# Typing Rule of Baremetalisp

Yuuki Takano
ytakano@wide.ad.jp

April 30, 2020

## 1 Introduction

In this paper, I will formally describe the typing rule of Baremetalisp, which is a well typed Lisp for trusted execution environment.

## 2 Notation and Syntax

Table 1 and Fig. 1 shows notation used in this paper and syntax for the typing rule, respectively.

Listing 1: Example of variable and type

```
(defun add (a b) (Pure (-> (Int Int) Int))
    (+ a b))
```

$x$ is a variable. For example, $x \in \{a, b\}$ in Listing 1. $\mathcal{T}$ is a type. For example, $\mathcal{T} \in \{\texttt{Int}, (\rightarrow\ (\texttt{Int Int})\ \texttt{Int})\}$ in Listing 1. $(\rightarrow\ (\texttt{Int Int})\ \texttt{Int})$ is a function type which takes 2 integer values and return 1 integer value. `Pure` in Listing 1 denotes the effect of the function but I just ignore it now. Function effects will be described in Sec. 4. $\mathcal{T}$ can be other forms as described in Fig. 1 such as `Bool`, $'(\texttt{Int})$, $[\texttt{Bool Int}]$, $(\text{List } a)$, $(\text{List } \texttt{Int})$. $\Gamma$ is a map from variable to type. For example, $\Gamma = \{a : t_1, b : t_2, + : (\rightarrow\ (\texttt{Int Int})\ \texttt{Int})\}$ in Listing 1. $\Gamma$ is called context generally, thus I call $\Gamma$ context in this paper. $C$ is a type constraint, which is a set of pairs of types. For example, $C = \{(\rightarrow\ (t_1\ t_2)\ t) = (\rightarrow\ (\texttt{Int Int})\ \texttt{Int})\}$ deduced from Listing 1 means $(\rightarrow\ (t_1\ t_2)\ t)$ and $(\rightarrow\ (\texttt{Int Int})\ \texttt{Int})$ are semantically equal and every type variable in $C$, $t_1, t_2, t$, is thus `Int`.

Listing 2: Example of user defined data type

```
(data (List a)
    (Cons a (List a))
    Nil)
```

$t$ is a type variable. For example, $t \in \{a\}$ in Listing 2. $L$ is a label for user defined type. For example, $L \in \{\text{Cons}, \text{Nil}\}$ in Listing 2. $L_{type}$ is a function from

Table 1: Notation

| | |
|---|---|
| $e$ | expression |
| $z$ | integer literal such as 10, -34, 112 |
| $x$ | variable |
| $t$ | type variable |
| $D$ | type name of user defined data |
| $L$ | label of user defined data |
| $\mathcal{T}$ | type |
| $C$ | type constraint |
| $\Gamma : x \to \mathcal{T}$ | context |
| $L_{type} : L \to \mathcal{T}$ | function from label to type |
| $L_{nth} : L \to \texttt{Int} \to \mathcal{T}$ | function to n-th type of label $L$ |
| $\mathcal{P}$ | pattern |
| $\mathcal{P}_{let}$ | pattern of let expression |
| $\mathcal{T}_1 \equiv_\alpha \mathcal{T}_2$ | $\mathcal{T}_1$ and $\mathcal{T}_2$ are $\alpha$-equivalent |
| $\mathcal{S} : t \to \mathcal{T}$ | substitution from type variable to type |
| $\mathcal{T} \cdot \mathcal{S}$ | apply $\mathcal{S}$ to $\mathcal{T}$ |
| $\mathcal{X}$ | set of $t$ |
| $FV_{\mathcal{T}} : \mathcal{T} \to \mathcal{X}$ | function from $\mathcal{T}$ to its free variables |
| $FV_{\Gamma} : \Gamma \to \mathcal{X}$ | function from $\Gamma$ to its free variables |
| $Size : L \to \texttt{Int}$ | the number of labels $L$'s type has |
| $\Gamma \vdash e : \mathcal{T} \mid_{\mathcal{X}} C$ | $e$'s type is deduced as $\mathcal{T}$ from $\Gamma$ under constraint $C$ and type variables $\mathcal{X}$ |

label to type. For example, $L_{type}(\text{Cons}) = (\text{List } a)$ and $L_{type}(\text{Nil}) = (\text{List } a)$ in Listing 2. $L_{nth}$ is a function to n-th type of label. For example, $L_{nth}(\text{Cons}, 0) = a$ and $L_{nth}(\text{Cons}, 1) = (\text{List } a)$ in Listing 2. $D$ is user defined data. For example, $D \in \{\text{List}\}$ in Listing 2.

$FV_{\mathcal{T}}$ and $FV_{\Gamma}$ are functions, which take $\mathcal{T}$ and $\Gamma$ and return free variables. For example, $FV_{\mathcal{T}}((\to\ (t_1\ t_2)\ t)) = \{t_1, t_2, t\}$ and

$$\begin{aligned}
&FV_{\Gamma}(\{a : t_1, b : t_1, + : (\to\ (\texttt{Int Int})\ \texttt{Int})\}) \\
&= \{FV_{\mathcal{T}}(t_1), FV_{\mathcal{T}}(t_1), FV_{\mathcal{T}}((\to\ (\texttt{Int Int})\ \texttt{Int}))\} \\
&= \{t_1, t_2\}.
\end{aligned}$$

$\mathcal{T}_1 \equiv_\alpha \mathcal{T}_2$ denotes that $\mathcal{T}_1$ and $\mathcal{T}_2$ are $\alpha$-equivalent, which means $\mathcal{T}_1$ and $\mathcal{T}_2$ are semantically equal. For example, $(\to\ (t_1\ t_2)\ t) \equiv_\alpha (\to\ (t_{10}\ t_{11})\ t_{12})$. $\mathcal{S}$ is a substitution, which is a map from type variable to type, and it can be applied to $\mathcal{T}$ as $\mathcal{T} \cdot \mathcal{S}$. For example, if $\mathcal{S}(t_1) = [\texttt{Bool Int}], \mathcal{S}(t_2) = (\text{List } t_3)$ then $(\to\ (t_1\ t_2)\ t) \cdot \mathcal{S} = (\to\ ([\texttt{Bool Int}]\ (\text{List } t_3))\ t)$.

Listing 3: Example of pattern matching

```
(data Dim2 (Dim2 Int Int))

(data (Maybe t)
    (Just t)
```

| | | | |
|---|---|---|---|
| $\mathcal{C}$ | $:=$ | $\mathcal{T} = \mathcal{T}, \mathcal{C}$ | **type constraint** |
| | $\mid$ | $\varnothing$ | |
| $\mathcal{T}$ | $:=$ | | **type** |
| | | `Int` | |
| | $\mid$ | `Bool` | |
| | $\mid$ | $'(\mathcal{T})$ | list type |
| | $\mid$ | $[\mathcal{T}+]$ | tuple type |
| | $\mid$ | $D$ | user defined type |
| | $\mid$ | $(D\ \mathcal{T}+)$ | user defined type with type arguments |
| | $\mid$ | $(\rightarrow\ (\mathcal{T}*)\ \mathcal{T})$ | function type |
| | $\mid$ | $t$ | type variable |
| $\mathcal{P}$ | $:=$ | | **pattern** |
| | | $x$ | variable |
| | $\mid$ | $L$ | label |
| | $\mid$ | $(L\ \mathcal{P}+)$ | label with patterns |
| | $\mid$ | $'()$ | empty list |
| | $\mid$ | $[\mathcal{P}+]$ | tuple |
| $\mathcal{P}_{let}$ | $:=$ | | **patten for let** |
| | | $x$ | variable |
| | $\mid$ | $(L\ \mathcal{P}_{let}+)$ | label with patterns |
| | $\mid$ | $[\mathcal{P}_{let}+]$ | tuple |

Figure 1: Syntax

```
5      Nothing)
6
7  (defun match-let (a) (Pure (-> ((Maybe Dim2)) Int))
8      (match a
9          ((Just val)
10             (let (((Dim2 x y) val))
11                 (+ x y)))
12         (Nothing
13             0)))
```

$\mathcal{P}$ and $\mathcal{P}_{let}$ are pattern in match and let expressions. For example, in listings 3, *(Just val)* and *Nothing* at line 9 and 12 are from $\mathcal{P}$ and *(Dim2 x y)* at line 10 is from $\mathcal{P}_{let}$. *Size* is a function which takes a label and return the number of labels the label's type has. For example, $Size(\text{Just}) = Size(\text{Nothing}) = 2$ because Maybe type has 2 labels and $Size(\text{Dim2}) = 1$ because Dim2 type has 1 label in listings 3.

# 3 Typing Rule

# 4 Effect

$$\Gamma \vdash \texttt{true} : \texttt{Bool} \mid_{\varnothing} \varnothing \quad \text{(T-True)} \qquad \Gamma \vdash \texttt{false} : \texttt{Bool} \mid_{\varnothing} \varnothing \quad \text{(T-False)}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T \mid_{\varnothing} \varnothing} \quad \text{(T-Var)} \qquad \Gamma \vdash z : \texttt{Int} \mid_{\varnothing} \varnothing \quad \text{(T-Num)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathcal{P}_{let} : \mathcal{T}_0 \mid_{\mathcal{X}_0} C_0 \quad \Gamma \vdash e_1 : \mathcal{T}_1 \mid_{\mathcal{X}_1} C_1 \quad \Gamma \vdash e_2 : \mathcal{T}_2 \mid_{\mathcal{X}_2} C_2 \\ \mathcal{X}_0 \cap \mathcal{X}_1 \cap \mathcal{X}_2 = \varnothing \quad C = C_0 \cup C_1 \cup C_2 \cup \{\mathcal{T}_0 = \mathcal{T}_1\}\end{array}}{\Gamma \vdash (\texttt{let1}\ \mathcal{P}_{let}\ e_1\ e_2) : \mathcal{T}_2 \mid_{\mathcal{X}_0 \cup \mathcal{X}_1 \cup \mathcal{X}_2} C} \quad \text{(T-Let1)}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \mathcal{T}_1 \mid_{\mathcal{X}_1} C_1 \quad \Gamma \vdash e_2 : \mathcal{T}_2 \mid_{\mathcal{X}_2} C_2 \quad \Gamma \vdash e_3 : \mathcal{T}_3 \mid_{\mathcal{X}_3} C_3 \\ \mathcal{X}_1 \cap \mathcal{X}_2 \cap \mathcal{X}_3 = \varnothing \quad C = C_1 \cup C_2 \cup C_3 \cup \{\mathcal{T}_1 = \texttt{Bool}, \mathcal{T}_2 = T_3\}\end{array}}{\Gamma \vdash (\texttt{if}\ e_1\ e_2\ e_3) : \mathcal{T}_2 \mid_{\mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3} C} \quad \text{(T-If)}$$

$$\frac{\begin{array}{l}\Gamma \vdash e_1 : \mathcal{T}_1 \mid_{\mathcal{X}_1} C_1 \quad \Gamma \vdash e_2 : \mathcal{T}_2 \mid_{\mathcal{X}_2} C_2 \wedge \cdots \wedge \Gamma \vdash e_n : \mathcal{T}_n \mid_{\mathcal{X}_n} C_n \\ \{t\} \cap FV_{\Gamma}(\Gamma) = \varnothing \quad \{t\} \cap \mathcal{X}_1 \cap \cdots \cap \mathcal{X}_n = \varnothing \\ \mathcal{X} = \{t\} \cup \mathcal{X}_1 \cup \cdots \cup \mathcal{X}_n \\ C = C_1 \cup \cdots \cup C_n \cup \{\mathcal{T}_1 = (\rightarrow\ (\mathcal{T}_2\ \cdots\ \mathcal{T}_n)\ t)\}\end{array}}{\Gamma \vdash (e_1\ e_2\ \cdots\ e_n) : t \mid_{\mathcal{X}} C} \quad \text{(T-App)}$$

$$\frac{\begin{array}{l}\Gamma \vdash e_0 : \mathcal{T}_0 \mid_{\mathcal{X}_0} C_0 \\ \Gamma \vdash e_1 : \mathcal{T}_{e1} \mid_{\mathcal{X}_{e1}} C_{e1} \wedge \cdots \wedge \Gamma \vdash e_n : \mathcal{T}_{en} \mid_{\mathcal{X}_{en}} C_{en} \\ \Gamma \vdash \mathcal{P}_1 : \mathcal{T}_{p1} \mid_{\mathcal{X}_{p1}} C_{p1} \wedge \cdots \wedge \Gamma \vdash \mathcal{P}_{pn} : \mathcal{T}_{pn} \mid_{\mathcal{X}_{pn}} C_{pn} \\ \mathcal{X}_0 \cap \mathcal{X}_{e1} \cap \cdots \cap \mathcal{X}_{en} \cap \mathcal{X}_{p1} \cap \cdots \cap \mathcal{X}_{pn} = \varnothing \\ \mathcal{X} = \mathcal{X}_0 \cup \mathcal{X}_{e1} \cup \cdots \cup \mathcal{X}_{en} \cup \mathcal{X}_{p1} \cup \cdots \cup \mathcal{X}_{pn} \\ C = C_0 \cup C_{e1} \cup \cdots \cup C_{en} \cup C_{p1} \cup \cdots \cup C_{pn} \cup \\ \qquad \{\mathcal{T}_0 = \mathcal{T}_{p1}, \cdots, \mathcal{T}_0 = \mathcal{T}_{pn}\} \cup \{\mathcal{T}_{e1} = \mathcal{T}_{e2}, \cdots, \mathcal{T}_{e1} = \mathcal{T}_{en}\}\end{array}}{\Gamma \vdash (\texttt{match}\ e_0\ (\mathcal{P}_1\ e_1)\ \cdots\ (\mathcal{P}_n\ e_n)) : T_{e1} \mid_{\mathcal{X}} C} \quad \text{(T-Match)}$$

$$\frac{\begin{array}{l}\Gamma \vdash x_1 : \mathcal{T}_1 \mid_{\varnothing} \varnothing \wedge \cdots \wedge \Gamma \vdash x_n : \mathcal{T}_n \mid_{\varnothing} \varnothing \\ \Gamma \vdash e : \mathcal{T}_0 \mid_{\mathcal{X}} C_0 \quad FV_{\mathcal{T}}(\mathcal{T}) = FV_{\mathcal{T}}(\mathcal{T}_1) = \cdots = FV_{\mathcal{T}}(\mathcal{T}_n) = \varnothing \\ C = C_0 \cup \{\mathcal{T} = (\rightarrow (\mathcal{T}_1\ \cdots\ \mathcal{T}_n)\ \mathcal{T}_0)\}\end{array}}{\Gamma \vdash (\texttt{defun}\ \text{name}\ (x_1\ \cdots\ x_n)\ \mathcal{T}\ e) : \mathcal{T} \mid_{\mathcal{X}} C} \quad \text{(T-Defun)}$$

Figure 2: Typing rule

$$\Gamma \vdash \ '() : \ '(T) \ |_{\{T\}} \ \varnothing \quad \text{(P-Nil)} \qquad\qquad \frac{L_{type}(L) \cdot \mathcal{S} \equiv_\alpha \mathcal{T}}{\Gamma \vdash L : \mathcal{T} \ |_{FV_{\mathcal{T}}(\mathcal{T})} \ \varnothing} \quad \text{(P-Label0)}$$

$$\frac{\begin{array}{l} \Gamma \vdash \mathcal{P}_1 : \mathcal{T}_1 \ |_{\mathcal{X}_1} \ C_1 \wedge \cdots \wedge \Gamma \vdash \mathcal{P}_n : \mathcal{T}_n \ |_{\mathcal{X}_n} \ C_n \\ L_{type}(L) \cdot \mathcal{S} \equiv_\alpha \mathcal{T}_0 \quad FV(\mathcal{T}_0) \cap \mathcal{X}_1 \cap \cdots \cap \mathcal{X}_n = \varnothing \\ FV_{\mathcal{T}}(\mathcal{T}_0) \cap FV_\Gamma(\Gamma) = \varnothing \quad \mathcal{X} = FV(\mathcal{T}_0) \cup \mathcal{X}_1 \cup \cdots \cup \mathcal{X}_n \\ C = C_1 \cup \cdots \cup C_n \cup \{L_{nth}(L, 1) \cdot \mathcal{S} = \mathcal{T}_1, \cdots, L_{nth}(L, n) \cdot \mathcal{S} = \mathcal{T}_n\} \\ Size(L) = 1 \text{ for only } P_{let} \end{array}}{\Gamma \vdash (L \ \mathcal{P}_1 \ \cdots \ \mathcal{P}_n) : \mathcal{T}_0 \ |_{\mathcal{X}} \ C} \quad \text{(P-Label)}$$

$$\frac{\begin{array}{l} \Gamma \vdash \mathcal{P}_1 : \mathcal{T}_1 \ |_{\mathcal{X}_1} \ C_1 \wedge \cdots \wedge \Gamma \vdash \mathcal{P}_n : \mathcal{T}_n \ |_{\mathcal{X}_n} \ C_n \\ \mathcal{X}_1 \cap \cdots \cap \mathcal{X}_n = \varnothing \quad \mathcal{X} = \mathcal{X}_1 \cup \cdots \cup \mathcal{X}_n \quad C = C_1 \cup \cdots \cup C_n \end{array}}{\Gamma \vdash [\mathcal{P}_1 \ \cdots \ \mathcal{P}_n] : [\mathcal{T}_1 \cdots \mathcal{T}_n] \ |_{\mathcal{X}} \ C} \quad \text{(P-Tuple)}$$

Figure 3: Typing rule of pattern