Computation and Deduction 2016

Mini-project Report

Student
Dandan Xue

Supervisor
Andrzej Filinski

November 8, 2016

Contents

1 Introduction

We present a language Mini-List by showing its syntax and operational semantics. Then we introduce an equational system for this language and prove its soundness and completeness for evaluation. Finally, we extend Mini-List with **let** -binding and prove that these two properties still hold.

2 The Mini-List Language

2.1 Syntax

The syntax of Mini-List expressions e is as follows:

$$e ::= \mathbf{z} \mid \mathbf{s}\ e \mid \mathbf{nil} \mid e_1 : e_2 \mid e_1 +\!+ e_2 \mid x$$

nil stands for the empty list and $e_1 : e_2$ for a list with the first element e_1 and the others e_2. The operator $+\!+$ denotes the appending operation at the end of the list e_1 with the list e_2.

The syntax of values v is as follows:

$$v ::= \mathbf{z} \mid \mathbf{s}\ v \mid \mathbf{nil} \mid v_1 : v_2$$

2.2 Type system

The types for Mini-List is given by the following grammar:

$$\tau ::= \mathbf{nat} \mid \mathbf{list}\ \tau_1$$

Here **nat** stands for the type of natural numbers, **list** τ_1 is the type of a list of elements that has the type τ_1.

Then we define the following typing rules for expressions.

Judgment $\boxed{e : \tau}$:

T-Z: $\dfrac{}{\mathbf{z} : \mathbf{nat}}$ T-S: $\dfrac{e : \mathbf{nat}}{\mathbf{s}\ e : \mathbf{nat}}$ T-Nil: $\dfrac{}{\mathbf{nil} : \mathbf{list}\ \tau}$ T-Cons: $\dfrac{e_1 : \tau \qquad e_2 : \mathbf{list}\ \tau}{e_1 : e_2 : \mathbf{list}\ \tau}$

T-Append: $\dfrac{e_1 : \mathbf{list}\ \tau \qquad e_2 : \mathbf{list}\ \tau}{e_1 : e_2 : \mathbf{list}\ \tau}$

Figure 1: Type system for Mini-List

We translate the syntax and type system of Mini-List in Twelf as follows. Note that we formalize the expressions and types in *Church − style*.

```
% --- type system
tp : type.   %name tp T.
nat : tp.
list : tp -> tp.

% --- expressions
exp : tp -> type. %name exp E.
ez : exp nat.
es : exp nat -> exp nat.
enil : exp (list T).
econs : exp T -> exp (list T)-> exp (list T).
eappend : exp (list T) -> exp (list T) -> exp (list T).

% --- values
val : tp -> type.  %name val V.
vz : val nat.
vs : val nat -> val nat.
vnil : val (list T).
```

```
vcons : val T -> val (list T) -> val (list T).

% --- convert a value to its counterpart in expression type
val2exp : val T -> exp T -> type.  %name val2exp VE.
%mode val2exp +V -E.
% -- ... (omitted)
```

Note that in the language Mini-List we consider values also expressions, but in Twelf since we use different type families for expressions and values, we should define an extra judgment (such `val2exp` in the code) to convert values to expressions in Twelf.

2.3 Operational semantics

The operational semantics is defined by the rules in Figure 2.

Judgment $\boxed{e \downarrow v}$:

$$\textsc{Eval-Z}: \frac{}{\mathbf{z} \downarrow \mathbf{z}} \qquad \textsc{Eval-S}: \frac{e \downarrow v}{\mathbf{s}\ e \downarrow \mathbf{s}\ v}$$

$$\textsc{Eval-Nil}: \frac{}{\mathbf{nil} \downarrow \mathbf{nil}} \qquad \textsc{Eval-Cons}: \frac{e_1 \downarrow v_1 \qquad e_2 \downarrow v_2}{e_1 : e_2 \downarrow v_1 : v_2}$$

$$\textsc{Eval-Append}: \frac{e_1 \downarrow v_1 \qquad e_2 \downarrow v_2 \qquad \mathbf{fappend}\ (v_1, v_2, v)}{e_1 +\!+ e_2 \downarrow v}$$

Figure 2: Operational Semantics of Mini-List

The function $\mathbf{fappend}\ (v_1, v_2, v)$ in Eval-Append denotes the operation that appending the list v_2 to the end of the list v_1 obtains a new list v. We show its definition in the following figure.

Judgment $\boxed{\mathbf{fappend}\ (v_1, v_2, v)}$:

$$\textsc{Fappend-Nil}: \frac{}{\mathbf{fappend}\ (\mathbf{nil}\ , v, v)} \qquad \textsc{Fappend-cons}: \frac{\mathbf{fappend}\ (v_2, v_3, v)}{\mathbf{fappend}\ ((v_1 : v_2), v_3, (v_1 : v))}$$

Figure 3: The operation **fappend**

We can easily check that **fappend** has the following properties:

Lemma 1 (Fappend Determinacy)**.** *If* $\mathbf{fappend}\ (v_1, v_2, v_3)$*, and* $\mathbf{fappend}\ (v_1, v_2, v_3')$*, then* $v_3 = v_3'$.

Lemma 2 (Fappend Existance)**.** *If there ars two lists* v_1 *and* v_2*, then there must exist some list* v_3 *such that* $\mathbf{fappend}\ (v_1, v_2, v_3)$.

Lemma 3 (Fappend Nil Right)**.** *For any list* v*, there must be* $\mathbf{fappend}\ (v, \mathbf{nil}\ , v)$.

Lemma 4 (Fappend Associativity)**.** *If* $\mathbf{fappend}\ (v_1, v_2, v_3)$ *and* $\mathbf{fappend}\ (v_3, v_4, v_5)$*, then* $\mathbf{fappend}\ (v_2, v_3, v_6)$ *and* $\mathbf{fappend}\ (v_1, v_6, v_5)$ *for some list* v_6.

Now with Lemma 2 and Lemma 1 we can easily show that any expression e can be evaluated to some value v and the evalution is deterministic.

Theorem 5 (Evaluation Existence)**.** *For any* e*, there must exist a derivation of* $e \downarrow v$.

Theorem 6 (Evalutation Determinacy)**.** *If* $e \downarrow v$ *and* $e \downarrow v'$*, then* $v = v'$.

2.4 Equational system in Mini-List

We use the symbol $\sim$ to denote that two expressions in Mini-List are *provably equal* or *convertible*. Two expressions are considered *convertible* if they can be transformed to each other by the rules in Figure 4.

Judgment $\boxed{e \sim e'}$:

$$\textsc{Eq-Ref}: \frac{}{e \sim e} \qquad \textsc{Eq-Sym}: \frac{e' \sim e}{e \sim e'} \qquad \textsc{Eq-Trans}: \frac{e_1 \sim e' \qquad e' \sim e_2}{e_1 \sim e_2}$$

$$\textsc{Eq-S}: \frac{e \sim e'}{\mathbf{s}\ e \sim \mathbf{s}\ e'} \qquad \textsc{Eq-Cons}: \frac{e_1 \sim e_1' \qquad e_2 \sim e_2'}{e_1 : e_2 \sim e_1' : e_2'} \qquad \textsc{Eq-Append}: \frac{e_1 \sim e_1' \qquad e_2 \sim e_2'}{e_1 +\!\!+ e_2 \sim e_1' +\!\!+ e_2'}$$

$$\textsc{Eq-Append-Nil-L}: \frac{}{\mathbf{nil} +\!\!+ e \sim e} \qquad \textsc{Eq-Append-Nil-R}: \frac{}{e +\!\!+ \mathbf{nil} \sim e}$$

$$\textsc{Eq-Append-Cons}: \frac{}{(e_1 : e_2) +\!\!+ e_3 \sim e_1 : (e_2 +\!\!+ e_3)}$$

$$\textsc{Eq-Append-Asso}: \frac{}{(e_1 +\!\!+ e_2) +\!\!+ e_3 \sim e_1 +\!\!+ (e_2 +\!\!+ e_3)}$$

Figure 4: Equational system of Mini-List

The translation of the equational system in Twelf is simple, so we skip it here.

3 Equation Soundness

We introduce the symbol "==" to denote that two expressions e and e' are *semantically equal*, i.e., $\mathbf{e} == \mathbf{e}'$. Two expressions are semantically equal iff they evaluate to the same value.

Definition 7. *For any v, if $e \downarrow v$ and $e' \downarrow v$, then $e == e'$.*

Now we shall show that if two expressions are provably equal, then they are also semantically equal.

Theorem 8 (Equation Soundness)**.** *If $e \sim e'$, then $e == e'$.*

This theorem is equivalent to the following two lemmas if we replace $e == e'$ with its definition and set one of the clauses as a premise.

Lemma 9 (Equation Soundness Part 1)**.** *If $e \sim e'$, and $e \downarrow v$, then $e' \downarrow v$.*

Lemma 10 (Equation Soundness Part 2)**.** *If $e \sim e'$, and $e' \downarrow v$, then $e \downarrow v$.*

We only need to prove one of these lemmas since they are symmetric.

3.1 Paper proof

We prove Equation Soundness Part 1.

Proof. Let $\mathcal{Q}$ be the derivation of $e \sim e'$, $\mathcal{E}$ of $e \downarrow v$. We shall show that there is a deriation $\mathcal{E}'$ of $e' \downarrow v$. By induction on $\mathcal{Q}$:

- Case $\mathcal{Q} = \dfrac{}{e \sim e}$, so $e' = e$. We immediately obtain that $e' \downarrow v$.

- Case $\mathcal{Q} = \dfrac{\begin{array}{c}\mathcal{Q}_0\\ e' \sim e\end{array}}{e \sim e'}$.

  By Theroem 5 the existence of evaluation, there exists a derivation $\mathcal{E}_0$ of $e' \downarrow v'$. By IH on $\mathcal{Q}_0$ with $\mathcal{E}_0$, we get a derivation $\mathcal{E}_0'$ of $e \downarrow v'$.
  By the determinacy of evalutaion on $\mathcal{E}_0'$ with $\mathcal{E}$, we obtain $v' = v$.
  Then we obtain $\mathcal{E}'$ by replacing v' with v in $\mathcal{E}_0$.

- Case $\mathcal{Q} = \dfrac{\begin{array}{cc}\mathcal{Q}_1 & \mathcal{Q}_2\\ e_1 \sim e_1' & e_1' \sim e_2\end{array}}{e_1 \sim e_2}$.

  By IH on $\mathcal{Q}_1$ with $\mathcal{E}$, we obtain the derivation $\mathcal{E}_1$ of $e_1' \downarrow v$.
  Then by IH on $\mathcal{Q}_2$ with $\mathcal{E}_1$, we obtain $\mathcal{E}'$.

- Case $\mathcal{Q} = \dfrac{\begin{array}{c}\mathcal{Q}_0\\ e_0 \sim e_0'\end{array}}{\mathbf{s}\ e_0 \sim \mathbf{s}\ e_0'}$, so $e = \mathbf{s}\ e_0$ and $e' = \mathbf{s}\ e_0'$.

  $\mathcal{E}$ must have the shape:
  $\mathcal{E} = \dfrac{\begin{array}{c}\mathcal{E}_0\\ e_0 \downarrow v_0\end{array}}{\mathbf{s}\ e_0 \downarrow \mathbf{s}\ v_0}$
  By IH on $\mathcal{Q}_0$ with $\mathcal{E}_0$, we obtain the derivation $\mathcal{E}_0'$ of $e_0' \downarrow v_0$.
  Then we construct $\mathcal{E}'$ by Eval-S:
  $\mathcal{E}' = \dfrac{\begin{array}{c}\mathcal{E}_0'\\ e_0' \downarrow v_0\end{array}}{\mathbf{s}\ e_0' \downarrow \mathbf{s}\ v_0}$.

- Case $\mathcal{Q} = \dfrac{\begin{array}{cc}\mathcal{Q}_1 & \mathcal{Q}_2\\ e_1 \sim e_1' & e_2 \sim e_2'\end{array}}{e_1 : e_2 \sim e_1' : e_2'}$, so $e = e_1 : e_2$, and $e' = e_1' : e_2'$.

  $\mathcal{E}$ must have the shape:

  $$\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2\\ e_1 \downarrow v_1 & e_2 \downarrow v_2\end{array}}{e_1 : e_2 \downarrow v_1 : v_2}$$

  By IH on $\mathcal{Q}_1$ with $\mathcal{E}_1$, we get a derivation $\mathcal{E}_1'$ of $e_1' \downarrow v_1$.
  By IH on $\mathcal{Q}_2$ with $\mathcal{E}_2$, we get another derivaiton $\mathcal{E}_2'$ of $e_2' \downarrow v_2$.
  Then we construct $\mathcal{E}'$ by Eval-Cons:

  $$\mathcal{E}' = \dfrac{\begin{array}{cc}\mathcal{E}_1' & \mathcal{E}_2'\\ e_1' \downarrow v_1 & e_2' \downarrow v_2\end{array}}{e_1' : e_2' \downarrow v_1 : v_2}$$

- Case $\mathcal{Q} = \dfrac{\begin{array}{cc}\mathcal{Q}_1 & \mathcal{Q}_2\\ e_1 \sim e_1' & e_2 \sim e_2'\end{array}}{e_1 +\!\!+ e_2 \sim e_1' +\!\!+ e_2'}$, so $e = e_1 +\!\!+ e_2$, and $e' = e_1' +\!\!+ e_2'$.

  $\mathcal{E}$ must have the shape:

  $$\mathcal{E} = \dfrac{\begin{array}{ccc}\mathcal{E}_1 & \mathcal{E}_2 & \mathcal{F}\\ e_1 \downarrow v_1 & e_2 \downarrow v_2 & \mathbf{fappend}\ (v_1, v_2, v)\end{array}}{e_1 +\!\!+ e_2 \downarrow v}$$

  By IH on $\mathcal{Q}_1$ with $\mathcal{E}_1$, we get a derivation $\mathcal{E}_1'$ of $e_1' \downarrow v_1$.

By IH on $\mathcal{Q}_2$ with $\mathcal{E}_2$, we get another derivaiton $\mathcal{E}_2'$ of $e_2' \downarrow v_2$.
Then we construct $\mathcal{E}'$ by Eval-Append:

$$\mathcal{E}' = \dfrac{\begin{array}{ccc}\mathcal{E}_1' & \mathcal{E}_2' & \mathcal{F} \\ e_1' \downarrow v_1 & e_2' \downarrow v_2 & \textbf{fappend}\ (v_1, v_2, v)\end{array}}{e_1' +\!+ e_2' \downarrow v}$$

- Case $\mathcal{Q} = \dfrac{}{\textbf{nil} +\!+ e_1 \sim e_1}$, so $e = \textbf{nil} +\!+ e_1$, and $e' = e_1$.

  $\mathcal{E}$ must have the shape:

$$\mathcal{E} = \dfrac{\dfrac{}{\textbf{nil} \downarrow \textbf{nil}} \quad \begin{array}{cc}\mathcal{E}' & \mathcal{F} \\ e_1 \downarrow v & \textbf{fappend}\ (\textbf{nil}, v, v)\end{array}}{\textbf{nil} +\!+ e_1 \downarrow v}$$

  Thus $\mathcal{E}$ already includes $\mathcal{E}'$ and we are done.

- Case $\mathcal{Q} = \dfrac{}{e_1 +\!+ \textbf{nil} \sim e_1}$, so $e = e_1 +\!+ \textbf{nil}$, and $e' = e_1$.

  This case is analogous to Eq-Append-Nil-L (the last case) since the function **fappend** has the property **fappend** $(v, \textbf{nil}, v)$ by Lemma 3 Fappend Nil Right.

- Case $\mathcal{Q} = \dfrac{}{(e_1 : e_2) +\!+ e_3 \sim e_1 : (e_2 +\!+ e_3)}$, so $e = (e_1 : e_2) +\!+ e_3$, and $e' = e_1 : (e_2 +\!+ e_3)$.

  By the rules Eval-Append and Eval-Cons, $\mathcal{E}$ must look like:

$$\mathcal{E} = \dfrac{\dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2 \\ e_1 \downarrow v_1 & e_2 \downarrow v_2\end{array}}{e_1 : e_2 \downarrow v_1 : v_2} \quad \begin{array}{cc}\mathcal{E}_3 & \mathcal{F} \\ e_3 \downarrow v_3 & \textbf{fappend}\ (v_2, v_3, v') \\ & \hline & \textbf{fappend}\ (v_1 : v_2, v_3, v_1 : v')\end{array}}{(e_1 : e_2) +\!+ e_3 \downarrow v_1 : v'}$$

  Then we can construct $\mathcal{E}'$:

$$\mathcal{E}' = \dfrac{\begin{array}{cc}\mathcal{E}_1 & \begin{array}{ccc}\mathcal{E}_2 & \mathcal{E}_3 & \mathcal{F} \\ e_2 \downarrow v_2 & e_3 \downarrow v_3 & \textbf{fappend}\ (v_2, v_3, v')\end{array} \\ e_1 \downarrow v_1 & \hline & e_2 +\!+ e_3 \downarrow v'\end{array}}{e_1 : (e_2 +\!+ e_3) \downarrow v_1 : v'}$$

- Case $\mathcal{Q} = \dfrac{}{(e_1 +\!+ e_2) +\!+ e_3 \sim e_1 +\!+ (e_2 +\!+ e_3)}$, so $e = (e_1 +\!+ e_2) +\!+ e_3$, and $e' = e_1 +\!+ (e_2 +\!+ e_3)$.

  By the rule Eval-Append, $\mathcal{E}$ must look like:

$$\mathcal{E} = \dfrac{\begin{array}{ccc}\mathcal{E}_1 & \mathcal{E}_2 & \mathcal{F}_1 \\ e_1 \downarrow v_1 & e_2 \downarrow v_2 & \textbf{fappend}\ (v_1, v_2, v') \\ & & \hline & & e_1 +\!+ e_2 \downarrow v'\end{array} \quad \begin{array}{cc}\mathcal{E}_3 & \mathcal{F}_2 \\ e_3 \downarrow v_3 & \textbf{fappend}\ (v', v_3, v) \\ & \hline & (e_1 +\!+ e_2) +\!+ e_3 \downarrow v\end{array}}$$

  By Lemma 4 Fappend Associativity on $\mathcal{F}_1$ with $\mathcal{F}_2$, we get $\mathcal{F}_3$ and $\mathcal{F}_4$ for some list v''. Then by the rule Eval-Append we can construct $\mathcal{E}'$:

$$\mathcal{E}' = \dfrac{\mathcal{E}_1 \quad \dfrac{\mathcal{E}_2 \quad \mathcal{E}_3 \quad \mathcal{F}_3}{e_2 \downarrow v_2 \quad e_3 \downarrow v_3 \quad \textbf{fappend}\ (v_2, v_3, v'')} \quad \mathcal{F}_4}{e_1 \downarrow v_1 \qquad e_2 +\!\!+ e_3 \downarrow v'' \qquad \textbf{fappend}\ (v_1, v'', v)} \Big/ e_1 +\!\!+ (e_2 +\!\!+ e_3) \downarrow v$$

■

3.2 Implementation in Twelf

Before we translate this paper proof in Twelf, we first need to formalize the Evaluation Replacement lemma, although it is usually considered trivial in paper proof.

Lemma 11 (Evaluation Replacement)**.** *If $e \downarrow v$, and $v = v'$, then $e \downarrow v'$.*

We formalize this lemma in Twelf as follows.

```
eq-eval-val : eval E V -> eq-val V V' -> eval E V' -> type.
%mode eq-eval-val +EP +QV -EP'.
- : eq-eval-val EP eq-val/ EP.
%worlds () (eq-eval-val _ _ _).
%total {} (eq-eval-val _ _ _).
```

Then we can translate the paper proof as follows to check it.

```
% -- translation of the lemma
eq-sound : eq E E' -> eval E V -> eval E' V -> type.
%mode eq-sound +Q +EP -EP'.

% -- case Eq-Ref
eq-sound/ref : eq-sound eq/ EP EP.

% -- case Eq-Sym
eq-sound/sym : eq-sound (eq-sym Q) EP EP'
            <- eval-exists E' EP1'  % -- by Evaluation Existence lemma
            <- eq-sound Q EP1' EP1
            <- eval-determ EP1 EP QV  % -- by Evalution Determinacy lemma
            <- eq-eval-val EP1' QV EP'. % -- by Evaluation Replacement lemma

% -- case Eq-Trans
eq-sound/trans : eq-sound (eq-trans Q1 Q2) EP EP'
               <- eq-sound Q1 EP EP1'
               <- eq-sound Q2 EP1' EP'.

% -- case Eq-S
eq-sound/s : eq-sound (eq-s Q1) (eval/s EP) (eval/s EP')
            <- eq-sound Q1 EP EP'.

% -- case Eq-Cons
eq-sound/cons : eq-sound
               (eq-cons Q1 Q2)
               (eval/cons EP2 EP1)
               (eval/cons EP2' EP1')
               <- eq-sound Q1 EP1 EP1'
               <- eq-sound Q2 EP2 EP2'.

% -- case Eq-Append
eq-sound/append : eq-sound
                  (eq-append Q1 Q2)
                  (eval/append F EP2 EP1)
                  (eval/append F EP2' EP1')
                  <- eq-sound Q1 EP1 EP1'
                  <- eq-sound Q2 EP2 EP2'.

% -- case Eq-Append-Nil-L
eq-sound/append/nil-l : eq-sound
```

```
                          eq-append-nil-l
                          (eval/append fappend/nil EP eval/nil)
                          EP.

% -- case Eq-Append-Nil-R
eq-sound/append/nil-r : eq-sound
                          eq-append-nil-r
                          (eval/append F eval/nil EP1)
                          EP
                          <- fappend-nil-r F QV  % --by the lemma Fappend Nil
                              Right
                          <- eq-eval-val EP1 QV EP.

% -- case Eq-Append-Cons
eq-sound/append/cons : eq-sound
                          eq-append-cons
                          (eval/append (fappend/cons F) EP3 (eval/cons EP2 EP1))
                          (eval/cons (eval/append F EP3 EP2) EP1).

% -- case Eq-Append-Asso
eq-sound/append/asso : eq-sound
                          eq-append-asso
                          (eval/append F2 EP3 (eval/append F1 EP2 EP1))
                          (eval/append F4 (eval/append F3 EP3 EP2) EP1)
                          <- fappend-asso F1 F2 F3 F4. % -- by the lemma Fappend
                              Associativity

%worlds () (eq-sound _ _ _).
%total Q (eq-sound Q _ _ ).
```

4 Evaluation Completeness

The converse of Theorem 8 Equation Soundness also holds.

Theorem 12 (Equation Completeness)**.** *If* $e == e'$*, then* $e \sim e'$*.*

This theorem says that if both e and e' evaluate to the same value v, then there must exist a deriavation of $e \sim e'$. Since $e == v$ is evident by Definition 7, we only need to show the following Lemma 13. Because by this lemma we can also obtain $e' \sim v$, then by the symmetry and transitivity rules of the equational system, we will get $e \sim e'$.

Lemma 13. *If* $e \downarrow v$*, then* $e \sim v$*.*

4.1 Paper proof

Before we prove Lemma 13, we first show the following lemma.

Lemma 14 (Fappend Completeness)**.** *If* **fappend** (v_1, v_2, v_3) *(by some derivation* $\mathcal{F}$*), then* $(v_1 +\!\!+ v_2) \sim v_3$ *(by some derivation* $\mathcal{Q}$ *).*

Proof. By induction on $\mathcal{F}$:

- Case $\mathcal{F} = \dfrac{}{\textbf{fappend}\ (\textbf{nil}\ , v, v)}$, so $v_1 = \textbf{nil}, v_2 = v, v_3 = v$.

  By the rule Eq-Append-Nil-L, we immeditately get $v_1 +\!\!+ v_2 \sim v_3$.

- Case $\mathcal{F} = \dfrac{\begin{array}{c}\mathcal{F}_0 \\ \textbf{fappend}\ (v_2', v_3', v)\end{array}}{\textbf{fappend}\ ((v_1' : v_2'), v_3', (v_1' : v))}$, so $v_1 = (v_1' : v_2'), v_2 = v_3'$, and $v_3 = (v_1' : v)$.

  By IH on $\mathcal{F}_0$, we obatin a derivation $\mathcal{Q}_0$ of $v_2' +\!\!+ v_3' \sim v$.
  Then by the rule Eq-Cons on $\mathcal{Q}_0$ and the equation $v_1' = v_1'$, we get $\mathcal{Q}'$ of $v_1' : (v_2' +\!\!+ v_3') \sim v_1' : v$.
  By the rule Eq-Append-Cons we also have $\mathcal{Q}''$ of $(v_1' : v_2') +\!\!+ v_3' \sim v_1' : (v_2' +\!\!+ v_3')$.
  Finally, by the transtivity of equations on $\mathcal{Q}''$ with $\mathcal{Q}'$ we get $\mathcal{Q}$ of $(v_1' : v_2') +\!\!+ v_3' \sim v_1' : v$.

■

Now we start to prove Lemma 13.

Proof. Let $\mathcal{E}$ be the derivation of $e \downarrow v$. We shall show that there is a deriation $\mathcal{Q}$ of $e \sim v$. By induction on $\mathcal{E}$:

- Case $\mathcal{E} = \dfrac{}{\mathbf{z} \downarrow \mathbf{z}}$, so both e and v are $\mathbf{z}$, then immediately $\mathbf{z} \sim \mathbf{z}$ by Eq-Ref.

- Case $\mathcal{E} = \dfrac{\begin{array}{c}\mathcal{E}_0 \\ e_0 \downarrow v_0\end{array}}{\mathbf{s}\ e_0 \downarrow \mathbf{s}\ v_0}$, so $e = \mathbf{s}\ e_0$ and $v = \mathbf{s}\ v_0$.

  By IH on $\mathcal{E}_0$, we obtain a derivation $\mathcal{Q}_0$ of $e_0 \sim v_0$. Then we can construct $\mathcal{Q}$ by Eq-S:

  $$\mathcal{Q} = \dfrac{\begin{array}{c}\mathcal{Q}_0 \\ e_0 \sim v_0\end{array}}{\mathbf{s}\ e_0 \sim \mathbf{s}\ v_0}$$

- The case for $\mathcal{E} =$ Eval-Nil is analogous to the case $\mathcal{E} =$ Eval-Z .

- Case $\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2 \\ e_1 \downarrow v_1 & e_2 \downarrow v_2\end{array}}{e_1 : e_2 \downarrow v_1 : v_2}$, so $e = e_1 : e_2$, $v = v_1 : v_2$.

  By IH on $\mathcal{E}_1$, we obtain a derivation $\mathcal{Q}_1$ of $e_1 \sim v_1$. By IH on $\mathcal{E}_2$, we obtain a derivation $\mathcal{Q}_2$ of $e_2 \sim v_2$. Then we construct $\mathcal{Q}$ by Eq-Cons: $\mathcal{Q} = \dfrac{\begin{array}{cc}\mathcal{Q}_1 & \mathcal{Q}_2 \\ e_1 \sim v_1 & e_2 \sim v_2\end{array}}{e_1 : e_2 \sim v_1 : v_2}$

- Case $\mathcal{E} = \dfrac{\begin{array}{ccc}\mathcal{E}_1 & \mathcal{E}_2 & \mathcal{F} \\ e_1 \downarrow v_1 & e_2 \downarrow v_2 & \textbf{fappend}\ (v_1, v_2, v)\end{array}}{e_1 +\!\!+ e_2 \downarrow v}$, so $e = e_1 +\!\!+ e_2$.

  By IH on $\mathcal{E}_1$, we obtain a derivation $\mathcal{Q}_1$ of $e_1 \sim v_1$. By IH on $\mathcal{E}_2$, we obtain a derivation $\mathcal{Q}_2$ of $e_2 \sim v_2$. Then by Lemma 14 on $\mathcal{F}$, we obatin a derivation $\mathcal{Q}'$ of $v_1 +\!\!+ v_2 \sim v$. By the rule Eq-Append, we obtain another derivation $\mathcal{Q}''$ of $e_1 +\!\!+ e_2 \sim v_1 +\!\!+ v_2$:

  $$\mathcal{Q}'' = \dfrac{\begin{array}{cc}\mathcal{Q}_1 & \mathcal{Q}_2 \\ e_1 \sim v_1 & e_2 \sim v_2\end{array}}{e_1 +\!\!+ e_2 \sim v_1 +\!\!+ v_2}$$

  Now we get $\mathcal{Q}$ by Eq-Trans: $\mathcal{Q} = \dfrac{\begin{array}{cc}\mathcal{Q}'' & \mathcal{Q}' \\ e_1 +\!\!+ e_2 \sim v_1 +\!\!+ v_2 & v_1 +\!\!+ v_2 \sim v\end{array}}{e_1 +\!\!+ e_2 \sim v}$

■

4.2 Implementation in Twelf

We formalize the paper proof in Twelf as follows.

```
% --translation of the lamme
eq-evcomp : eval E V -> val2exp V E' -> eq E E' -> type.
%mode eq-evcomp +EP -VE -Q.

% --case Eval-Z
eq-evcomp/z : eq-evcomp eval/z vz2ez eq/.

% --case Eval-S
```

```
eq-evcomp/s : eq-evcomp (eval/s EP) (vs2es VE) (eq-s Q)
         <- eq-evcomp EP VE Q.

% --case Eval-Nil
eq-evcomp/nil : eq-evcomp eval/nil vnil2enil eq/.

% --case Eval-Cons
eq-evcomp/cons : eq-evcomp (eval/cons EP2 EP1)
                 (vcons2econs VE2 VE1)
                 (eq-cons Q1 Q2)
                 <- eq-evcomp EP1 VE1 Q1
                 <- eq-evcomp EP2 VE2 Q2.

% --case Eval-Append
eq-evcomp/append : eq-evcomp
                 (eval/append F EP2 EP1)
                 VE
                 (eq-trans
                 (eq-append Q1 Q2) Q')
                 <- eq-evcomp EP1 VE1 Q1
                 <- eq-evcomp EP2 VE2 Q2
                 <- fappend-comp F VE1 VE2 VE Q'.  % --by Fappend Completeness
                     Lemma

%worlds () (eq-evcomp _ _ _).
%total EP (eq-evcomp EP _ _).
```

5 Extend Mini-List with let-binding

We now extend Mini-List with **let** -binding to make it a higher-order language.

5.1 Extended Syntax

The type system and values will be the same as before the extension. The extended syntax of the expressions in Mini-List is as follows.

$$e ::= \mathbf{z} \mid \mathbf{s}\ e \mid \mathbf{nil} \mid e_1 : e_2 \mid e_1 +\!\!+ e_2 \mid \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \mid x$$

5.2 Extended operational semantics

Before we extend the operational semantics for **let** -binding, we first need to define an evaluation context Δ for the new operational semantics to denote the evaluations of variables in the context. We define $\Delta = [x_1 \downarrow v_1, ..., x_n \downarrow v_n]$ where all the x_i are distinct, and $\Delta \vdash e \downarrow v$ means "in context Δ, e evalutes to v.

The other evaluation rules will be the same as before the extension because their evaluation context is empty. So we just show the evaluation rule for **let** -binding.

Judgment $\boxed{\Delta \vdash e \downarrow v}$:

$$\textsc{Eval-Let}: \quad \frac{\Delta \vdash e_1 \downarrow v_1 \qquad \Delta[x \downarrow v_1] \vdash e_2 \downarrow v}{\Delta \vdash \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \downarrow v}$$

Figure 5: Evaluation rule for **let** -binding in the new operatinal semantics

We can check that the properties of the old operational semantics, such as evaluation existence and determinacy, still hold for the new one.

5.3 Extended equational system

Similarly, in the new equational system, we also need to define an equation context Γ to denote the convertible relation between variables in the context.

We add the following rules to the equational system in Mini-List.

Judgement $\boxed{\Gamma \vdash e \sim e'}$:

$$\textsc{Eq-Let}: \quad \frac{\Gamma \vdash e_1 \sim e_1' \qquad \Gamma[x \sim x'] \vdash e_2 \sim e_2'}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \sim \textbf{let}\ x' = e_1'\ \textbf{in}\ e_2'}$$

$$\textsc{Eq-Let-Subst}: \quad \frac{}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \sim e_2[e_1/x]}$$

Figure 6: Extended rules for the equational system in Mini-list

In the rule Eq-Let-Subst, $e_2[e_1/x]$ means we substitute e_1 for x in e_2.

5.4 Extended equation soundness

Since we add two rules for the equational system, we should add two cases in the proof of equation soundness as well.

Proof.

- Case $\mathcal{Q} = \dfrac{\mathcal{Q}_1 \qquad \mathcal{Q}_2}{[] \vdash e_1 \sim e_1' \qquad [x \sim x'] \vdash e_2 \sim e_2'}{[] \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \sim \textbf{let}\ x' = e_1'\ \textbf{in}\ e_2'}$, so e is $\textbf{let}\ x = e_1\ \textbf{in}\ e_2$ and e' is $\textbf{let}\ x' = e_1'\ \textbf{in}\ e_2'$.

  Then $\mathcal{E}$ must have the shape: $\mathcal{E} = \dfrac{\mathcal{E}_1 \qquad \mathcal{E}_2}{[] \vdash e_1 \downarrow v_1 \qquad [x \downarrow v_1] \vdash e_2 \downarrow v}{[] \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \downarrow v}$

  By IH on $\mathcal{Q}_1$ with $\mathcal{E}_1$ we obtain $\mathcal{E}_1'$ of $e_1' \downarrow v_1$.
  By IH on the context of $\mathcal{Q}_2$ with the context of $\mathcal{E}_2$, we get a derivation $\mathcal{E}_x$ of $x' \downarrow v_1$.
  By IH on $\mathcal{Q}_2$ with $\mathcal{E}_2$, and in the context $x' \downarrow v_1$, we get $\mathcal{E}_2'$ of $[x' \downarrow v_1] \vdash e_2' \downarrow v$.

  Then we can construct $\mathcal{E}' = \dfrac{\mathcal{E}_1' \qquad \mathcal{E}_2'}{[] \vdash e_1' \downarrow v_1 \qquad [x' \downarrow v_1] \vdash e_2' \downarrow v}{[] \vdash \textbf{let}\ x' = e_1'\ \textbf{in}\ e_2' \downarrow v}$

- Case $\mathcal{Q} = \dfrac{}{[] \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \sim e_2[e_1/x]}$, so e is $\textbf{let}\ x = e_1\ \textbf{in}\ e_2$, e' is $e_2[e_1/x]$.
  This case is immediate. Because in the derivation $\mathcal{E}$, we get v by first replacing x with the value v_1 of e_1 in e_2, then evaluating e_2. While in $\mathcal{E}'$ we first replace x with e_1 in e_2, then evaluate e_2. So the evaluation results are the same.

■

The translation for the added proof cases in Twelf is as follows (all the other properties of this language must still hold).

```
% -- case Eq-let
eq-sound/let : eq-sound
               (eq-let Q1 Q2)
               (eval/let EP2 EP1)
               (eval/let EP2' EP1')
               <- eq-sound Q1 EP1 EP1'
               <- {x} {ex: eval x V1}  {qx : eq x x}
                  {_ : eval-determ ex ex eq-val/}
                  {_ : eval-exists x ex}
                  eq-sound qx ex ex
                  -> eq-sound (Q2 x x qx) (EP2 x ex) (EP2' x ex).
```

```
% -- case Eq-Let-Subst
eq-sound/let/subst : eq-sound
                     eq-let-subst
                     (eval/let EP2 EP1)
                     (EP2 _ EP1).
```

Note that in the translation of the case EQ-LET, we use the same variable name x (rather than x and x') in the equation context to simplify the formalization in Twelf. Also, the evaluation determinacy and existence are proved separately in our paper proof and not necessarily checked again for the context. But in Twelf they must be added to ensure they still hold for the variables in the context.

5.5 Extended equation completeness

The extended proof case for **let** -binding is as follows.

Proof. • Case $\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2\\ [] \vdash e_1 \downarrow v_1 & [x \downarrow v_1] \vdash e_2 \downarrow v\end{array}}{[] \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \downarrow v}$, so e is **let** $x = e_1$ **in** e_2.

By the rule EQ-LET-SUBST, we have $\mathcal{Q}'$ of **let** $x = e_1$ **in** $e_2 \sim e_2[e_1/x]$. So we only need to show that there is a derivation $\mathcal{Q}''$ of $e_2[e_1/x] \sim v$, then by EQ-TRANS on $\mathcal{Q}'$ with $\mathcal{Q}''$ we will get $\mathcal{Q}$.
By IH on $\mathcal{E}_1$ we get a derivation $\mathcal{Q}_1$ of $e_1 \sim v_1$.
By IH on the context of $\mathcal{E}_2$ we get $\mathcal{Q}_x$ of $x \sim v_1$. Then by IH on $\mathcal{E}_2$ in the context $x \sim v_1$, we get $\mathcal{Q}_2$ of $[x \sim v_1] \vdash e_2 \sim v$. Since we already have Q_1, substituting e_1 for x in e_2, which satisfies the context of $\mathcal{Q}_2$, will give us $e_2 \sim v$ and we are done.

■

The translation of this case in Twelf is as follows.

```
% -- case Eval-Let
eq-evcomp/let : eq-evcomp
            (eval/let EP2 EP1)
            (VE2 E1)
            (eq-trans eq-let-subst (Q2 E1 Q1))
            <- eq-evcomp EP1 VE1 Q1
            <- {x} {ex:eval x V1}  {qx: eq x E1'}
               eq-evcomp ex VE1 qx
               -> eq-evcomp (EP2 x ex) (VE2 x) (Q2 x qx).
```