Clam semantics

Maxime Mulder

Introduction

This document is the work-in-progress formalization of the Clam programming language. It covers the semantics of the core calculus used by Clam, which is based on System $F^{\omega}_{<:}$ enriched with a few basic data types, unions and intersections.

This document is not yet complete. For now, it only specifies declarative rules and lacks associated theorems and proofs.

This document has been written using Typst.

Contents

Grammar

The abstract syntax of Clam is given by the following grammar:

$u ::= \texttt{unit} \qquad b ::= \texttt{true} \mid \texttt{false} \qquad n ::= 0 \mid 1 \mid \ldots \qquad s ::= \text{"}\ldots\text{"}$

Expression $e ::=$	u	unit
	$\mid b$	boolean
	$\mid n$	integer
	$\mid s$	string
	$\mid x$	variable
	$\mid \langle e_1, \ldots, e_n \rangle$	tuple
	$\mid e.n$	tuple projection
	$\mid \langle l_1 = e_1, \ldots, l_n = e_n \rangle$	record
	$\mid e.l_n$	record projection
	$\mid \lambda x : \tau.\ e$	lambda abstraction
	$\mid e(e)$	lambda application
	$\mid \lambda t : \tau\ ..\ \tau.\ e$	universal abstraction
	$\mid e[\tau]$	universal application
	$\mid e : \tau$	type ascription
Type $\tau ::=$	$\top$	top
	$\mid \bot$	bottom
	$\mid \tau \cup \tau$	union
	$\mid \tau \cap \tau$	intersection
	$\mid \texttt{Unit}$	unit
	$\mid \texttt{Bool}$	boolean
	$\mid \texttt{Int}$	integer
	$\mid \texttt{String}$	string
	$\mid t$	variable
	$\mid \langle \tau_1, \ldots, \tau_n \rangle$	tuple
	$\mid \langle l_1 : \tau_1, \ldots, l_n : \tau_n \rangle$	record
	$\mid \tau \to \tau$	lambda abstraction
	$\mid \forall t : \tau\ ..\ \tau.\ \tau$	universal abstraction
	$\mid \Lambda t : \tau\ ..\ \tau.\ \tau$	type abstraction
	$\mid \tau[\tau]$	type application

Kinding

The syntax of kinds is given by the following grammar:

$$\begin{array}{rll} \text{Kind } k ::= & * & \text{type} \\ | & \tau \mathrel{..} \tau \to k & \text{arrow} \end{array}$$

The syntax of the typing context Δ is given by the following grammar:

$$\Delta ::= \emptyset \mid \Delta, t : \tau \mathrel{..} \tau$$

$$\frac{}{Q(\emptyset)}\,\text{Q-Nil} \qquad \frac{Q(\Delta) \quad t \notin \Delta \quad \Delta \vdash \tau_1 \leq \tau_2}{Q(\Delta, t : \tau_1 \mathrel{..} \tau_2)}\,\text{Q-Cons}$$

The kinding judgement is of the form $\Delta \vdash \tau :: k$. The hypotheses of Δ are of the form $t : \tau_1 \mathrel{..} \tau_2$, where τ_1 and τ_2 represent respectively the lower and the upper bounds of a type variable.

$$\frac{}{\texttt{Unit} :: *}\,\text{K-Unit} \qquad \frac{}{\texttt{Bool} :: *}\,\text{K-Bool} \qquad \frac{}{\texttt{Int} :: *}\,\text{K-Int} \qquad \frac{}{\texttt{String} :: *}\,\text{K-String}$$

$$\frac{}{\top :: *}\,\text{K-Top} \qquad \frac{}{\bot :: *}\,\text{K-Bot} \qquad \frac{t : \tau_1 \mathrel{..} \tau_2 \in \Delta \quad \Delta \vdash \tau_1 :: k}{\Delta \vdash t :: k}\,\text{K-Var}$$

$$\frac{\Delta \vdash \tau_1 :: * \quad \ldots \quad \Delta \vdash \tau_n :: *}{\Delta \vdash \langle \tau_1, \ldots, \tau_n \rangle :: *}\,\text{K-Tuple} \qquad \frac{\Delta \vdash \tau_1 :: * \quad \ldots \quad \Delta \vdash \tau_n :: *}{\Delta \vdash \langle l_1 : \tau_1, \ldots, l_n : \tau_n \rangle :: *}\,\text{K-Record}$$

$$\frac{\Delta \vdash \tau_1 :: * \quad \Delta \vdash \tau_2 :: *}{\Delta \vdash \tau_1 \to \tau_2 :: *}\,\text{K-Lam} \qquad \frac{\Delta \vdash \tau_1 \leq \tau_2 \quad \Delta, t : \tau_1 \mathrel{..} \tau_2 \vdash \tau_3 :: *}{\Delta \vdash \forall t : \tau_1 \mathrel{..} \tau_2.\ \tau_3 :: *}\,\text{K-Univ}$$

$$\frac{\Delta \vdash \tau_1 \leq \tau_2 \quad \Delta, t : \tau_1 \mathrel{..} \tau_2 \vdash \tau_3 :: k}{\Delta \vdash \Lambda t : \tau_1 \mathrel{..} \tau_2.\ \tau_3 :: \tau_1 \mathrel{..} \tau_2 \to k}\,\text{K-Abs}$$

$$\frac{\Delta \vdash \tau_1 :: \tau_2 \mathrel{..} \tau_3 \to k \quad \Delta \vdash \tau_2 \leq \tau_4 \quad \Delta \vdash \tau_4 \leq \tau_3}{\Delta \vdash \tau_1[\tau_4] :: k}\,\text{K-App}$$

$$\frac{\Delta \vdash \tau_1 :: k \quad \Delta \vdash \tau_2 :: k}{\Delta \vdash \tau_1 \cup \tau_2 :: k}\,\text{K-Union} \qquad \frac{\Delta \vdash \tau_1 :: k \quad \Delta \vdash \tau_2 :: k}{\Delta \vdash \tau_1 \cap \tau_2 :: k}\,\text{K-Inter}$$

Rules K-Univ and K-Abs prohibit inconsistent bounds in universal types and type abstractions.

Type equivalence

The type equivalence judgement is of the form $\Delta \vdash \tau \equiv \tau'$. The following rules describe the reflexivity, symmetry and transitivity properties of the type equivalence relation.

$$\frac{}{\tau \equiv \tau}\ \textsc{E-Refl} \qquad \frac{\Delta \vdash \tau_1 \equiv \tau_2}{\Delta \vdash \tau_2 \equiv \tau_1}\ \textsc{E-Symm} \qquad \frac{\Delta \vdash \tau_1 \equiv \tau_2 \quad \Delta \vdash \tau_2 \equiv \tau_3}{\Delta \vdash \tau_1 \equiv \tau_3}\ \textsc{E-Trans}$$

The following rules describe type equivalence for composite data types.

$$\frac{\Delta \vdash \tau_1 \equiv \tau_1' \quad ... \quad \Delta \vdash \tau_n \equiv \tau_n'}{\Delta \vdash \langle \tau_1, ..., \tau_n \rangle \equiv \langle \tau_1', ..., \tau_n' \rangle}\ \textsc{E-Tuple}$$

$$\frac{\Delta \vdash \tau_1 \equiv \tau_1' \quad ... \quad \Delta \vdash \tau_n \equiv \tau_n'}{\Delta \vdash \langle l_1 : \tau_1, ..., l_n : \tau_n \rangle \equiv \langle l_1 : \tau_1', ..., l_n : \tau_n' \rangle}\ \textsc{E-Record}$$

$$\frac{\Delta \vdash \tau_1 \equiv \tau_1' \quad \Delta \vdash \tau_2 \equiv \tau_2'}{\Delta \vdash \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'}\ \textsc{E-Abs} \qquad \frac{\Delta \vdash \tau_1 \equiv \tau_1' \quad \Delta \vdash \tau_2 \equiv \tau_2' \quad \Delta \vdash \tau_3 \equiv \tau_3'}{\Delta \vdash \forall t : \tau_1 \mathrel{..} \tau_2.\ \tau_3 \equiv \forall t : \tau_1' \mathrel{..} \tau_2'.\ \tau_3'}\ \textsc{E-Univ}$$

$$\frac{\Delta \vdash \tau_1 \equiv \tau_1' \quad \Delta \vdash \tau_2 \equiv \tau_2' \quad \Delta \vdash \tau_3 \equiv \tau_3'}{\Delta \vdash \Lambda t : \tau_1 \mathrel{..} \tau_2.\ \tau_3 \equiv \Lambda t : \tau_1' \mathrel{..} \tau_2'.\ \tau_3'}\ \textsc{E-TAbs} \qquad \frac{\Delta \vdash \tau_1 \equiv \tau_1' \quad \Delta \vdash \tau_2 \equiv \tau_2'}{\Delta \vdash \tau_1[\tau_2] \equiv \tau_1'[\tau_2']}\ \textsc{E-TApp}$$

$$\frac{}{(\Lambda t : \tau_1 \mathrel{..} \tau_2.\ \tau_3)[\tau_4] \equiv [t/\tau_3]\tau_2}\ \textsc{E-TAbsApp}$$

The following rules describe the commutativity, associativity, distribution and inclusion properties of union and intersection types[1].

$$\frac{}{\tau_1 \cup \tau_2 \equiv \tau_2 \cup \tau_1}\ \textsc{E-UnionComm} \qquad \frac{}{\tau_1 \cap \tau_2 \equiv \tau_2 \cap \tau_1}\ \textsc{E-InterComm}$$

$$\frac{}{(\tau_1 \cup \tau_2) \cup \tau_3 \equiv \tau_1 \cup (\tau_2 \cup \tau_3)}\ \textsc{E-UnionAssoc}$$

$$\frac{}{(\tau_1 \cap \tau_2) \cap \tau_3 \equiv \tau_1 \cap (\tau_2 \cap \tau_3)}\ \textsc{E-InterAssoc}$$

$$\frac{}{\tau_1 \cup (\tau_2 \cap \tau_3) \equiv (\tau_1 \cup \tau_2) \cap (\tau_1 \cup \tau_3)}\ \textsc{E-UnionDistrib}$$

$$\frac{}{\tau_1 \cap (\tau_2 \cup \tau_3) \equiv (\tau_1 \cap \tau_2) \cup (\tau_1 \cap \tau_3)}\ \textsc{E-InterDistrib}$$

$$\frac{\Delta \vdash \tau_1 \leq \tau_2}{\Delta \vdash \tau_1 \cup \tau_2 \equiv \tau_2}\ \textsc{E-UnionIncl} \qquad \frac{\Delta \vdash \tau_1 \leq \tau_2}{\Delta \vdash \tau_1 \equiv \tau_1 \cap \tau_2}\ \textsc{E-InterIncl}$$

[1] What about the absorbtion law ?

The following rules describe the distributivity of intersection types over composite data types.

$$\frac{}{\langle \tau_1, ..., \tau_n \rangle \cap \langle \tau'_1, ..., \tau'_n \rangle \equiv \langle \tau_1 \cap \tau'_1, ..., \tau_n \cap \tau'_n \rangle}\ \textsc{E-MeetTuple}$$

$$\frac{}{\begin{array}{c}\langle l_1 : \tau_1, ..., l_n : \tau_n, l''_1 : \tau''_1, ..., l''_n : \tau''_n \rangle \cap \langle l_1 : \tau'_1, ..., l_n : \tau'_n, l'''_1 : \tau'''_1, ..., l'''_n : \tau'''_n \rangle \\ \equiv \\ \langle l_1 : \tau_1 \cap \tau'_1, ..., l_n : \tau_n \cap \tau'_n, l''_1 : \tau''_1, ..., l''_n : \tau''_n, l'''_1 : \tau'''_1, ..., l'''_n : \tau'''_n \rangle\end{array}}\ \textsc{E-MeetRecord}$$

$$\frac{}{(\tau_1 \to \tau_2) \cap (\tau'_1 \to \tau'_2) \equiv \tau_1 \cup \tau'_1 \to \tau_2 \cap \tau'_2}\ \textsc{E-MeetAbs}$$

$$\frac{\Delta \vdash \tau_1 \equiv \tau'_1 \qquad \Delta \vdash \tau_2 \equiv \tau'_2 \qquad \Delta \vdash \tau_3 \equiv \tau'_3}{\Delta \vdash (\forall t : \tau_1 \ .. \ \tau_2 .\ \tau_3) \cap (\forall t : \tau'_1 \ .. \ \tau'_2 .\ \tau'_3) \equiv \forall t : \tau_1 \ .. \ \tau_2 .\ \tau_3 \cap \tau'_3}\ \textsc{E-MeetUniv}$$

$$\frac{\Delta \vdash \tau_1 \equiv \tau'_1 \qquad \Delta \vdash \tau_2 \equiv \tau'_2 \qquad \Delta \vdash \tau_3 \equiv \tau'_3}{\Delta \vdash (\Lambda t : \tau_1 \ .. \ \tau_2 .\ \tau_3) \cap (\Lambda t : \tau'_1 \ .. \ \tau'_2 .\ \tau'_3) \equiv \Lambda t : \tau_1 \ .. \ \tau_2 .\ \tau_3 \cap \tau'_3}\ \textsc{E-MeetTAbs}$$

TODO: Check meet rules.

Subtyping

The subtyping judgement is of the form $\Delta \vdash \tau \leq \tau'$.

$$\frac{\Delta \vdash \tau_1 \equiv \tau_2}{\Delta \vdash \tau_1 \leq \tau_2}\textsc{S-Eq} \qquad \frac{\Delta \vdash \tau_1 \leq \tau_2 \quad \Delta \vdash \tau_2 \leq \tau_3}{\Delta \vdash \tau_1 \leq \tau_3}\textsc{S-Trans}$$

$$\frac{t : \tau_1 \,..\, \tau_2 \in \Delta}{\Delta \vdash \tau_1 \leq t}\textsc{S-VarSub} \qquad \frac{t : \tau_1 \,..\, \tau_2 \in \Delta}{\Delta \vdash t \leq \tau_2}\textsc{S-VarSup}$$

$$\frac{\Delta \vdash \tau :: *}{\Delta \vdash \tau \leq \top}\textsc{S-Top} \qquad \frac{\Delta \vdash \tau :: *}{\Delta \vdash \bot \leq \tau}\textsc{S-Bot}$$

$$\frac{\Delta \vdash \tau_1 \leq \tau_1' \quad ... \quad \Delta \vdash \tau_n \leq \tau_n'}{\Delta \vdash \langle \tau_1, ..., \tau_n \rangle \leq \langle \tau_1', ..., \tau_n' \rangle}\textsc{S-Tuple}$$

$$\frac{\Delta \vdash \tau_1 \leq \tau_1' \quad ... \quad \Delta \vdash \tau_n \leq \tau_n' \qquad n \leqslant m}{\Delta \vdash \langle l_1 : \tau_1, ..., l_m : \tau_m \rangle \leq \langle l_1 : \tau_1', ..., l_n : \tau_n' \rangle}\textsc{S-Record}$$

$$\frac{\Delta \vdash \tau_1' \leq \tau_1 \quad \Delta \vdash \tau_2 \leq \tau_2'}{\Delta \vdash \tau_1 \to \tau_2 <: \tau_1' \to \tau_2'}\textsc{S-Abs}$$

$$\frac{\Delta \vdash \tau_1 \leq \tau \quad \Delta \vdash \tau \leq \tau_2 \quad \Delta \vdash [t/\tau]\tau_3 \leq \tau_4}{\Delta \vdash \forall t : \tau_1 \,..\, \tau_2.\ \tau_3 \leq \tau_4}\textsc{S-UnivL} \qquad \frac{\Delta, t : \tau_1 \,..\, \tau_2 \vdash \tau \leq \tau_3}{\Delta \vdash \tau \leq \forall t : \tau_1 \,..\, \tau_2.\ \tau_3}\textsc{S-UnivR}$$

$$\frac{\Delta \vdash \tau_1 \,..\, \tau_2 \equiv \tau_1' \,..\, \tau_2' \quad \Delta \vdash \tau_3 \leq \tau_3'}{\Delta \vdash \Lambda t : \tau_1 \,..\, \tau_2.\ \tau_3 \leq \Lambda t : \tau_1' \,..\, \tau_2'.\ \tau_3'}\textsc{S-TAbs} \qquad \frac{\Delta \vdash \tau_1 \leq \tau_1' \quad \Delta \vdash \tau_2 \equiv \tau_2'}{\Delta \vdash \tau_1[\tau_2] \leq \tau_1'[\tau_2']}\textsc{S-TApp}$$

$$\frac{\Delta \vdash \tau \leq \tau_1}{\Delta \vdash \tau \leq \tau_1 \cup \tau_2}\textsc{S-Union1} \qquad \frac{\Delta \vdash \tau_1 \leq \tau}{\Delta \vdash \tau_1 \cap \tau_2 \leq \tau}\textsc{S-Inter2}$$

$$\frac{\Delta \vdash \tau_1 \leq \tau \quad \Delta \vdash \tau_2 \leq \tau}{\Delta \vdash \tau_1 \cup \tau_2 \leq \tau}\textsc{S-Union1} \qquad \frac{\Delta \vdash \tau \leq \tau_1 \quad \Delta \vdash \tau \leq \tau_2}{\Delta \vdash \tau \leq \tau_1 \cap \tau_2}\textsc{S-Inter2}$$

Rules S-UnivL and S-UnivR correspond to the higher-rank polymrohism subtyping relation by Odersky and Läufer [1].

Typing

The typing judgement is of the form $\Delta\ \Gamma \vdash e : \tau$. The hypotheses of Γ are of the form $x : \tau$.

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \textsc{T-Var} \qquad \frac{\Delta\ \Gamma \vdash e : \tau \quad \Delta \vdash \tau \leq \tau'}{\Delta\ \Gamma \vdash e : \tau'}\ \textsc{T-Sub}$$

$$\frac{}{u : \texttt{Unit}}\ \textsc{T-Unit} \qquad \frac{}{b : \texttt{Bool}}\ \textsc{T-Bool} \qquad \frac{}{i : \texttt{Int}}\ \textsc{T-Int} \qquad \frac{}{s : \texttt{String}}\ \textsc{T-String}$$

$$\frac{\Delta\ \Gamma \vdash e_1 : \tau_1 \quad ... \quad \Delta\ \Gamma \vdash e_n : \tau_n}{\Delta\ \Gamma \vdash \langle e_1, ... e_n \rangle : \langle \tau_1, ..., \tau_n \rangle}\ \textsc{T-Tuple} \qquad \frac{\Delta\ \Gamma \vdash e : \langle \tau_1, ..., \tau_n \rangle}{\Delta\ \Gamma \vdash e.i : \tau_i}\ \textsc{T-TupleProj}$$

$$\frac{\Delta\ \Gamma \vdash e_1 : \tau_1 \quad ... \quad \Delta\ \Gamma \vdash e_n : \tau_n}{\Delta\ \Gamma \vdash \langle l_1 = e_1, ..., l_n = e_n \rangle : \langle l_1 : \tau_1, ..., l_n : \tau_n \rangle}\ \textsc{T-Record}$$

$$\frac{\Delta\ \Gamma \vdash e : \langle l_1 : \tau_1, ..., l_n : \tau_n \rangle}{\Delta\ \Gamma \vdash e.l_i : \tau_i}\ \textsc{T-RecordProj}$$

$$\frac{\Delta \vdash \tau_1 :: * \quad \Delta\ \Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta\ \Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2}\ \textsc{T-Abs} \qquad \frac{\Delta\ \Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Delta\ \Gamma \vdash e_2 : \tau_1}{\Delta\ \Gamma \vdash e_1(e_2) : \tau_2}\ \textsc{T-AbsApp}$$

$$\frac{\Delta \vdash \tau_1 \leq \tau_2 \quad \Delta, t : \tau_1\ ..\ \tau_2\ \Gamma \vdash e : \tau_3}{\Delta\ \Gamma \vdash \lambda t : \tau_1\ ..\ \tau_2.\ e : \forall t : \tau_1\ ..\ \tau_2.\ \tau_3}\ \textsc{T-Univ}$$

$$\frac{\Delta\ \Gamma \vdash e : \forall t : \tau_1\ ..\ \tau_2.\ \tau_3 \quad \Delta \vdash \tau_1 \leq \tau \quad \Delta \vdash \tau \leq \tau_2}{\Delta\ \Gamma \vdash e[\tau_3] : [t/\tau]\tau_3}\ \textsc{T-UnivApp}$$

$$\frac{\Delta \vdash \tau :: * \quad \Delta\ \Gamma \vdash e : \tau}{\Delta\ \Gamma \vdash e : \tau : \tau}\ \textsc{T-Ascr}$$

TODO: Fix T-UnivApp.

Evaluation

The syntax of values is given by the following grammar:

Value $v ::= u$	unit
$\mid b$	boolean
$\mid i$	integer
$\mid s$	string
$\mid \langle v_1, ..., v_n \rangle$	tuple
$\mid \langle l_1 = v_1, ..., l_n = v_n \rangle$	record
$\mid \lambda x.\ e$	lambda abstraction

The evaluation judgement is of the form $e \Downarrow v$.

$$\frac{}{u \Downarrow u}\ \textsc{V-Unit} \qquad \frac{}{b \Downarrow b}\ \textsc{V-Bool} \qquad \frac{n = i}{n \Downarrow i}\ \textsc{V-Int} \qquad \frac{}{s \Downarrow s}\ \textsc{V-String}$$

$$\frac{e_1 \Downarrow v_1 \quad ... \quad e_n \Downarrow v_n}{\langle e_1, ...e_n \rangle \Downarrow \langle v_1, ..., v_n \rangle}\ \textsc{V-Tuple} \qquad \frac{e \Downarrow \langle v_1, ..., v_n \rangle}{e.i \Downarrow v_i}\ \textsc{V-TupleProj}$$

$$\frac{e_1 \Downarrow v_1 \quad ... \quad e_n \Downarrow v_n}{\langle l_1 = e_1, ..., l_n = e_n \rangle \Downarrow \langle l_1 = v_1, ..., l_n = v_n \rangle}\ \textsc{V-Record}$$

$$\frac{e \Downarrow \langle l_1 = v_1, ..., l_n = v_n \rangle}{e.l_i \Downarrow v_i}\ \textsc{V-RecordProj}$$

$$\frac{}{\lambda x : \tau.\ e \Downarrow \lambda x.\ e}\ \textsc{V-Abs} \qquad \frac{e_1 \Downarrow \lambda x.\ e_3 \quad [e_2/x]e_3 \Downarrow v}{e_1(e_2) \Downarrow v}\ \textsc{V-AbsApp}$$

$$\frac{e \Downarrow v}{\lambda t : \tau_1\ ..\ \tau_2.\ e \Downarrow v}\ \textsc{V-Univ} \qquad \frac{e \Downarrow v}{e[\tau] \Downarrow v}\ \textsc{V-UnivApp}$$

$$\frac{e \Downarrow v}{e : \tau \Downarrow v}\ \textsc{V-Ascr}$$

Primitives

The types of the primitive values of Clam are given by the following table:

Primitive	**Type**
+ (unary)	`Int` $\rightarrow$ `Int`
- (unary)	`Int` $\rightarrow$ `Int`
`!`	`Bool` $\rightarrow$ `Bool`
+ (binary)	`Int` $\rightarrow$ `Int` $\rightarrow$ `Int`
- (binary)	`Int` $\rightarrow$ `Int` $\rightarrow$ `Int`
`*`	`Int` $\rightarrow$ `Int` $\rightarrow$ `Int`
`/`	`Int` $\rightarrow$ `Int` $\rightarrow$ `Int`
`%`	`Int` $\rightarrow$ `Int` $\rightarrow$ `Int`
`++`	`String` $\rightarrow$ `String` $\rightarrow$ `String`
`==`	$\top \rightarrow \top \rightarrow$ `Bool`
`!=`	$\top \rightarrow \top \rightarrow$ `Bool`
`<`	`Int` $\rightarrow$ `Int` $\rightarrow$ `Bool`
`>`	`Int` $\rightarrow$ `Int` $\rightarrow$ `Bool`
`<=`	`Int` $\rightarrow$ `Int` $\rightarrow$ `Bool`
`>=`	`Int` $\rightarrow$ `Int` $\rightarrow$ `Bool`
`|`	`Bool` $\rightarrow$ `Bool` $\rightarrow$ `Bool`
`&`	`Bool` $\rightarrow$ `Bool` $\rightarrow$ `Bool`
`if`[2]	$\forall t.$ `Bool` $\rightarrow t \rightarrow t \rightarrow t$

Bibliography

[1] M. Odersky and K. Läufer, "Putting type annotations to work," in *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, in POPL '96. St. Petersburg Beach, Florida, USA: Association for Computing Machinery, 1996, pp. 54–67. doi: 10.1145/237721.237729.

[2] `if` is not implemented as a primitive value yet.