# LambdaM: A Simple Language with Termination Checking based on Dependent Types

Wenhao Tang
1800013088

May 31, 2020

Department of EECS, Peking University

# Main Work

$\lambda_M$, where the $M$ stands for metrics, is a simple language with *termination checking* based on *dependent types*.

- Implement the complete mechanism for ***dependent product types*** and ***dependent sum types***, together with *a primitive vector type dependented on natural numbers*.
- Introduce the ***recursive functions with metrics*** which will be checked for termination during typing.
- The termination checking supports ***different comparison functions between metrics***.

The source code of $\lambda_M$: `https://github.com/thwfhk/lambdaM`

# A Simple Example

Consider the function which calculates the sum of a list:

$$sum\ [] = 0$$

$$sum\ (x : xs) = x + sum\ xs$$

It can be written in $\lambda_M$:

```
fun sum : [n] Πv:Vector n. Nat.
  λv : Vector n.
    if isnil v then 0
    else head v + sum [pred n] (tail v)
```

Instuitively, since $pred\ n < n$ is always true, it will terminate given any input.

# Dependent Types

**Dependent product types**: $\Pi x : S.T$, generalization of $S \to T$.
**Dependent sum types**: $\Sigma x : S.T$, generalization of $(S, T)$.

For example, consider the vector type dependented on natural numbers:

- $Vector :: Nat \to *$
- $Vector\ n$ is the type of all vectors with length $n$.
- $cons : \Pi n : Nat.\ \Pi x : Nat.\ \Pi v : Vector\ n.\ Vector\ (succ\ n)$
- $(n, v) : (\Sigma n : Nat.\ Vector\ n)$

# Metrics

New syntax:

| | | | |
|---|---|---|---|
| $t$ | $::=$ | ... | terms: |
| | | $fun\ f\ :\ [m]\ T.t$ | recursive functions with metrics |
| | | $f\ [m]$ | application of functions to metrics |
| $m$ | $::=$ | | metrics: |
| | | $t$ | terms |
| | | $m, t$ | tuples of terms |

Back to the example of $sum : \Pi n.\ \Pi v : Vector\ n.\ Nat$

```
fun sum : [n] Πv:Vector n. Nat.
  λv : Vector n.
    if isnil v then 0
    else head v + sum [pred n] (tail v)
```

# Metrics

For simplicity we only consider the vector types dependented on natural numbers.

**Definition (Metric of Function)**

Suppose all parameters of type Vector $n$ of function $f$ are $Vector\ n_1$, $Vector\ n_2$, ..., $Vector\ n_k$, then an arbitrary subsequence of $n_1, \dots, n_k$ is a metric of function $f$.

If we choose $n_{i_1}, \dots, n_{i_p}$ to be the metric of function $f$, then we remove every $\Pi n_{i_j}$ from the type of $f$ and write the definition of $f$ as $fun\ f\ :\ [n_{i_1}, \dots, n_{i_p}]T.t$.

# Termination Checking

**Theorem (Termination of Function)**

*Suppose function $f$ has a metric $n_1, \ldots, n_k$ and is defined as $fun\ f : [n_1, \ldots, n_k]T.t$, if every recursive call of the form $f\ [m_1, \ldots, m_k]\ t'$ in $t$ satisfy $(m_1, \ldots, m_k) < (n_1, \ldots, n_k)$, then $f$ will terminates given any input.*

$<$ is any comparison operator between tuples of natural numbers which guarantees it won't decrease forever.

- Usual comparison between tuples.
- Lexicographical order.
- $(n_1, \ldots, n_k) < (m_1, \ldots, m_k) = \sum_{i=1}^{k} n_i < \sum_{i=1}^{k} m_i$

# Typing Rules for Metrics

**Definition (Definitional Metric of Function)**

Suppose $f$ is a function whose definition is $fun\ f : [m].T.t$, then $m_f := m$ is the definitional metric of $f$.

**Definition (Typing Relation with Metric)**

The Typing Relation with Metrics $\boxed{\Gamma \vdash t : T <_f m_f}$

- the term $t$ has type $T$ under context $\Gamma$.
- $f$ is a recursive function with a definitional metric $m_f$ and for every occurence of $f[m]$ in term $t$, $m < m_f$ is true.

# Typing Rules for Metrics

Typing: $\boxed{\Gamma \vdash t : T}$

$$\frac{\Gamma \vdash n_i : Nat \quad \Gamma \vdash T :: \boxed{?} \qquad \Gamma, m_f : (n_1, \dots, n_k), f : \Pi n_1 : Nat \dots \Pi n_k : Nat.T, n_i : Nat \vdash t : T <_f m_f}{\Gamma \vdash fun\ f : [n_1, n_2, \dots, n_k]\ T.t : \Pi n_1 : Nat \dots \Pi n_k : Nat.T}$$

$$\frac{\Gamma \vdash f : \Pi n_1 : Nat \dots \Pi n_k : Nat.T \quad m = (m_1, \dots, m_k) \quad \Gamma \vdash m_i : Nat}{\Gamma \vdash f[m] : [n_1 \mapsto m_1, \dots, n_k \mapsto m_k]T}$$

Typing with metric: $\boxed{\Gamma \vdash t : T <_f m_f}$

$$\frac{\Gamma \vdash f : \Pi n_1 : Nat \dots \Pi n_k : Nat.T \quad m = (m_1 \dots m_k) \quad \Gamma \vdash m_i : Nat \quad m < m_f}{\Gamma \vdash f[m] : [n_1 \mapsto m_1, \dots, n_k \mapsto m_k]T <_f m_f}$$

# Evaluation Rules

Just use the derived forms:

$$\text{fun } f : [n_1, \dots, n_k] T.t \stackrel{\text{def}}{=}$$
$$\text{fix } (\lambda f : \Pi n_1 : \text{Nat}. \dots \Pi n_k : \text{Nat}.T \,.\, \lambda n_1 : \text{Nat}. \dots \lambda n_k : \text{Nat}.t)$$

$$f[n_1, n_2, \dots, n_k] \stackrel{\text{def}}{=} (\dots((f\ n_1) n_2) \dots) n_k$$

# More Examples

```
fun lenless : [n₁, n₂] Π v₁: Vector n₁. Π v₂: Vector n₂. Bool.
  λ v₁ : Vector n₁.
    λ v₂ : Vector n₂.
      if isnil n₁ v₁ then true
      else if isnil n₂ v₂ then false
      else lenless [pred n₁, pred n₂] (tail n₁ v₁) (tail n₂ v₂)
```

A wrong version which will never terminate:

```
fun lenless : [n₁, n₂] Π v₁: Vector n₁. Π v₂: Vector n₂. Bool.
  λ v₁ : Vector n₁.
    λ v₂ : Vector n₂.
      if isnil n₁ v₁ then true
      else if isnil n₂ v₂ then false
      else lenless [n₁, succ n₂] v₁ (cons 1 n₂ v₂)
```

# More Examples

Return a vector of all elements at even positions of the original vector.

```
fun evens : [n] Π v:Vector n. Π d:Nat. Σ p:Nat.Vector(p).
  λ v : Vector n. λ d : Nat.
    if isnil n v then (0, nil)
    else if iseven d then
      (succ (evens (pred n) (tail n v) (succ d)).1,
        cons
        (succ (evens (pred n) (tail n v) (succ d)).1
        (head n v) (evens (pred n) (tail n v) (succ d)).2
      )
    else
      ((evens (pred n) (tail n v) (succ d)).1,
        (evens (pred n) (tail n v) (succ d)).2)
```

# More Examples

$$
g(v_1, v_2) =
\begin{cases}
g(tail\ (tail\ v_1), cons\ 0\ v_2) & \\
\qquad ++ g(cons\ 0\ v_1 + tail\ (tail\ v_2)) & |\ v_1\ | \geq 2 \land |\ v_2\ | \geq 2 \\
v_1 ++ g(cons\ 0\ v_1 + tail\ (tail\ v_2)) & |\ v_1\ | < 2 \land |\ v_2\ | \geq 2 \\
g(tail\ (tail\ v_1), cons\ 0\ v_2) ++ v_2 & |\ v_1\ | \geq 2 \land |\ v_2\ | < 2 \\
v_1 ++ v_2 & |\ v_1\ | < 2 \land |\ v_2\ | < 2
\end{cases}
$$

# Conclusion

$\lambda_M$ is simple language which supports dependent product types and dependent sum types and is able to check the termination of recursive functions with metrics.

The basic idea of $\lambda_M$ is similar to the $ML_{0,\ll}^{\Pi,\Sigma}$ proposed by H. Xi.(2001).

A slight advantage is that $\lambda_M$ supports different comparison functions between metrics and it can be easily extended to other metrics in addition to natural numbers.

# Thanks