau \subseteq \alpha \Rightarrow \Gamma[x : [m_d : F_d(\tau)]_{d\in D}] \models b : F_e(\tau)}{\Gamma \models a_\circ m_e := \varsigma(x)b : \alpha}$$

$$(\textsc{SubSelfObj})\ \frac{E \subseteq D}{\varsigma(\lambda\tau \in \mathsf{Type}.\, [m_d : F_d(\tau)]_{d\in D}) \subseteq \varsigma(\lambda\tau \in \mathsf{Type}.\, [m_e : F_e(\tau)]_{e\in E})}$$

[8]Please note that if we would replace ς with μ in the subtyping rule for self types, the resulting rule would be unsound.

Variance Annotations [9]

In order to have subtyping in depth for objects, method selection and update can be tracked using object types with variance annotations: $\nu \in \{0, +, -\}$. Methods annotated with zero can be both selected and updated, but as before they need to be *invariant* when subtyping. Methods annotated with plus can only be selected but not updated, so they are *covariant* when subtyping. Similarly, methods annotated with minus can only be updated, and are thus *contravariant*. These annotations extend the usual object types, which can now be seen as a special case where all methods are invariant[10].

Let $\alpha = [m_d :_{\nu_d} \tau_d]_{d \in D}$

$$\begin{aligned}\alpha =_{def} \{\langle k, v\rangle \mid v = [m_e = \varsigma(x_e)b_e]_{e\in E}, D \subseteq E,\\ \forall d \in D.\, \forall j < k.\, ((\nu_d \in \{+, 0\} \Rightarrow [x_d \mapsto v](b_d) :_j \tau_d)\\ \wedge\ (\nu_d \in \{-, 0\} \Rightarrow \forall \varsigma(x)b.\, \langle j, \varsigma(x)b\rangle \in \alpha \rightarrow \tau_d\\ \Rightarrow \langle j, [m_d = \varsigma(x)b, m_e = \varsigma(x_e)b_e]_{e\in E\setminus\{d\}}\rangle \in \alpha))\}\end{aligned}$$

$$(\textsc{VarObj})\ \frac{\forall d \in D.\, \Gamma[x : \alpha] \models b_d : \tau_d}{\Gamma \models [m_d = \varsigma(x_d)b_d]_{d\in D} : \alpha}$$

$$(\textsc{VarSel})\ \frac{\Gamma \models a : \alpha \quad e \in D \quad \nu_e \in \{+, 0\}}{\Gamma \models a.m_e : \tau_e}$$

$$(\textsc{VarUpd})\ \frac{\Gamma \models a : \alpha \quad e \in D \quad \nu_e \in \{-, 0\} \quad \Gamma[x : \alpha] \models b : \tau_e}{\Gamma \models a.m_e := \varsigma(x)b : \alpha}$$

$$(\textsc{VarSubObj})\ \frac{\begin{array}{r} E \subseteq D \quad \forall e \in E.\, (\nu_e \in \{+, 0\} \Rightarrow \alpha_e \subseteq \beta_e)\\ \wedge\ (\nu_e \in \{-, 0\} \Rightarrow \beta_e \subseteq \alpha_e)\end{array}}{[m_d :_{\nu_d} \alpha_d]_{d\in D} \subseteq [m_e :_{\nu_e} \beta_e]_{e\in E}}$$

[9] For now only for the functional calculus since we did not find a way to describe "write-only" references

[10] We could even go one step further and define object types with variance annotations from the start, and then allowing the invariance annotation to be omitted for the same effect as our object types without annotations. We would only have to change the (Sem-SObj), (Sem-Sel) and (Sem-Upd) rules with the ones in this section, and also the definition of self types and the (Sem-SelfSObj) rule. Actually this would make perfect sense, since it would remove a couple of useless rules and proofs.

Misc

Semantic vs. Syntactic Typing Rules

$$(\textsc{Pack})\ \frac{\exists \tau \in \mathsf{Type}.\ \tau \subseteq \alpha\ \wedge\ \Gamma \models a : F(\tau)}{\Gamma \models \text{pack } a : \exists_\alpha F}$$

$$(\textsc{Syn-Pack})\ \frac{\Psi, \Gamma \vdash C \leq A \quad \Psi, \Gamma \vdash a[X := C] : B[X := C]}{\Psi, \Gamma \vdash \text{pack}_{\ X \leq A = C}\ a : \exists X \leq A.\, B}$$

Example of what it means to belong to a type

$[f = \varsigma(x)[]] :_1 [f : \mathit{int}]$