

Atomic Types	T	$::=$	$A \rightarrow B$ \perp $\forall \alpha * B. A$	Function type Bottom type Universal quantification
Types	A, B, C, D	$::=$	α $A \cap B$ T	Type variable Intersection type Atomic type
Expressions	e	$::=$	x $\lambda(x:A). e$ $e_1 e_2$ $\Lambda \alpha * A. e$ $e A$ e_1, e_2	Variable Lambda Application Big lambda Type application Merge
Contexts	Γ	$::=$	ϵ $\Gamma, \alpha * A$ $\Gamma, x:A$	
Sugar	$\Lambda \alpha. e$	$::=$	$\Lambda \alpha * \perp. e$	

Figure 1. Syntax.

$A <: B \hookrightarrow F$	
$\alpha <: \alpha \hookrightarrow \lambda(x: \alpha). x$	SUBVAR
$\tau_1 \rightarrow \tau_2 <: \tau_3 \rightarrow \tau_4 \hookrightarrow \lambda(f: \tau_1 \rightarrow \tau_2). \lambda(x: \tau_3). C_2 (f (C_1 x))$	SUBFUN
$\forall \alpha_1 * \tau_3. \tau_1 <: \forall \alpha_2 * \tau_3. \tau_2 \hookrightarrow \lambda(f: \forall \alpha_1 * \tau_3. \tau_1). \Lambda \alpha. C (f \alpha)$	SUBFORALL
$\tau_1 <: \tau_2 \hookrightarrow C_1 \quad \tau_1 <: \tau_3 \hookrightarrow C_2$	SUBAND
$\tau_1 <: \tau_2 \cap \tau_3 \hookrightarrow \lambda(x: \tau_1). (C_1 x, C_2 x)$	
$\tau_1 <: \tau_3 \hookrightarrow C$	SUBAND ₁
$\tau_1 \cap \tau_2 <: \tau_3 \hookrightarrow \lambda(x: \tau_1 \cap \tau_2). C (\text{proj}_1 x)$	
$\tau_2 <: \tau_3 \hookrightarrow C$	SUBAND ₂
$\tau_1 \cap \tau_2 <: \tau_3 \hookrightarrow \lambda(x: \tau_1 \cap \tau_2). C (\text{proj}_2 x)$	

1

Figure 2. Subtyping.

0.1 “Testsuite” of examples

1. $\lambda(x : \text{Int} * \text{Int}). (\lambda(z : \text{Int}). z) x$: This example should not type-check because it leads to an ambiguous choice in the body of the lambda. In the current system the well-formedness checks forbid such example.
2. $\Lambda A. \Lambda B. \lambda(x : A). \lambda(y : B). (\lambda(z : A). z)(x, y)$: This example should not type-check because it is not guaranteed that the instantiation of A and B produces a well-formed type. The TyMerge rule forbids it with the disjointness check.

$\Gamma \vdash A \perp B$	
$\frac{\alpha * B \in \Gamma}{\Gamma \vdash \alpha \perp B}$	DISJOINTREFL
$\frac{\alpha * A \in \Gamma}{\Gamma \vdash A \perp \alpha}$	DISJOINTSYM
$\frac{\Gamma \vdash A \perp C \quad \Gamma \vdash B \perp C}{\Gamma \vdash A \& B \perp C}$	DISJOINTSUB1
$\frac{\Gamma \vdash A \perp B \quad \Gamma \vdash A \perp C}{\Gamma \vdash A \perp B \& C}$	DISJOINTSUB2
$\frac{T_1 \not<: T_2 \quad T_2 \not<: T_1}{\Gamma \vdash T_1 \perp T_2}$	DISJOINTATOMIC

Figure 3. Disjointness.

$\Gamma \vdash A \text{ type}$	
$\Gamma \vdash \perp \text{ type}$	WFBOT
$\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash A \rightarrow B \text{ type}}$	WFFUN
$\frac{\Gamma, \alpha * B \vdash A \text{ type}}{\Gamma \vdash \forall \alpha * B. A \text{ type}}$	WFFORALL
$\frac{\alpha * A \in \Gamma}{\Gamma \vdash \alpha \text{ type}}$	WFVAR
$\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type} \quad \Gamma \vdash A \perp B}{\Gamma \vdash A \cap B \text{ type}}$	WFINTER

Figure 4. Well-formed types.

$\Gamma \vdash e : A \hookrightarrow E$	
$\frac{x:A \in \Gamma}{\Gamma \vdash x : A \hookrightarrow x}$	TYVAR
$\frac{\Gamma \vdash A \text{ type} \quad \Gamma, x:A \vdash e : B \hookrightarrow E}{\Gamma \vdash \lambda(x:A). e : A \rightarrow B \hookrightarrow \lambda(x: A). E}$	TYLAM
$\frac{\Gamma \vdash e_1 : A_1 \rightarrow A_2 \hookrightarrow E_1 \quad \Gamma \vdash e_2 : A_3 \hookrightarrow E_2 \quad A_3 <: A_1 \hookrightarrow C}{\Gamma \vdash e_1 e_2 : A_2 \hookrightarrow E_1 (C E_2)}$	TYAPP
$\frac{\Gamma, \alpha * B \vdash e : A \hookrightarrow E \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash \Lambda \alpha * B. e : \forall \alpha * B. A \hookrightarrow \Lambda \alpha. E}$	TYBLAM

$\frac{\Gamma \vdash e : \forall \alpha * C. B \hookrightarrow E \quad \Gamma \vdash A \perp C \quad \Gamma \vdash A \text{ type}}{\Gamma \vdash e A : [A/\alpha]B \hookrightarrow E [A]}$	TYTAPP
$\frac{\Gamma \vdash e_1 : A \hookrightarrow E_1 \quad \Gamma \vdash e_2 : B \hookrightarrow E_2 \quad \Gamma \vdash A \perp B}{\Gamma \vdash e_1, e_2 : A \cap B \hookrightarrow (E_1, E_2)}$	TYMERGE

Figure 5. Typing.

3. $\Lambda A. \Lambda B * A. \lambda(x : A). \lambda(y : B). (\lambda(z : A). z)(x, y)$: This example should type-check because B is guaranteed to be disjoint with A. Therefore instantiation should produce a well-formed type.
4. $(\lambda(z : \text{Int}). z)((1, 'c'), (2, \text{False}))$: This example should not type-check, since it leads to an ambiguous lookup of integers (can either be 1 or 2). The definition of disjointness is crucial to prevent this example from type-checking. When type-checking the large merge, the disjointness predicate will detect that more than one integer exists in the merge.

$$e \vdash 1, 2 : (\text{Int} * \text{Int}) \Rightarrow \text{Int} \cap \text{Int}$$

Definition 1. (Disjointness) Two sets S and T are *disjoint* if there does not exist an element x , such that $x \in S$ and $x \in T$.

Definition 2. (Disjointness) Two types A and B are *disjoint* if there does not exist an expression e , which is not a merge, such that $e \vdash e : A', e \vdash e : B', A' <: A$, and $B' <: B$.

Definition 3. (Disjointness) $A \perp B = \exists C. A <: C \wedge B <: C$

Two types A and B are *disjoint* if their least common supertype is \top .