Typed Lambda Calculus

"There may, indeed, be other applications of the system other than its use as a logic", Alonzo Church, 1932

January 19, 2019

What is the Lambda Calculus?

- Model of computation based on functions
  - gives rise to functional programming
- Introduced by Alonzo Church in 1934
- Turing complete
- Considered a faithful model of general purpose PLs
  - The next 700 programming languages, Peter Landin, 1966
  - Motto: Use LC as a testbed to embed (DSLs) new PL features

Why the Lambda Calculus?

- In an industrial-strength language is it difficult to determine with precision/rigor
  - basic properties on semantics and types
  - effect of adding new program constructions
  - relation with other languages/paradigms
- Convenient to restrict the language to a subset that is
  - representative of the paradigm/problem area
  - concise
  - reduced number of primitives
  - rigorous in its formulation

Type Expressions of $\lambda_V^{\mathbb{B},\to}$

The type expressions (or simply types) of $\lambda_V^{\mathbb{B},\to}$ are:

$$\sigma, \tau \quad ::= \quad \mathbb{B} \,|\, \sigma \to \tau$$

Informally:

- $\mathbb{B}$ is type of booleans
- $\sigma \to \tau$ is type of functions from σ to τ

Terms of $\lambda_V^{\mathbb{B},\to}$

Let $\mathcal{X}$ be a denumerable set of variables and $x \in \mathcal{X}$. The set of terms of $\lambda_V^{\mathbb{B},\to}$ are:

$$
\begin{array}{lcll}
M, N, P, Q & ::= & x & \text{term variable} \\
& \mid & \mathit{true} & \\
& \mid & \mathit{false} & \\
& \mid & \mathit{if}\ \ M\ \ \mathit{then}\ \ P\ \ \mathit{else}\ \ Q & \\
& \mid & \lambda x : \sigma . M & \text{anonymous function;} \\
& & & \ x \text{ is binding variable} \\
& \mid & M\,N & \text{application}
\end{array}
$$

Examples

- $\lambda x : \mathbb{B}.x$
- $\lambda x : \mathbb{B}.\mathit{if}\ \ x\ \ \mathit{then}\ \ \mathit{false}\ \ \mathit{else}\ \ \mathit{true}$
- $\lambda f : \sigma \to \tau.\lambda x : \sigma.f\,x$
- $(\lambda f : \mathbb{B} \to \mathbb{B}.f\ \mathit{true})(\lambda y : \mathbb{B}.y)$
- $x\,y$

Free Variables

A variable can occur free or bound in a term.

- $\lambda x : \mathbb{B}.\mathit{if}\ \underbrace{x}_{\mathit{bound}}\ \mathit{then}\ \mathit{true}\ \mathit{else}\ \mathit{false}$
- $\lambda x : \mathbb{B}.\lambda y : \mathbb{B}.\mathit{if}\ \mathit{true}\ \mathit{then}\ \underbrace{x}_{\mathit{bound}}\ \mathit{else}\ \underbrace{y}_{\mathit{bound}}$
- $\lambda x : \mathbb{B}.\mathit{if}\ \underbrace{x}_{\mathit{bound}}\ \mathit{then}\ \mathit{true}\ \mathit{else}\ \underbrace{y}_{\mathit{free}}$
- $(\lambda x : \mathbb{B}.\mathit{if}\ \underbrace{x}_{\mathit{bound}}\ \mathit{then}\ \mathit{true}\ \mathit{else}\ \mathit{false})\ \underbrace{x}_{\mathit{free}}$

Free Variables

$$
\begin{aligned}
FV(x) &\stackrel{\text{def}}{=} \{x\} \\
FV(\mathit{true}) = FV(\mathit{false}) &\stackrel{\text{def}}{=} \emptyset \\
FV(\mathit{if}\ \ M\ \ \mathit{then}\ P\ \mathit{else}\ \ Q) &\stackrel{\text{def}}{=} FV(M) \cup FV(P) \cup FV(Q) \\
FV(M\,N) &\stackrel{\text{def}}{=} FV(M) \cup FV(N) \\
FV(\lambda x : \sigma.M) &\stackrel{\text{def}}{=} FV(M) \setminus \{x\}
\end{aligned}
$$

$$
\begin{aligned}
BV(x) &\stackrel{\text{def}}{=} \emptyset \\
BV(\mathit{true}) = FV(\mathit{false}) &\stackrel{\text{def}}{=} \emptyset \\
BV(\mathit{if}\ \ M\ \ \mathit{then}\ P\ \mathit{else}\ \ Q) &\stackrel{\text{def}}{=} BV(M) \cup BV(P) \cup BV(Q) \\
BV(M\,N) &\stackrel{\text{def}}{=} BV(M) \cup BV(N) \\
BV(\lambda x : \sigma.M) &\stackrel{\text{def}}{=} BV(M) \cup (\{x\} \cap FV(M))
\end{aligned}
$$

Renaming

$$M^x_y$$

Replace every free occurrence of x with y

Examples

- $(x\,y)^x_z = z\,y$
- $(\lambda y.x)^x_z = \lambda y.z$
- $(\lambda y.x)^x_y = \lambda y.y$ Wrong??
  - Renaming can capture variables
  - y is typically required not to be bound in M

α-Equivalence

$$\frac{y \notin FV(M) \text{ and } y \text{ not a binding variable in } M}{\lambda x.M =_\alpha \lambda y.M^x_y}$$

$$\frac{M =_\alpha M'}{\lambda x.M =_\alpha \lambda x.M'} \qquad \frac{M =_\alpha M' \quad P =_\alpha P'}{M\,P =_\alpha M'\,P'}$$

$$\frac{M =_\alpha M' \quad P =_\alpha P' \quad Q =_\alpha Q'}{\mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ Q =_\alpha \mathit{if}\ M'\ \mathit{then}\ P'\ \mathit{else}\ Q'}$$

- Uses renaming as a tool
- When we write rules like this we mean the smallest congruence generated by them

α-Equivalence

Examples:

- $\lambda x.x =_\alpha \lambda y.y$
- $\lambda x.y =_\alpha \lambda z.y$
- $\lambda x.y \neq_\alpha \lambda x.z$
- $\lambda x.\lambda x.x \neq_\alpha \lambda y.\lambda x.y$

Terms that differ only in the names of bound variables are considered identical

Substitution

$M\{x := N\}$ "*Substitute all free occurrences of x in M with N*"

$$
\begin{array}{rcl}
x\{x := N\} & \stackrel{\text{def}}{=} & N \\
y\{x := N\} & \stackrel{\text{def}}{=} & y, \quad x \neq y \\
a\{x := N\} & \stackrel{\text{def}}{=} & a \quad \text{si } a \in \{\mathit{true}, \mathit{false}\} \\
(\mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ Q)\{x := N\} & \stackrel{\text{def}}{=} & \mathit{if}\ M\{x := N\}\ \mathit{then}\ P\{x := N\} \\
 & & \qquad\qquad \mathit{else}\ Q\{x := N\} \\
(M_1\, M_2)\{x := N\} & \stackrel{\text{def}}{=} & M_1\{x := N\}\, M_2\{x := N\} \\
(\lambda y : \sigma.M)\{x := N\} & \stackrel{\text{def}}{=} & \lambda y : \sigma.M\{x := N\}\ x \neq y,\ y \notin FV(N)
\end{array}
$$

1. Note: condition $x \neq y,\ y \notin FV(N)$ can always be met by renaming
2. Technically, subst. is defined on α-equivalence classes of terms

Variable Capture

- Condition $x \neq y, \ y \notin FV(N)$ is important in this clause:

$$(\lambda y : \sigma.M)\{x := N\} \quad \stackrel{\text{def}}{=} \quad \lambda y : \sigma.M\{x := N\} \ x \neq y, \ y \notin FV(N)$$

- It avoids examples such as this:

$$(\lambda z : \sigma.x)\{x := z\} = \lambda z : \sigma.z$$

  - Here z has been captured by a binder

Type System

- Formal deductive system that uses axioms and inference schemes to characterize a subset of the terms
- First we need to introduce typing contexts
  - What is the type of *true*?
  - What is the type of *if x then true else false*?

Type System

A typing context is a set of pairs $x_i : \sigma_i$, written $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ where the $\{x_i\}_{i \in 1..n}$ are distinct. We use letters $\Gamma, \Delta, \ldots$ for typing contexts.

A typing judgement is an expression of the form $\Gamma \triangleright M : \sigma$, read:

"*term M has type σ under typing context Γ*"

Type System

- The meaning of $\Gamma \rhd M : \sigma$ is established through typing axioms and inference schemes/rules.
- If $\Gamma \rhd M : \sigma$ can be derived using the typing axioms and inference schemes then we say that it is derivable.
- We say that M is typable if the typing judgement $\Gamma \rhd M : \sigma$ can be derived, for some Γ and σ.
- We next present the typing axioms and inference schemes for $\lambda_V^{\mathbb{B},\to}$

Axioms and Inference Schemes

$$\frac{x : \sigma \in \Gamma}{\Gamma \rhd x : \sigma}\ (\textsc{T-Var})$$

$$\frac{}{\Gamma \rhd \mathit{true} : \mathbb{B}}\ (\textsc{T-True}) \qquad \frac{}{\Gamma \rhd \mathit{false} : \mathbb{B}}\ (\textsc{T-False})$$

$$\frac{\Gamma \rhd M : \mathbb{B} \quad \Gamma \rhd P : \sigma \quad \Gamma \rhd Q : \sigma}{\Gamma \rhd \mathit{if}\ \ M\ \ \mathit{then}\ P\ \mathit{else}\ Q : \sigma}\ (\textsc{T-If})$$

$$\frac{\Gamma, x : \sigma \rhd M : \tau}{\Gamma \rhd \lambda x : \sigma . M : \sigma \to \tau}\ (\textsc{T-Abs}) \qquad \frac{\Gamma \rhd M : \sigma \to \tau \quad \Gamma \rhd N : \sigma}{\Gamma \rhd M\, N : \tau}\ (\textsc{T-App})$$

Examples of Typing Derivations

On the board

1. $\triangleright \lambda x : \mathbb{B}.\lambda f : \mathbb{B} \to \mathbb{B}.f\, x : \mathbb{B} \to (\mathbb{B} \to \mathbb{B}) \to \mathbb{B}$
2. $x : \mathbb{B}, y : \mathbb{B} \triangleright \mathit{if}\ x\ \mathit{then}\ y\ \mathit{else}\ y : \mathbb{B}$
3. $\triangleright$
   $\lambda f : \rho \to \tau.\lambda g : \sigma \to \rho.\lambda x : \sigma.f(g\, x) : (\rho \to \tau) \to (\sigma \to \rho) \to \sigma \to \tau$
4. There are no Γ and σ s.t. $\Gamma \triangleright \mathit{true}\ \mathit{false} : \sigma$
5. Are there Γ and σ s.t. $\Gamma \triangleright x\, x : \sigma$?

Basic Metatheory

Uniqueness of Types

If $\Gamma \triangleright M : \sigma$ and $\Gamma \triangleright M : \tau$ are derivable, then $\sigma = \tau$

Weakening+Strengthening

If $\Gamma \triangleright M : \sigma$ is derivable and $\Gamma \cap \Gamma'$ contains all the free variables of M, then $\Gamma' \triangleright M : \sigma$

Substitution

If $\Gamma, x : \sigma \triangleright M : \tau$ and $\Gamma \triangleright N : \sigma$ are derivable, then $\Gamma \triangleright M\{x := N\} : \tau$ is derivable

Typability Questions

- Typability

$$? \triangleright M :?$$

$$\Gamma \triangleright M :?$$

- Type-Checking

$$\frac{?}{\Gamma \triangleright M : \sigma}$$

- Inhabitation

$$\Gamma \triangleright \, ? : \sigma$$

Semantics

- Having defined the syntax of $\lambda_V^{\mathbb{B},\rightarrow}$, we are interested in addressing how terms are evaluated or executed
- There are various ways of defining the semantics of a PL rigorously:
  - Operational
  - Denotational
  - Axiomatic
- We will define the operational semantics of $\lambda_V^{\mathbb{B},\rightarrow}$

What is “Operational Semantics”

- It consists in:
  - interpreting the terms as states of an abstract machine; and
  - defining a transition function that indicates, given a state, what the next state is
- Meaning of a term M: the final state reached by the abstract machine when it starts operating from state M
- Types of operational semantics:
  1. Small-step: the transition function describes one step of computation
  2. Big-step (or Natural Semantics): the transition function, in one step, evaluates the term to its result (if it exists)

Operational Semantics

- Formulated through reduction judgements

$$M \rightarrow N$$

  "term M reduces, in one step, to term N"

- Meaning of such judgements will be established through:
  - Reduction Axioms
  - Reduction Rules

Small-Step Operational Semantics for $\lambda_V^{\mathbb{B},\to}$

- Operational semantics shall presented in two steps
  - Define the set of values: expected results of a computation
  - Define the axioms and rules

Operational Semantics - Functions

Values

$$V \quad ::= \quad \mathit{true} \mid \mathit{false} \mid \lambda x : \sigma.M$$

By declaring those expressions as values we are saying:z every closed, well-typed term of type

- $\mathbb{B}$ evaluates, in zero or more steps, to *true*, *false*
- $\sigma \to \tau$, evaluates, in zero or more steps, to $\lambda x : \sigma.M$, for some variable x and term M

Operational Semantics - Boolean Expressions

$$\frac{}{\textit{if true then } M_2 \textit{ else } M_3 \rightarrow M_2}\ (\textsc{E-IfTrue})$$

$$\frac{}{\textit{if false then } M_2 \textit{ else } M_3 \rightarrow M_3}\ (\textsc{E-IfFalse})$$

$$\frac{M_1 \rightarrow M_1'}{\textit{if } M_1 \textit{ then } M_2 \textit{ else } M_3 \rightarrow \textit{if } M_1' \textit{ then } M_2 \textit{ else } M_3}\ (\textsc{E-If})$$

Operational Semantics - Functions

$$\frac{M_1 \to M_1'}{M_1\, M_2 \to M_1'\, M_2} \text{ (E-APP1)}$$

$$\frac{M_2 \to M_2'}{(\lambda x : \sigma . M_1)\, M_2 \to (\lambda x : \sigma . M_1)\, M_2'} \text{ (E-APP2)}$$

$$\frac{}{(\lambda x : \sigma . M)\, V \to M\{x := V\}} \text{ (E-APPABS)}$$

Examples

- *if* (*if false then false else true*) *then false else true* $\to$ *if true then false else true*
- $(\lambda y : \mathbb{B}.y)\, \mathit{true} \to \mathit{true}$
- $(\lambda x : \mathbb{B} \to \mathbb{B}.x\, \mathit{true})\, (\lambda y : \mathbb{B}.y) \to (\lambda y : \mathbb{B}.y)\, \mathit{true}$
- $(\lambda z : \mathbb{B}.z)\, ((\lambda y : \mathbb{B}.y)\, \mathit{true}) \to (\lambda z : \mathbb{B}.z)\, \mathit{true}$
- There is no M s.t. $\mathit{true} \to M$ (same with *false*).
- There is no M' s.t. $x \to M'$
  - x is in normal form but is not a value

Examples

$$\begin{array}{rl} & \textit{if true then } (\textit{if false then false else true}) \textit{ else true} \\ \not\rightarrow & \textit{if true then true else true} \end{array}$$

- This is quite expected:
  1. First evaluate guard
  2. Then evaluate then or else expression, according to value of guard
- Same thing with reduction under a lambda

Properties

Lemma (Determinism of reduction in one step)

If $M \to M'$ and $M \to M''$, then $M' = M''$

Properties

A normal form is a term that cannot be further evaluated (i.e. M is s.t. there is no N, $M \to N$)

Lemma

Every value is in normal form

- The inverse does not hold
  - *if x then true else false*
  - x
  - *true false*

Multi-Step Evaluation

The multi-step evaluation judgment $\twoheadrightarrow$ is the reflexive, transitive closure of $\rightarrow$:

1. Si $M \rightarrow M'$, then $M \twoheadrightarrow M'$
2. $M \twoheadrightarrow M$ for all M
3. Si $M \twoheadrightarrow M'$ and $M' \twoheadrightarrow M''$, then $M \twoheadrightarrow M''$

Example

$$\begin{array}{ll} & \textit{if true then (if false then false else true) else true} \\ \twoheadrightarrow & \textit{true} \end{array}$$

Multi-Step Evaluation

Lemma (Uniqueness of Normal Forms)

If $M \twoheadrightarrow P$ and $M \twoheadrightarrow Q$ with P, Q normal forms, then $P = Q$

Lemma (Termination)

For all M there exists a normal form N such that $M \twoheadrightarrow N$

Error State

- State (=term) that is not a value but in which reduction is stuck
- Represents a state in which the run-time system would generate an exception

Examples

- *if x then M else N*
  - Obs: is not closed
- *true M*
  - Obs: is not typable

Safety of a Typing System

Guarantee the absence of error states

- If a closed term is typed (and terminates!), then it evaluates to a value

Recommend reading

- Type systems, Luca Cardelli, The Computer Science and Engineering Handbook, CRC Press, 2004.

Correctness

$$\text{Correction} = \text{Progress} + \text{Preservation}$$

Progress

If M is closed and typable, then

1. M is a value; or
2. there exists M' s.t. $M \to M'$

Evaluation cannot get stuck on closed, typable terms that are not values

Preservation

If $\Gamma \rhd M : \sigma$ and $M \to N$, then $\Gamma \rhd N : \sigma$

Evaluation preserves types

Big-Step Operational Semantics (for $\lambda_V^{\rightarrow}$)

$$\frac{}{V \Downarrow V}\ (\textsc{EB-Val})$$

$$\frac{M \Downarrow \mathit{true} \quad P \Downarrow V}{\mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ Q \Downarrow V}\ (\textsc{EB-IfT})$$

$$\frac{M \Downarrow \mathit{false} \quad Q \Downarrow V}{\mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ Q \Downarrow V}\ (\textsc{EB-IfF})$$

$$\frac{M \Downarrow \lambda x : \sigma.P \quad N \Downarrow W \quad P\{x := W\} \Downarrow V}{M\,N \Downarrow V}\ (\textsc{EB-App})$$

Equivalence

$$M \twoheadrightarrow V \text{ iff } M \Downarrow V$$

- ($\Rightarrow$) Consider first proving: $M \rightarrow N$ and $N \Downarrow V$ implies $M \Downarrow V$
- ($\Leftarrow$) Induction on the derivation of $M \Downarrow V$

Types and Terms

$$\sigma \quad ::= \quad \mathbb{B} \,|\, \mathbb{N} \,|\, \sigma \to \rho$$

$$M \quad ::= \quad \ldots \,|\, 0 \,|\, \mathit{succ}(M) \,|\, \mathit{pred}(M) \,|\, \mathit{iszero}(M)$$

Informally:

- $\mathit{succ}(M)$: evaluate M until it yields a numeral and then increment it
- $\mathit{pred}(M)$: evaluate M until it yields a numeral and then decrement it
- $\mathit{iszero}(M)$: evaluate M until it yields a numeral and then return *true*/*false* depending on whether it is zero or not

Typing

$$\frac{}{\Gamma \rhd 0 : \mathbb{N}} \text{(T-Zero)}$$

$$\frac{\Gamma \rhd M : \mathbb{N}}{\Gamma \rhd \mathit{succ}(M) : \mathbb{N}} \text{(T-Succ)} \qquad \frac{\Gamma \rhd M : \mathbb{N}}{\Gamma \rhd \mathit{pred}(M) : \mathbb{N}} \text{(T-Pred)}$$

$$\frac{\Gamma \rhd M : \mathbb{N}}{\Gamma \rhd \mathit{iszero}(M) : \mathbb{B}} \text{(T-IsZero)}$$

Values and One-Step Evaluation (1/2)

Values

$$V \quad ::= \quad \ldots \mid \underline{n} \quad \text{where } \underline{n} \text{ abbreviates } succ^n(0).$$

Evaluation (1/2)

$$\frac{M_1 \to M_1'}{succ(M_1) \to succ(M_1')} \ (\textsc{E-Succ})$$

$$\frac{}{pred(0) \to 0} \ (\textsc{E-PredZero}) \qquad \frac{}{pred(\underline{n+1}) \to \underline{n}} \ (\textsc{E-PredSucc})$$

$$\frac{M_1 \to M_1'}{pred(M_1) \to pred(M_1')} \ (\textsc{E-Pred})$$

Values and One-Step Evaluation (2/2)

Evaluation (2/2)

$$\frac{}{\mathit{iszero}(0) \to \mathit{true}} \ (\textsc{E-IsZeroZero})$$

$$\frac{}{\mathit{iszero}(\underline{n+1}) \to \mathit{false}} \ (\textsc{E-IsZeroSucc})$$

$$\frac{M_1 \to M_1'}{\mathit{iszero}(M_1) \to \mathit{iszero}(M_1')} \ (\textsc{E-IsZero})$$

Types and Terms

$$\sigma \quad ::= \quad \mathbb{B} \,|\, \mathbb{N} \,|\, \mathbb{U} \,|\, \sigma \to \rho$$

$$M \quad ::= \quad \ldots \,|\, \mathit{unit}$$

Informally:

- $\mathbb{U}$ is a unitary type: its only possible value is *unit*.

Typing

$$\frac{}{\Gamma \rhd \mathit{unit} : \mathbb{U}} \ (\textsc{T-Unit})$$

- No rules for evaluation
- We extend the set of values V with *unit*

$$V \quad ::= \quad \ldots \mid \mathit{unit}$$

Use

- Languages with side-effects (next class)
- Sequencing

$$M_1; M_2 \stackrel{\text{def}}{=} (\lambda x : \mathbb{U}.M_2)\, M_1 \qquad x \notin FV(M_2)$$

- Evaluation of $M_1; M_2$ consists in first evaluating M_1, then M_2

Types and Terms

$$M \quad ::= \quad \ldots \mid let\ x : \sigma = M\ in\ N$$

Informally:

- $let\ x : \sigma = M\ in\ N$: evaluate M to a value V, bind x to V in N and evaluate the resulting term
- Helps readability
- No new types required

Example

- $let\ x : \mathbb{N} = \underline{2}\ in\ succ(x)$
- $pred\ (let\ x : \mathbb{N} = \underline{2}\ in\ x)$
- $let\ x : \mathbb{N} = \underline{2}\ in\ let\ x : \mathbb{N} = \underline{3}\ in\ x$

Typing

$$\frac{\Gamma \rhd M : \sigma_1 \quad \Gamma, x : \sigma_1 \rhd N : \sigma_2}{\Gamma \rhd \mathit{let}\ x : \sigma_1 = M\ \mathit{in}\ N : \sigma_2}\ (\textsc{T-Let})$$

Operational Semantics

$$\frac{M_1 \rightarrow M_1'}{\mathit{let}\ x : \sigma = M_1\ \mathit{in}\ M_2 \rightarrow \mathit{let}\ x : \sigma = M_1'\ \mathit{in}\ M_2}\ (\text{E-Let})$$

$$\frac{}{\mathit{let}\ x : \sigma = V\ \mathit{in}\ M \rightarrow M\{x := V\}}\ (\text{E-LetV})$$

Types and Terms

Let $\mathcal{L}$ be a set of labels

$$\sigma \quad ::= \quad \ldots \mid \{l_i : \sigma_i \ ^{i \in 1..n}\}$$

- $\{name : String, age : \mathbb{N}\}$
- $\{person : \{name : String, age : \mathbb{N}\}, cwid : \mathbb{N}\}$

$\{name : String, age : \mathbb{N}\} \neq \{age : \mathbb{N}, name : String\}$

Types and Terms

$$M \quad ::= \quad \ldots \mid \{l_i = M_i \ ^{i \in 1..n}\} \mid M.l$$

Informally:

- The record $\{l_i = M_i \ ^{i \in 1..n}\}$ evaluates to $\{l_i = V_i \ ^{i \in 1..n}\}$ where V_i is the value of M_i, $i \in 1..n$
- $M.l$: evaluates M until it yields $\{l_i = V_i \ ^{i \in 1..n}\}$, then it projects the corresponding field

Examples

- $\lambda x : \mathbb{N}.\lambda y : \mathbb{B}.\{age = x, gender = y\}$
- $\lambda p : \{age : \mathbb{N}, gender : \mathbb{B}\}.p.age$
- $(\lambda p : \{age : \mathbb{N}, gender : \mathbb{B}\}.p.age)\,\{age = 20, gender = false\}$

Typing

$$\frac{\Gamma \rhd M_i : \sigma_i \quad \text{for each } i \in 1..n}{\Gamma \rhd \{l_i = M_i \;^{i\in 1..n}\} : \{l_i : \sigma_i \;^{i\in 1..n}\}} \text{(T-RCD)}$$

$$\frac{\Gamma \rhd M : \{l_i : \sigma_i \;^{i\in 1..n}\} \quad j \in 1..n}{\Gamma \rhd M.l_j : \sigma_j} \text{(T-PROJ)}$$

Operational Semantics

Values

$$V \quad ::= \quad \ldots \mid \{l_i = V_i \;{}^{i \in 1..n}\}$$

Operational Semantics

$$\frac{j \in 1..n}{\{l_i = V_i \ ^{i \in 1..n}\}.l_j \to V_j} \text{ (E-ProjRcd)}$$

$$\frac{M \to M'}{M.l \to M'.l} \text{ (E-Proj)}$$

$$\frac{M_j \to M_j'}{\begin{array}{c}\{l_i = V_i \ ^{i \in 1..j-1}, l_j = M_j, l_i = M_i \ ^{i \in j+1..n}\} \\ \to \\ \{l_i = V_i \ ^{i \in 1..j-1}, l_j = M_j', l_i = M_i \ ^{i \in j+1..n}\}\end{array}} \text{ (E-Rcd)}$$