1 INTERFACE AND SHARED INTERPRETER FOR PCF

```
class Arrow c => IsVal v c | c -> v where
  succ :: c v v
  pred :: c v v
  zero :: c () v
  ifZero :: c x v -> c y v -> c (v, (x, y)) v

class Arrow c => IsClosure v env c | c -> env, c -> v where
  closure :: c (Expr, env) v
  applyClosure :: c Expr v -> c (v, v) v

class Arrow c => ArrowEnv x y env c | c -> x, c -> y, c -> env where
  lookup :: c x (Maybe y)
  getEnv :: c () env
  extendEnv :: c (x,y,env) env
  localEnv :: c a b -> c (env,a) b

class Arrow c => ArrowFail e c | c -> e where
  failA :: c e x

data Expr
  = Var Text
  | Lam Text Expr
  | App Expr Expr
  | Zero
  | Succ Expr
  | Pred Expr
  | IfZero Expr Expr Expr
  | Y Expr

eval :: (IsVal v c, IsClosure v env c, ArrowChoice c, ArrowFix Expr v c,
  ArrowEnv Text v env c, ArrowFail String c) => c Expr v
eval = fixA $ \ev -> proc e0 -> case e0 of
  Var x -> do
    m <- lookup -< x
    case m of
      Just v -> returnA -< v
      Nothing -> failA -< printf "Variable␣\"%s\"␣not␣bound" (unpack x)
  Lam x e -> do
    env <- getEnv -< ()
```

```
    closure -< (Lam x e, env)
  App e1 e2 -> do
    fun <- ev -< e1
    arg <- ev -< e2
    applyClosure ev -< (fun, arg)
  Zero -> zero -< ()
  Succ e -> do
    v <- ev -< e
    succ -< v
  Pred e -> do
    v <- ev -< e
    pred -< v
  IfZero e1 e2 e3 -> do
    v1 <- ev -< e1
    ifZero ev ev -< (v1, (e2, e3))
  Y e -> do
    fun <- ev -< e
    env <- getEnv -< ()
    arg <- closure -< (Y e, env)
    applyClosure ev -< (fun, arg)
```

2 SOUNDNESS OF THE ABSTRACT INTERPRETER FOR PCF

2.1 Soundness of Value Operations

```
class Arrow c => IsVal v c | c -> v where
  succ :: c v v
  pred :: c v v
  zero :: c () v
  ifZero :: c x v -> c y v -> c (v, (x, y)) v

class Arrow c => IsClosure v env c | c -> env, c -> v where
  closure :: c (Expr, env) v
  applyClosure :: c ((Expr,env),v) v -> c (v, v) v

data Closure = Closure Expr Env
data Val = NumVal Int | ClosureVal Closure

data Closure^ = Closure Expr Env
data Val^ = Bot | NumVal^ Interval | ClosureVal^ (Pow Closure^) | Top
```

Definition 1 (Ordering on $\widehat{\mathtt{Val}}$)**.** The ordering on abstract values is defined as follows:

$$\begin{aligned}
\mathtt{Bot} &\sqsubseteq x \\
x &\sqsubseteq \mathtt{Top} \\
\widehat{\mathtt{NumVal}}\ [i,j] &\sqsubseteq \widehat{\mathtt{NumVal}}\ [i',j'] \ \text{ iff } \ i' \leq i \wedge j \leq j' \\
\widehat{\mathtt{ClosureVal}}\ X &\sqsubseteq \widehat{\mathtt{ClosureVal}}\ Y \ \text{ iff } \ X = \varnothing \vee (\forall y \in Y.\, \exists x \in X.\, x \sqsubseteq y)
\end{aligned}$$

Definition 2. We define the powerset lifting $\mathcal{P} : \mathtt{Interp} \to \mathtt{Interp}_{\mathcal{P}}$ on the underlying function space of Interp.

Definition 3. We define a Galois connection $\alpha : \mathtt{Interp}_{\mathcal{P}} \leftrightarrows \widehat{\mathtt{Interp}} : \gamma$ between the power-set lifted concrete arrow and the abstract interpreter arrow $\widehat{\mathtt{Interp}}$ on the underlying function space.

Lemma 4 (succ is sound)**.** To show $\alpha(\mathcal{P}(\mathsf{succ})) \sqsubseteq \widehat{\mathsf{succ}}$ where succ and $\widehat{\mathsf{succ}}$ are defined as

```
succ = proc x -> case x of
  NumVal n -> returnA -< NumVal (n + 1)
  ClosureVal _ -> failA -< "Expected a number as argument for 'succ'"

succ^ = proc x -> case x of
  Top -> returnA -< Top
  NumVal^ (i,j) -> returnA -< NumVal (i+1,j+1)
  ClosureVal^ _ -> failA -< "Expected a number as argument for 'succ'"
  Bot -> returnA -< Bot
```

Proof. We distinguish four cases for $\widehat{x}$ in

$$X \xrightarrow{\ x\ } \mathtt{Val} \xrightarrow{\ \mathcal{P}\ \mathsf{succ}\ } \mathtt{Val}$$

$$X \xrightarrow{\ \widehat{x}\ } \mathtt{Val} \xrightarrow{\ \widehat{\mathsf{succ}}\ } \mathtt{Val},$$

- In case $\widehat{x} = \bot$, then $x \subseteq \varnothing = \gamma\bot$ and hence $\alpha(\mathcal{P}(\mathsf{succ}) \circ x) \sqsubseteq \alpha(\mathcal{P}(\mathsf{succ}) \circ \varnothing) \sqsubseteq \alpha\varnothing \sqsubseteq \bot = \widehat{\mathsf{succ}} \circ \bot$.
- In case $\widehat{x} = \top$, then $\alpha(\mathcal{P}(\mathsf{succ}) \circ x) \sqsubseteq \top = \widehat{\mathsf{succ}} \circ \top$.
- In case $\widehat{x} = \widehat{\mathtt{ClosureVal}}$, then $x \subseteq \mathcal{P}(\mathtt{ClosureVal}) = \gamma\widehat{\mathtt{ClosureVal}}$ and hence $\alpha(\mathcal{P}(\mathsf{succ}) \circ x) \sqsubseteq \alpha(\mathcal{P}(\mathsf{succ}) \circ \mathcal{P}(\mathtt{ClosureVal})) = \alpha(\mathcal{P}(\mathtt{failA})) = \mathtt{failA} = \widehat{\mathsf{succ}} \circ \widehat{\mathtt{ClosureVal}}$
- In case $\widehat{x} = \widehat{\mathtt{NumVal}}$, then $x \subseteq \mathcal{P}(\mathtt{NumVal}) = \gamma(\widehat{\mathtt{NumVal}})$ and $\mathsf{succ} \circ \mathtt{NumVal} = \mathtt{arr}\ (\lambda n.\, n+1)$ and $\widehat{\mathsf{succ}} \circ \widehat{\mathtt{NumVal}} = \mathtt{arr}\ (\lambda(i,j).\, (i+1, j+1))$. Let $X \subseteq \mathtt{Int}$ and $\alpha(X) = [i,j]$, then $\{n \mid i \leq n \leq j\} = \gamma[i,j]$. The previous statment tells us that all numbers in X are bounded by i and j, i.e., $i \leq \min X$ and $\max X \leq j$. It follows, $\alpha(\{n+1 \mid n \in X\}) = [\min\{n+1 \mid n \in X\}, \max\{n+1 \mid n \in X\}] \sqsubseteq [i+1, j+1]$ and hence $\alpha(\mathcal{P}(\mathtt{arr}\ (\lambda n.\, n+1))) \sqsubseteq \mathtt{arr}\ (\lambda(i,j).\, (i+1, j+1))$.

□

Lemma 5 (pred is sound)**.** To show $\alpha(\mathcal{P}(\mathsf{pred})) \sqsubseteq \widehat{\mathsf{pred}}$, where pred and $\widehat{\mathsf{pred}}$ are defined as

```
pred = proc x -> case x of
  NumVal n -> returnA -< NumVal (n - 1)
  ClosureVal _ -> failA -< "Expected a number as argument for 'pred'"

pred^ = proc x -> case x of
  Top -> returnA -< Top
  NumVal^ (i,j) -> returnA -< NumVal (i-1,j-1)
  ClosureVal^ _ -> failA -< "Expected a number as argument for 'pred'"
  Bot -> returnA -< Bot
```

Proof. Analogous to proof of succ. □

Lemma 6 (zero is sound)**.** To show $\alpha(\mathcal{P}(\mathsf{pred})) \sqsubseteq \widehat{\mathsf{pred}}$, where zero and $\widehat{\mathsf{zero}}$ are defined as

```
zero = arr $ const (NumVal 0)
zero^ = arr $ const (NumVal (0,0))
```

Proof. $\alpha(\{0\}) = [0, 0]$. □

Lemma 7 (ifZero is sound)**.** To show: $\alpha(\mathcal{P}(f)) \sqsubseteq \widehat{f}$ and $\alpha(\mathcal{P}(g)) \sqsubseteq \widehat{g}$ imply $\alpha(\mathcal{P}(\mathtt{ifZero}\ f\ g)) \sqsubseteq \widehat{\mathtt{ifZero}}\ \widehat{f}\ \widehat{g}$, where where ifZero and $\widehat{\mathtt{ifZero}}$ are defined as

```
ifZero f g = proc (v1, (x, y)) -> case v1 of
  NumVal 0 -> f -< x
  NumVal _ -> g -< y
  _ -> failA -< "Expected a number as condition for 'ifZero'"

ifZero^ f g = proc v -> case v of
  (Top, (x,y)) -> returnA -< Top
  (NumVal (i1,i2)), (x, y))
    | (i1, i2) == (0, 0) -> f -< x
    | i2 < 0 || 0 < i1 -> g -< y
    | otherwise -> (f -< x) ⊔ (g -< y)
  (ClosureVal _, _) -> failA -< "Expected a number as condition for 'ifZero'"
  (Bot, _) -> returnA -< Bot
```

Proof. We distinguish four cases for $\widehat{x}$ in

$$X \xrightarrow{\ x\ } \mathsf{Val} \times (A \times B) \xrightarrow{\ \mathcal{P}(\mathtt{ifZero}\ f\ g)\ } \mathsf{Val}$$

$$X \xrightarrow{\ \widehat{x}\ } \mathsf{Val} \times (A \times B) \xrightarrow{\ \widehat{\mathtt{ifZero}}\ \widehat{f}\ \widehat{g}\ } \mathsf{Val},$$

- The cases $\widehat{x} = \top \times (\widehat{a} \times \widehat{b})$, $\widehat{x} = \bot \times (\widehat{a} \times \widehat{b})$ and $\widehat{x} = \widehat{\texttt{ClosureVal}} \times (\widehat{a} \times \widehat{b})$ are analogous to cases in the proof of succ.
- In case $\widehat{x} = (\widehat{\texttt{NumVal}} \circ \widehat{i}) \times (\widehat{a} \times \widehat{b})$, then $x \subseteq \mathcal{P}((\texttt{NumVal} \circ i) \times (a \times b)) \sqsubseteq \gamma((\widehat{\texttt{NumVal}} \circ \widehat{i}) \times (\widehat{a} \times \widehat{b}))$: We distinguish three cases for $\widehat{i} : X' \to \texttt{Num}$, the input interval for $\widehat{\texttt{ifZero}}$.
  - In case $\widehat{i} = [0, 0]$, then $\mathcal{P}i \subseteq \{0\} = \gamma(\widehat{i})$. It follows

$$\begin{aligned}\alpha(\mathcal{P}(\texttt{ifZero}\ f\ g \circ ((\texttt{NumVal} \circ 0) \times (a \times b)))) &= \alpha(\mathcal{P}(f \circ a)) \sqsubseteq \widehat{f} \circ \alpha(\mathcal{P}(a)) \\ &\sqsubseteq \widehat{f} \circ \widehat{a} = \texttt{ifZero}\ f\ g \circ ((\widehat{\texttt{NumVal}} \circ [0,0]) \times (a \times b))\end{aligned}$$

  - The case $[0, 0] \not\sqsubseteq \widehat{i}$ is analogous to the previous case.
  - In case $[0, 0] \sqsubset \widehat{i}$,

$$\begin{aligned}\alpha(\mathcal{P}(\texttt{ifZero}\ f\ g \circ ((\texttt{NumVal} \circ i) \times (a \times b)))) &\sqsubseteq \alpha(\mathcal{P}(f \circ a)\ \cup\ \mathcal{P}(g \circ b)) \\ &\sqsubseteq \alpha(\mathcal{P}(f \circ a))\ \sqcup\ \alpha(\mathcal{P}(g \circ b)) \\ &\sqsubseteq (\widehat{f} \circ \alpha(\mathcal{P}(a)))\ \sqcup\ (\widehat{g} \circ \alpha(\mathcal{P}(b))) \\ &\sqsubseteq (\widehat{f} \circ \widehat{a})\ \sqcup\ (\widehat{g} \circ \widehat{b}) \\ &\sqsubseteq \widehat{\texttt{ifZero}}\ \widehat{f}\ \widehat{g} \circ ((\widehat{\texttt{NumVal}} \circ \widehat{i}) \times (\widehat{a} \times \widehat{b}))\end{aligned}$$

□

Lemma 8 (`closure` is sound)**.** To show $\alpha(\mathcal{P}(\texttt{closure})) \sqsubseteq \widehat{\texttt{closure}}$, where `closure` and $\widehat{\texttt{closure}}$ are defined as

```
closure = arr $ \(e, env) -> ClosureVal (Closure e env)
closure^ = arr $ \(e, env) -> ClosureVal^ (singleton (Closure^ e env))
```

Proof. Let $X \subseteq \texttt{Expr} \times \texttt{Env}$ and $\alpha(X) = (e, \widehat{\rho})$, then $X \subseteq \{(e, \rho) \mid \rho \in \gamma(\widehat{\rho})\} = \gamma(e, \widehat{\rho})$. We conclude

$$\begin{aligned}\alpha(\{(\texttt{Closure}\ e\ \rho) \mid (e, \rho) \in X\}) &\sqsubseteq \alpha(\{(\texttt{Closure}\ e\ \rho) \mid \rho \in \gamma(\widehat{\rho})\}) \\ &\sqsubseteq \{\widehat{\texttt{Closure}}\ e\ \alpha(\{\rho\}) \mid \rho \in \gamma(\widehat{\rho})\} \\ &\sqsubseteq \{\widehat{\texttt{Closure}}\ e\ \alpha(\gamma(\widehat{\rho}))\} \\ &\sqsubseteq \{\widehat{\texttt{Closure}}\ e\ \widehat{\rho}\}.\end{aligned}$$

□

Lemma 9 (`applyClosure` is sound)**.** To show: $\alpha(\mathcal{P}(f)) \sqsubseteq \widehat{\texttt{f}}$ implies $\alpha(\mathcal{P}(\texttt{applyClosure}\ f)) \sqsubseteq \widehat{\texttt{applyClosure}}\ \widehat{\texttt{f}}$, where `applyClosure` and $\widehat{\texttt{applyClosure}}$ are defined as

```
applyClosure f = proc (fun, arg) -> case fun of
  ClosureVal (Closure e env) -> f -< ((e,env),arg)
  NumVal _ -> failA -< "Expected a closure"
```

```
applyClosure f = proc (fun, arg) -> case fun of
  Top -> returnA -< Top
  ClosureVal cls -> lubA (proc (Closure e env,arg) -> f -< ((e,env),arg))
                      -< [ (c,arg) | c <- toList cls]
  NumVal _ -> failA -< "Expected a closure"
  Bot -> returnA -< Bot
```

Proof. We distinguish four cases of $\widehat{x}$ in

$$X \xrightarrow{\quad x \quad} \mathsf{Val} \times \mathsf{Val} \xrightarrow{\mathcal{P}\ (\mathsf{applyClosure}\ f)} \mathsf{Val}$$

$$X \xrightarrow{\quad \widehat{x} \quad} \mathsf{Val} \times \mathsf{Val} \xrightarrow{\mathsf{applyClosure}\ \widehat{f}} \mathsf{Val},$$

- The cases $\widehat{x} = \top \times \widehat{a}$, $\widehat{x} = \bot \times \widehat{a}$, and $\widehat{x} = \widehat{\mathsf{NumVal}} \times \widehat{a}$ are analogous to cases in the proof for succ.
- In case $\widehat{x} = (\widehat{\mathsf{ClosureVal}} \circ \widehat{c}) \times \widehat{a}$, then $x \subseteq \mathcal{P}((\mathsf{ClosureVal} \circ c) \times \widehat{a}) = \gamma((\widehat{\mathsf{ClosureVal}} \circ \widehat{c}) \times \widehat{a})$ and $\{c\} \subseteq \gamma(\widehat{c})$. It follows,

$$\begin{aligned}
\alpha(\mathcal{P}(\mathsf{applyClosure}\ f \circ ((\mathsf{ClosureVal} \circ c) \times a))) &\sqsubseteq \alpha(\mathcal{P}(f \circ (c \times a))) \\
&\sqsubseteq \alpha(\bigsqcup \mathcal{P}(f) \circ \mathcal{P}(\{c \times a\})) \\
&\sqsubseteq \alpha(\bigsqcup \mathcal{P}(f)) \circ \alpha(\mathcal{P}(\{c \times a\})) \\
&\sqsubseteq \bigsqcup \alpha(\mathcal{P}(f)) \circ \alpha(\mathcal{P}(\mathsf{strength} \circ (\{c\} \times a))) \\
&\sqsubseteq \bigsqcup \widehat{f} \circ \mathsf{strength} \circ (\widehat{c} \times \widehat{a}) \\
&\sqsubseteq \widehat{\mathsf{applyClosure}}\ \widehat{f} \circ ((\widehat{\mathsf{ClosureVal}} \circ \widehat{c}) \times \widehat{a}).
\end{aligned}$$

□

2.2 Soundness of Environment Operations

```
class Arrow c => ArrowEnv x y env c | c -> x, c -> y, c -> env where
  lookup :: c x (Maybe y)
  getEnv :: c () env
  extendEnv :: c (x,y,env) env
  localEnv :: c a b -> c (env,a) b

newtype Environment var val c x y = Environment (c (Env var val,x) y)
newtype BoundedEnv var addr val c x y =
  BoundedEnv ( c ((Env var addr,Store addr val),x) (Store addr val, y) )
```

Definition 10 (Galois Connection for Environments)**.** Let $\mathsf{Env} = \mathsf{Var} \mapsto \mathsf{Val}$ be the concrete environment and $\widehat{\mathsf{Env}} = \mathsf{Var} \mapsto \mathsf{Addr}$, $\widehat{\mathsf{Store}} = \mathsf{Addr} \mapsto \widehat{\mathsf{Val}}$. Abstract stores are ordered as follows:

$$\sigma \sqsubseteq \sigma' \text{ iff } \mathrm{dom}(\sigma) \subseteq \mathrm{dom}(\sigma') \wedge \forall x \in \sigma.\, \sigma(x) \sqsubseteq \sigma'(x)$$

This way the least upper bound of two stores is the union of both stores where overlapping variables are joined.

We say that an environment $\rho \in \widehat{\mathsf{Env}}$ is consistent with respect to a store $\sigma \in \widehat{\mathsf{Store}}$, written $\rho \lesssim \sigma$, if and only if, all variables that have a bound address in the environment, there exists an a binding of this address in the store, i.e., $\forall v \in \rho.\, \rho(v) \in \sigma$.

Given a Galois connection $\alpha_\# : \mathcal{P}((\mathsf{Var} \mapsto \mathsf{Var}) \times (\mathsf{Var} \mapsto \mathsf{Val})) \leftrightarrows \widehat{\mathsf{Env}} \times \widehat{\mathsf{Store}} : \gamma_\#$ with $\rho \lesssim \sigma$ for all $(\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma})$ iff $v \in \widehat{\rho} \wedge \widehat{\rho}(v) \in \widehat{\sigma}$. We define a Galois connection $\alpha_E : \mathcal{P}\mathsf{Env} \leftrightarrows \widehat{\mathsf{Env}} \times \widehat{\mathsf{Store}} : \gamma_E$ as follows.

$$\alpha_E(X) = \alpha_\#(\{([v \mapsto v \mid v \in \rho], [v \mapsto \rho(v) \mid v \in \rho]) \mid \rho \in X\}$$
$$\gamma_E(\widehat{\rho}, \widehat{\sigma}) = \{[v \mapsto \sigma(\rho(v)) \mid v \in \rho] \mid (\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma})\}$$

Then clearly, for all $\alpha_E(X) = (\widehat{\rho}, \widehat{\sigma})$, $\widehat{\rho} \lesssim \widehat{\sigma}$

Definition 11 (Soundness of the Environment Arrow Transformers)**.** $f \in$ `Environment Var Val C X Y` is sound w.r.t $\widehat{\mathsf{f}} \in$ `BoundedEnv Var Addr Val C X Y`, written $f \mathrel{\dot\sqsubseteq} \widehat{f}$

$$\begin{aligned}
&\text{iff } \forall \alpha(\mathcal{P}(\rho \times \sigma \times x)) \sqsubseteq \widehat{\rho} \times \widehat{\sigma} \times \widehat{x} \wedge \mathsf{envs}(\widehat{x}) \lesssim \widehat{\sigma} \\
&\quad \Longrightarrow\ \alpha(\mathcal{P}(f \circ (\rho \times \sigma \times x)) \sqsubseteq \widehat{f} \circ (\widehat{\rho} \times \widehat{\sigma} \times \widehat{x}) \\
&\qquad\quad \wedge \text{ let } \widehat{\sigma}' \times \widehat{y} = \widehat{f} \circ (\widehat{\rho} \times \widehat{\sigma} \times \widehat{x}) \text{ in } \widehat{\sigma} \sqsubseteq \widehat{\sigma}' \wedge \mathsf{envs}(\widehat{y}) \lesssim \widehat{\sigma}'
\end{aligned}$$

where $\mathsf{envs}(\widehat{x}) \lesssim \widehat{\sigma}$ compares all environments in $\widehat{x}$ for consistency w.r.t. the store $\widehat{\sigma}$.

Lemma 12 (`lookup` is sound)**.** To show: $\mathtt{lookup} \mathrel{\dot\sqsubseteq} \widehat{\mathtt{lookup}}$, where `lookup` and $\widehat{\mathtt{lookup}}$ are defined as

```
lookup = arr (\(env,x) -> lookup x env)
lookup^ = arr (\((env,store),x) -> (store,do addr <- lookup x env; lookup addr store))
```

Proof. Let $X \subseteq \mathsf{Env} \times \mathsf{Var}$ and $\alpha(X) = ((\widehat{\rho}, \widehat{\sigma}), v)$. We distinguish two cases if v is has a binding both in $\widehat{\rho}$ and $\widehat{\sigma}$.

- In case $v \in \widehat{\rho} \wedge \widehat{\rho}(v) \in \widehat{\sigma}$, then also $v \in \rho$ for all $\rho \in \gamma(\widehat{\rho}, \widehat{\sigma})$. Furthermore, $\{\sigma(\rho(v)) \mid (\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma})\} \subseteq \gamma(\widehat{\sigma}(\widehat{\rho}(v)))$, and $X \subseteq \{([x \mapsto \sigma(\rho(x)) \mid x \in \rho \wedge \rho(x) \in \sigma], v) \mid (\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma})\} = \gamma_E(\widehat{\rho}, \widehat{\sigma}) \times \gamma(v)$. It follows

  $$\begin{aligned}
  &\alpha(\{\rho(v) \mid (\rho, v) \in X\}) \sqsubseteq \alpha(\{[x \mapsto \sigma(\rho(x)) \mid x \in \rho \wedge \rho(x) \in \sigma](v) \mid (\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma})\}) \\
  &\qquad \sqsubseteq \alpha(\{\sigma(\rho(v)) \mid (\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma})\}) \sqsubseteq \widehat{\sigma}(\widehat{\rho})(v).
  \end{aligned}$$

- In case $v \notin \widehat{\rho} \vee \widehat{\rho}(v) \in \widehat{\sigma}$, then $v \notin \rho$ for all $(\rho, \sigma) \in \gamma(\widehat{\rho}, \widehat{\sigma})$.

□

Lemma 13 (getEnv is sound)**.** To show: $\mathtt{getEnv} \mathrel{\dot{\sqsubseteq}} \widehat{\mathtt{getEnv}}$, where getEnv and $\widehat{\mathtt{getEnv}}$ are defined as

```
getEnv = arr (\(env,()) -> env)
getEnv^ = arr (\((env,store),()) -> (store,env))
```

Proof. Let $\alpha(\mathcal{P}(\rho)) = \widehat{\rho} \times \widehat{\sigma}$, then $\alpha(\mathcal{P}(\mathtt{getEnv} \circ (\rho \times ())) \sqsubseteq \alpha(\mathcal{P}(\rho)) \sqsubseteq \widehat{\sigma} \times \widehat{\rho} \sqsubseteq \widehat{\mathtt{getEnv}}$. □

Lemma 14 (extendEnv is sound)**.** To show: $\mathtt{extendEnv} \mathrel{\dot{\sqsubseteq}} \widehat{\mathtt{extendEnv}}$, where extendEnv and $\widehat{\mathtt{extendEnv}}$ are defined as

```
extendEnv = arr (\(x,y,env) -> E.insert x y env)
extendEnv^ = (id &&& alloc) >>>
   arr (\(_,store,((x,y,env),addr)) -> (insertWith (⊔) addr y store,insert x addr env))
```

Proof. Let $X \subseteq \mathtt{Env} \times \mathtt{Var} \times \mathtt{Val}$ and $\alpha(X) = (\widehat{\rho}, \widehat{\sigma}, x, \widehat{v})$ and $a \in \mathtt{Addr}$, then $X \subseteq \{([x' \mapsto \sigma(\rho(x)) \mid x' \in \rho \wedge \rho(x') \in \sigma], x, v) \mid (\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma}), v \in \gamma(\widehat{v})\} = \gamma(\widehat{\rho}, \widehat{\sigma}, x, \widehat{v})$.

$$
\begin{aligned}
&\alpha(\{\rho[x \mapsto v] \mid (\rho, x, v) \in X\}) \\
&\quad \sqsubseteq \alpha(\{[x' \mapsto \sigma(\rho(x)) \mid x' \in \rho \wedge \rho(x') \in \sigma][x \mapsto v] \mid (\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma}), v \in \gamma(\widehat{v})\}) \\
&\quad \sqsubseteq \alpha_\#\{([x' \mapsto x' \mid x' \in \rho[x \mapsto v]], [x' \mapsto \sigma(\rho(x')) \mid x' \in \rho, \rho(x') \in \sigma][x \mapsto v]) \mid (\rho, \sigma) \in \gamma_\#(\widehat{\rho}, \widehat{\sigma}), v \in \gamma(\widehat{v})\}) \\
&\quad \sqsubseteq (\widehat{\sigma}[a \mapsto \widehat{v}], \widehat{\rho}[x \mapsto a]).
\end{aligned}
$$

Furthermore, $\widehat{\rho}[x \mapsto a] \lesssim \widehat{\sigma}[a \mapsto \widehat{v}]$ follows from the assumption $\widehat{\rho} \lesssim \widehat{\sigma}$. □

Lemma 15 (localEnv is sound)**.** To show: Given $f \mathrel{\dot{\sqsubseteq}} \widehat{f}$, then $\mathtt{localEnv}\ \mathtt{f} \mathrel{\dot{\sqsubseteq}} \widehat{\mathtt{localEnv}}\ \widehat{f}$, where localEnv and $\widehat{\mathtt{localEnv}}$ are defined as

```
localEnv f = arr (\(_,(env,x)) -> (env,x)) >>> f
localEnv^ f = arr (\((_,store),(env,x)) -> ((env,store),x)) >>> f.
```

Proof. Let $\alpha(\mathcal{P}(\rho \times \rho' \times x)) = \widehat{\rho} \times \widehat{\rho}' \times \widehat{x}$. The assumption $\widehat{\rho}' \lesssim \widehat{\sigma}$ allows us to invoke the premise $f \mathrel{\dot{\sqsubseteq}} \widehat{f}$.

$$
\begin{aligned}
&\alpha(\mathcal{P}(\mathtt{localEnv}\ f \circ (\rho \times (\rho' \times x)))) \sqsubseteq \alpha(\mathcal{P}(f \circ (\rho' \times x))) \\
&\quad \sqsubseteq \widehat{f} \circ \alpha(\mathcal{P}(\rho' \times x)) \sqsubseteq \widehat{f} \circ ((\widehat{\rho}' \times \widehat{\sigma}) \times \widehat{x}) \sqsubseteq \widehat{\mathtt{localEnv}}\ \widehat{f} \circ ((\widehat{\rho} \times \widehat{\sigma}) \times \widehat{\rho}' \times \widehat{x})
\end{aligned}
$$

Furthermore, let $\widehat{\sigma}' \times \widehat{y} = \widehat{f} \circ ((\widehat{\rho}' \times \widehat{\sigma}) \times \widehat{x})$, then $\widehat{\sigma} \sqsubseteq \widehat{\sigma}'$ follows by the assumption $f \mathrel{\dot{\sqsubseteq}} \widehat{f}$. □

2.3 Soundness of Language-Independent Arrow Operations

```
class Arrow c => ArrowFix x y c | c -> y, c -> x where
  fixA :: (c x y -> c x y) -> c x y

newtype Fix a b x y = Fix { runFix :: x -> y }
newtype CacheArrow a b x y = CacheArrow (((Store a b,Store a b),x) -> (Store a b,y))
```

The concrete arrow `Fix` and abstract arrow `CacheArrow` appear on the inner-most level of the arrow transformer stack. And hence all inputs and outputs are explicit to these arrows.

Lemma 16 (`fixA` is sound)**.** To show: $\left[\forall x, \widehat{x}.\, x \mathbin{\dot{\sqsubseteq}} \widehat{x} \Rightarrow f(x) \mathbin{\dot{\sqsubseteq}} f(\widehat{x})\right] \implies \texttt{fixA}\ f \mathbin{\dot{\sqsubseteq}} \widehat{\texttt{fixA}}\ f$, where `fixA` and $\widehat{\texttt{fixA}}$ are defined as follows:

```
fixA f = f (fixA f)

fixA^ f = proc x -> do
  old <- getOutCache -< ()
  setOutCache -< bottom
  y <- localInCache (fix (memoize . f)) -< (old,x)
  new <- getOutCache -< ()
  if (new ⊑ old) -- We are in the reductive set of `f` and have overshot the fixpoint
    then returnA -< y
    else fixA f -< x
  where
    memoize f = proc x -> do
      m <- lookupOutCache -< x
      case m of
        Just y -> do
          returnA -< y
        Nothing -> do
          yOld <- lookupInCache -< x
          writeOutCache -< (x, fromMaybe bottom yOld)
          y <- f -< x
          updateOutCache -< (x, y)
          returnA - y
```

Proof. The proof is the same as in **?**]. □

3 INTERFACE AND SHARED INTERPRETER FOR STRATEGO

```
class Arrow c => ArrowTry c where
  tryA :: c x y -> c y z -> c x z -> c x z
```

```
type Ar c = (ArrowChoice c, ArrowTry c)
class IsTerm t where
  matchTermAgainstConstructor :: Ar c => c ([t'],[t]) [t] -> c (Text,[t'],t) t
  matchTermAgainstString :: Ar c => c (Text,t) t
  matchTermAgainstNumber :: Ar c => c (Int,t) t
  matchTermAgainstExplode :: Ar c => c t t -> c t t -> c t t

  equal :: Ar c => c (t,t) t                  cons :: Ar c => c (Text,[t]) t
  convertFromList :: Ar c => c (t,t) t        numberLiteral :: Ar c => c Int t
  mapSubterms :: Ar c => c [t] [t] -> c t t   stringLiteral :: Ar c => c Text t

class HasTermEnv env c | c -> env where
  getTermEnv :: c () env
  putTermEnv :: c env ()

class IsTermEnv env t | env -> t where
  lookupTerm :: (Ar c,HasTermEnv env c) => c t a -> c () a -> c (TermVar,env) a
  insertTerm :: (Ar c, HasTermEnv env c) => c (TermVar,t,env) env
  deleteTerms :: (Ar c, HasTermEnv env c) => c ([TermVar],env) env
  unionTermEnvs :: (Ar c, HasTermEnv env c) => c ([TermVar],env,env) env

data Strat = Fail | Id | Seq Strat Strat | GuardedChoice Strat Strat Strat
  | One Strat | Some Strat | All Strat | Match TermPattern | Build TermPattern
  | Scope [TermVar] Strat | Let [(StratVar,Strategy)] Strat
  | Call StratVar [Strat] [TermVar]

eval' :: (ArrowChoice c, ArrowTry c, ArrowPlus c, ArrowApply c, ArrowFix c t,
          HasStratEnv c, IsTerm t, HasTermEnv env c, IsTermEnv env t)
      => (Strat -> c t t)
eval' = fixA $ \ev s0 -> case s0 of
  Id -> id;  Fail -> failA
  Seq s1 s2 -> ev s1 >>> ev s2
  GuardedChoice s1 s2 s3 -> tryA (ev s1) (ev s2) (ev s3)
  One s -> mapSubterms (one (ev s))
  Some s -> mapSubterms (some (ev s))
  All s -> mapSubterms (all (ev s))
  Scope xs s -> scope xs (ev s)
  Match f -> proc t -> match -< (f,t)
  Build f -> proc _ -> build -< f
  Let bnds body -> let_ bnds body ev
```

```
  Call f ss ps -> call f ss ps ev

one :: (ArrowChoice c, ArrowPlus c) => c t t -> c [t] [t]
one f = proc l -> case l of
  (t:ts) -> do
    (t',ts') <- first f <+> second (one f) -< (t,ts)
    returnA -< (t':ts')
  [] -> failA -< ()

some :: (ArrowChoice c, ArrowTry c, ArrowPlus c) => c t t -> c [t] [t]
some f = go
  where
    go = proc l -> case l of
      (t:ts) -> do
        (t',ts') <- tryA (first f) (second go') (second go) -< (t,ts)
        returnA -< t':ts'
      [] -> failA -< ()
    go' = proc l -> case l of
      (t:ts) -> do
        (t',ts') <- tryA (first f) (second go') (second go') -< (t,ts)
        returnA -< t':ts'
      [] -> returnA -< []

all :: ArrowChoice c => c x y -> c [x] [y]
all f = proc l -> case l of
  [] -> returnA -< []
  (x:xs) -> do
    y <- f -< x
    ys <- all f -< xs
    returnA -< (y:ys)

scope :: (ArrowChoice c, ArrowTry c, HasTermEnv env c, IsTermEnv env t)
      => [TermVar] -> c x y -> c x y
scope vars s = proc t -> do
  env  <- getTermEnv  -< ()
  _    <- deleteTerms -< vars
  t'   <- s           -< t
  env' <- getTermEnv  -< ()
  putTermEnv <<< unionTermEnvs -< (vars,env,env')
  returnA -< t'
```

```
let_ :: (ArrowApply c, HasStratEnv c)
     => [(StratVar,Strategy)] -> Strat -> (Strat -> c t t) -> c t t
let_ ss body ev = proc a -> do
  let ss' = [ (v,Closure s' M.empty) | (v,s') <- ss ]
  senv <- readStratEnv -< ()
  localStratEnv (M.union (M.fromList ss') senv) (ev body) -<< a

call :: (ArrowChoice c, ArrowTry c, ArrowApply c,
         HasTermEnv env c, IsTermEnv env t, HasStratEnv c, HasStack t c)
     => StratVar -> [Strat] -> [TermVar] -> (Strat -> c t t) -> c t t
call f actualStratArgs actualTermArgs ev = proc a -> do
  senv <- readStratEnv -< ()
  case Map.lookup f senv of
    Just (Closure formalStratArgs formalTermArgs body senv') -> do
      tenv <- getTermEnv -< ()
      mapA bindTermArg -< zip actualTermArgs formalTermArgs
      let senv'' = foldl bindStratArgs (if Map.null senv' then senv else senv')
                                       (zip formalStratArgs actualStratArgs)
      b <- localStratEnv senv'' (ev body) -<< a
      tenv' <- getTermEnv -< ()
      putTermEnv <<< unionTermEnvs -< (formalTermArgs,tenv,tenv')
      returnA -< b
    Nothing -> error (printf "strategy␣%s␣not␣in␣scope" (show f)) -< ()
  where
    bindTermArg = proc (actual,formal) ->
      lookupTerm (proc t -> insertTerm -< (formal,t)) failA -<< actual

    bindStratArgs senv (v,Call v' [] []) senv =
      case M.lookup v' senv of
        Just s -> M.insert v s senv
        _ -> error $ "unknown␣strategy:␣" ++ show v'
    bindStratArgs senv (v,s) = M.insert v (Closure [] [] s senv) senv

data TermPattern
  = As TermVar TermPattern
  | Cons Constructor [TermPattern]
  | Explode TermPattern TermPattern
  | Var TermVar
  | StringLiteral Text
  | NumberLiteral Int
```

```
match :: (ArrowChoice c, ArrowTry c, ArrowApply c,
          IsTerm t, HasTermEnv env c, IsTermEnv env t)
      => c (TermPattern,t) t
match = proc (p,t) -> case p of
  Pat.As v p2 -> do
    t' <- match -< (Pat.Var v,t)
    match -< (p2,t')
  Pat.Var "_" ->
    success -< t
  Pat.Var x ->
    lookupTerm
      (proc t' -> do t'' <- equal -< (t,t'); insertTerm -< (x,t''); returnA -< t'')
      (proc () -> do insertTerm -< (x,t); returnA -< t) -<< x
  Pat.Cons c ts ->
    matchTermAgainstConstructor (zipWithA match) -< (c,ts,t)
  Pat.Explode c ts ->
    matchTermAgainstExplode
      (proc c' ->  match -< (c,c'))
      (proc ts' -> match -< (ts,ts')) -<< t
  Pat.StringLiteral s ->
    matchTermAgainstString -< (s,t)
  Pat.NumberLiteral n ->
    matchTermAgainstNumber -< (n,t)

build :: (ArrowChoice c, ArrowTry c, IsTerm t, HasTermEnv env c, IsTermEnv env t)
      => c TermPattern t
build = proc p -> case p of
  Pat.Var x ->
    lookupTerm returnA failA -< x
  Pat.Cons c ts -> do
    ts' <- mapA build -< ts
    cons -< (c,ts')
  Pat.Explode c ts -> do
    c'  <- build -< c
    ts' <- build -< ts
    convertFromList -< (c',ts')
  Pat.NumberLiteral n -> numberLiteral -< n
  Pat.StringLiteral s -> stringLiteral -< s
```

4 SOUNDNESS OF THE ABSTRACT INTERPRETER FOR STRATEGO

Definition 17. The concrete and abstract interpreter arrow for Stratego are defined as follows:

$$I(A,B) = \texttt{StratEnv} \times \texttt{TermEnv} \times A \to \texttt{Result}(B, \texttt{TermEnv})$$
$$\widehat{I}(A,B) = \texttt{StratEnv} \times \widehat{\texttt{TermEnv}} \times A \to \mathcal{P}(\texttt{Result}(B, \widehat{\texttt{TermEnv}})).$$

We continue by defining the ordering for the abstract arrow, which is the standard point-wise ordering on functions, however, with a non-standard ordering on powersets:

Definition 18. Given an order $\sqsubseteq_{\widehat{A}}$ for a set $\widehat{A}$, we define a complete preorder for all $X, Y \in \mathcal{P}\widehat{A}$:

$$X \sqsubseteq_{\mathcal{P}\widehat{A}} Y \quad \text{iff} \quad \forall x \in X.\, \exists y \in Y. x \sqsubseteq_{\widehat{A}} y$$

Definition 19. We define a Galois connection $\alpha : \mathcal{P}A \rightleftarrows \mathcal{P}\widehat{A} : \gamma$, given a function $\iota : A \to \widehat{A}$. The abstraction function is defined by $\alpha(X) = \{\iota(x) \mid x \in X\}$. Clearly α preserves least upper bounds, i.e. $\alpha(X \cup Y) = \alpha(X) \sqcup \alpha(Y) = \alpha(X) \cup \alpha(Y)$ and $\mathcal{P}\widehat{A}$ is a complete order. By [**?** , Lemma 4.23], we obtain that the concretization function γ exists and α and γ form a Galois connection.

We can simplify a soundness proofs $f \mathrel{\dot\sqsubseteq} \widehat{f}$ for our special ordering on $\mathcal{P}\widehat{B}$ as follows

$$\begin{aligned}
&\alpha(\{f(x) \mid x \in X\}) \sqsubseteq \widehat{f}(\alpha(X)) \\
&\quad \Leftrightarrow \forall \widehat{y} \in \alpha(\{f(x) \mid x \in X\}).\, \exists \widehat{y}' \in \widehat{f}(\alpha(X)).\ \widehat{y} \sqsubseteq \widehat{y}' \\
&\quad \Leftrightarrow \forall x \in X.\, \exists \widehat{y}' \in \widehat{f}(\alpha(X)).\ \alpha(f(x)) \sqsubseteq \widehat{y}'.
\end{aligned}$$

Lemma 20 (The fixpoint combinators are sound)**.** In particular, `fix` calculates the least fixpoint and $\widehat{\texttt{fix}}_n$ approximates the greatest fixpoint.

$$\texttt{fix}(f) = \bigsqcup_i f^i(\bot) \qquad\qquad \widehat{\texttt{fix}}_i(\widehat{f}) = \bigsqcup_{i \le n} \widehat{f}^i(\top)$$

Proof. To show $\texttt{fix}(f) \mathrel{\dot\sqsubseteq} \widehat{\texttt{fix}}_n(f)$ where for f holds $\forall x \mathrel{\dot\sqsubseteq} \widehat{x} \Rightarrow f(x) \mathrel{\dot\sqsubseteq} f(\widehat{x})$. We prove by induction over natural numbers on n.

- In the base case $n = 1$, $\texttt{fix}(f) \mathrel{\dot\sqsubseteq} \top$.
- In the step case the induction hypothesis is $\texttt{fix}(f) \mathrel{\dot\sqsubseteq} \widehat{\texttt{fix}}_n(f)$ and we have to show $\texttt{fix}(f) \mathrel{\dot\sqsubseteq} \widehat{\texttt{fix}}_{n+1}(f)$.

$$\begin{aligned}
\alpha(\texttt{fix}(f)) &\sqsubseteq \alpha(\bigsqcup_i f^i(\bot)) \sqsubseteq \alpha(f^{n+1}(\bot)) \sqcup \alpha(\bigsqcup_i f^i(\bot)) \\
&\sqsubseteq f^{n+1}(\top) \sqcup \bigsqcup_{i \le n} f^i(\top) \sqsubseteq \bigsqcup_{i \le n+1} f^i(\top) = \widehat{\texttt{fix}}_{n+1}(f)
\end{aligned}$$

□

Lemma 21 (`try` is sound)**.** That is $f \dot{\sqsubseteq} \widehat{f} \wedge g \dot{\sqsubseteq} \widehat{g} \wedge h \dot{\sqsubseteq} \widehat{h} \implies \texttt{try}\ f\ g\ h \dot{\sqsubseteq} \widehat{\texttt{try}}\ \widehat{f}\ \widehat{g}\ \widehat{h}$, where the semantics of `try` and $\widehat{\texttt{try}}$ are

$$\texttt{try}\ f\ g\ h\ x = \begin{cases} g(y), & \texttt{Success}\ y = f(x) \\ h(x), & \texttt{Fail} = f(x), \end{cases} \qquad \widehat{\texttt{try}}\ \widehat{f}\ \widehat{g}\ \widehat{h}\ \widehat{x} = \begin{cases} \widehat{g}(\widehat{y}), & \texttt{Success}\ \widehat{y} \in \widehat{f}(\widehat{x}) \\ \widehat{h}(\widehat{x}), & \texttt{Fail} \in \widehat{f}(\widehat{x}), \end{cases}$$

Proof. To show: $\forall x \in X.\ \exists \widehat{y} \in \widehat{\texttt{try}}\ \widehat{f}\ \widehat{g}\ \widehat{h}\ \alpha(X).\ \alpha(\texttt{try}\ f\ g\ h\ x) \sqsubseteq \widehat{y}$
We proof by case distinction over $f(x)$:

- In case $f(x) = \texttt{Success}\ y$, by the assumption $f \dot{\sqsubseteq} \widehat{f}$ it follows that there exists an $\texttt{Success}\ \widehat{y} \in \widehat{f}(\alpha(X))$ with $\alpha(y) \sqsubseteq \widehat{y}$. Furthermore, from the assumption $g \dot{\sqsubseteq} \widehat{g}$ follows that there exists an $\widehat{z} \in \widehat{g}(\alpha(y))$ with $\alpha(g(y)) \sqsubseteq \widehat{z}$. We conclude
$$\alpha(\texttt{try}\ f\ g\ h\ x) = \alpha(g(y)) \sqsubseteq \widehat{z} \in \widehat{g}(\alpha(y)) \sqsubseteq \widehat{g}(\widehat{y}) \subseteq \widehat{\texttt{try}}\ \widehat{f}\ \widehat{g}\ \widehat{h}\ \alpha(X)$$
- In case $f(x) = \texttt{Fail}$, by the assumption $f \dot{\sqsubseteq} \widehat{f}$ we know that $\texttt{Fail} \in \widehat{f}(\alpha(X))$ and from the assumption $h \dot{\sqsubseteq} \widehat{h}$, we know that there exists an $\widehat{z} \in \widehat{h}(\alpha(X))$ with $\alpha(h(x))) \sqsubseteq \widehat{z}$.
$$\alpha(\texttt{try}\ f\ g\ h\ x) = \alpha(h(x)) \sqsubseteq \widehat{z} \in \widehat{h}(\alpha(X)) \subseteq \widehat{\texttt{try}}\ \widehat{f}\ \widehat{g}\ \widehat{h}\ \alpha(X)$$

□

Lemma 22 (`fail` is sound)**.** Proof. The `fail` operations are defined by $\texttt{fail}_M\ x = \texttt{Fail}$ and $\texttt{fail}_{\widehat{M}}\ x = \{\texttt{Fail}\}$. We show that $\forall x \in X.\ \exists \widehat{y} \in \texttt{fail}_{\widehat{M}}\ \alpha(X).\ \alpha(\texttt{fail}_M\ x) \sqsubseteq \widehat{y}$. Clearly $\widehat{y} = \texttt{Fail}$ satisfies this proposition.
□

Lemma 23 (`orElse` is sound)**.** In particular, we prove soundness of the `ArrowPlus` operation `<+>` with the following semantics:

$$(f\ \texttt{<+>}\ g)(x) = \begin{cases} f(x), & f(x) = \texttt{Success}\ y \\ g(x), & f(x) = \texttt{Fail} \end{cases} \qquad (\widehat{f}\ \widehat{\texttt{<+>}}\ \widehat{g})(\widehat{x}) = \widehat{f}(\widehat{x}) \sqcup \widehat{g}(\widehat{x})$$

Proof. For all $x \in X$,

- In case $f(x) = \texttt{Success}\ y$, then $\alpha(f\ \texttt{<+>}\ g)(x) = \alpha(f(x)) \sqsubseteq \widehat{f}(\alpha(X)) \sqsubseteq (\widehat{f}\ \widehat{\texttt{<+>}}\ \widehat{g})(\alpha(X))$.
- In case $f(x) = \texttt{Fail}$, then $\alpha(f\ \texttt{<+>}\ g)(x) = \alpha(g(x)) \sqsubseteq \widehat{g}(\alpha(X)) \sqsubseteq (\widehat{f}\ \widehat{\texttt{<+>}}\ \widehat{g})(\alpha(X))$.

□

Lemma 24 (Soundness of the language independent arrow operations)**.** Both of our arrows are isomorphic to Kleisli arrows, i.e., arrows of the form $A \to M(B)$, where M is a monad. For our arrows these monads are

$$M(A) = \texttt{StratEnv} \times \texttt{TermEnv} \to \texttt{Result}(A, \texttt{TermEnv})$$
$$\widehat{M}(A) = \texttt{StratEnv} \times \widehat{\texttt{TermEnv}} \to \mathcal{P}(\texttt{Result}(A, \widehat{\texttt{TermEnv}})).$$

The arrow operations id, >>>, ***, ||| and app for Kleisli arrows have the same implementation, except for different instantiations of the monadic operations. If prove soundness of these monad operations, then soundness for the Kleisli arrow operations follows by parametricity. In particular, we prove (i) $\mathtt{fmap}_M \mathrel{\dot{\sqsubseteq}} \mathtt{fmap}_{\widehat{M}}$, (ii) $\mathtt{return}_M \mathrel{\dot{\sqsubseteq}} \mathtt{return}_{\widehat{M}}$, and (iii) $\mathtt{join}_M \mathrel{\dot{\sqsubseteq}} \mathtt{join}_{\widehat{M}}$.

Proof.

(i) The semantics of $\mathtt{fmap}_M$ and $\mathtt{fmap}_{\widehat{M}}$ is

$$\mathtt{fmap}_M\ f\ x = \begin{cases} \mathtt{Success}\ f(x'), & \mathtt{Success}\ x' = x \\ \mathtt{Fail}, & \mathtt{Fail} = x \end{cases}$$

$$\mathtt{fmap}_{\widehat{M}}\ \widehat{f}\ X = \begin{cases} \mathtt{Success}\ \widehat{f}(x'), & \mathtt{Success}\ x' \in X \\ \mathtt{Fail}, & \mathtt{Fail} \in X. \end{cases}$$

We show that $\forall x \in X. \exists u \in \{\mathtt{fmap}_{\widehat{M}}\ \widehat{f}\ \alpha(x) \mid x \in X\}. \alpha(\mathtt{fmap}_M\ f\ u) \sqsubseteq u$. We continue by case distinction on x.

- Case $x = \mathtt{Success}\ x'$, it follows

$$\begin{aligned} \alpha(\mathtt{fmap}_M\ f\ x) &= \alpha(\mathtt{Success}\ f(x')) \sqsubseteq \mathtt{Success}\ \widehat{f}(\alpha(x')) \\ &= \mathtt{fmap}_{\widehat{M}}\ \widehat{f}\ (\mathtt{Success}\ \alpha(x')) = \mathtt{fmap}_{\widehat{M}}\ \widehat{f}\ \alpha(x) \end{aligned}$$

- Case $x = \mathtt{Fail}$, it follows $\alpha(\mathtt{fmap}_M\ f\ x) = \alpha(\mathtt{Fail}) = \{\mathtt{Fail}\} = \mathtt{fmap}_{\widehat{M}}\ \widehat{f}\ \{\mathtt{Fail}\} = \mathtt{fmap}_{\widehat{M}}\ \widehat{f}\ \alpha(x)$

(ii) The return operations are defined by $\mathtt{return}_M\ x = \mathtt{Success}\ y$ and $\mathtt{return}_{\widehat{M}}\ x = \{\mathtt{Success}\ x\}$. We show that $\forall x \in X. \exists \widehat{y} \in \mathtt{return}_{\widehat{M}}\ \alpha(X). \alpha(\mathtt{return}_M x) \sqsubseteq \widehat{y}$. Clearly $\widehat{y} = \mathtt{Success}\ \alpha(X)$ satisfies this proposition.

(iii) The semantics of $\mathtt{join}_M$ and $\mathtt{join}_{\widehat{M}}$ is

$$\mathtt{join}_M\ x = \begin{cases} \mathtt{Success}\ y, & \mathtt{Success}(\mathtt{Success}\ y) = x \\ \mathtt{Fail}, & \text{otherwise} \end{cases}$$

$$\mathtt{join}_{\widehat{M}}\ X = \begin{cases} \mathtt{Fail}, & \mathtt{Fail} \in X \\ \mathtt{Fail}, & \mathtt{Success}\ Y \in X, \mathtt{Fail} \in Y \\ \mathtt{Success}\ z, & \mathtt{Success}\ Y \in X, \mathtt{Success}\ z \in Y. \end{cases}$$

We show that $\forall x \in X. \exists u \in \mathtt{join}_{\widehat{M}}\ \alpha(X). \alpha(\mathtt{join}_M x) \sqsubseteq u$. We distinguish three cases for the variable x:

- In case $x = \mathtt{Fail}$, it follows $\alpha(\mathtt{join}_M\ \mathtt{Fail}) = \alpha(\mathtt{Fail}) = \mathtt{Fail} \in \mathtt{join}_{\widehat{M}}\ \alpha(X)$.
- In case $x = \mathtt{Success}\ \mathtt{Fail}$, it follows $\alpha(\mathtt{join}_M\ (\mathtt{Success}\ \mathtt{Fail})) = \alpha(\mathtt{Fail}) = \mathtt{Fail} \in \mathtt{join}_{\widehat{M}}\ \alpha(X)$.

- In case $x = \mathsf{Success}\ (\mathsf{Success}\ z)$, then $\mathsf{Success}\ \{\mathsf{Success}\ z\} \in \alpha(X)$. It follows

$$\alpha(\mathsf{join}_M\ (\mathsf{Success}\ (\mathsf{Success}\ z))) = \alpha(\mathsf{Success}\ z) = \mathsf{Success}\ (\alpha(z)) \in \mathsf{join}_{\widehat{M}}\ \alpha(X).$$

□

Lemma 25 (Soundness of fetching and storing operations)**.** The operations that fetch values from the monad or store them back in the monad such as `readStratEnv`, `localStratEnv`, `getTermEnv`, `putTermEnv` are sound because of a similar argument. Both monads have the same structure and fetching operations access the same position in the monad, except that the abstract monad wraps the result in a powerset. □

Definition 26. Concrete terms are defined by the datatype

```
data Term = Cons String [Term] | StringLiteral String | NumberLiteral Double
```

and abstract terms by

```
data Term^ = Cons String [Term^] | StringLiteral String | NumberLiteral Double | Wildcard,
```

where `Wildcard` is the greatest element of $\widehat{\mathsf{Term}}$.

Lemma 27 (Matching a term against a pattern is sound)**.** In particular, we prove soundness of (i) `matchTermAgainstConstructor`, (ii) `matchTermAgainstString`, (iii) `matchTermAgainstNumber`, and (iv) `matchTermAgainstExplode`.

Proof.

(i) For readability of the proof, we rename `matchTermAgainstConstructor` to `matchCons`. The semantics of `matchCons` is given by the recursive equations

$$\mathsf{matchCons}\ f\ (c,n,t) = \begin{cases} \mathsf{Cons}\ c\ [t'_1 \ldots t'_n] & t = \mathsf{Cons}\ c\ [t_1 \ldots t_n] \wedge t'_1 = f(t_1) \ldots \wedge\ t'_n = f(t_n) \\ \mathsf{Fail} & \text{otherwise} \end{cases}$$

and

$$\widehat{\mathsf{matchCons}}\ \widehat{f}\ (c,n,t) = \begin{cases} \mathsf{Cons}\ c\ [\widehat{t'_1} \ldots \widehat{t'_n}] & t = \mathsf{Cons}\ c\ [\widehat{t_1} \ldots \widehat{t_n}] \wedge \widehat{t'_1} \in \widehat{f}(\widehat{t_1}) \ldots \wedge\ \widehat{t'_n} \in \widehat{f}(\widehat{t_n}) \\ \mathsf{Cons}\ c\ [\widehat{t'_1} \ldots \widehat{t'_n}] \sqcup \mathsf{Fail} & t = \mathsf{Wildcard} \wedge \widehat{t'_1} \in \widehat{f}(\mathsf{Wildcard}) \ldots \wedge\ \widehat{t'_n} \in \widehat{f}(\mathsf{Wildcard}) \\ \mathsf{Fail} & \text{otherwise} \end{cases}$$

We proceed by case distinction on $\alpha(X)$

- Case $\alpha(X) = (c, n, \mathsf{Cons}\ c\ [\widehat{t_1} \ldots \widehat{t_n}])$, then $X \subseteq \{(c, n, \mathsf{Cons}\ c\ [t_1 \ldots t_n]) \mid t_1 \in \gamma(\widehat{t_1}) \ldots t_n \in \gamma(\widehat{t_n})\}$. For all $(c, n, \mathsf{Cons}\ c\ [t_1 \ldots t_n]) \in X$, we know by $f \mathrel{\dot{\sqsubseteq}} \widehat{f}$ that for all t_i, there exists an $\widehat{t'_i} \in \widehat{f}(\alpha(t_i))$ with $\alpha(f(t_i)) \sqsubseteq \widehat{t'_i}$ or $\alpha(f(t_i)) \sqsubseteq \mathsf{Fail} \in \widehat{f}(\widehat{t_i})$. We conclude:

$$\alpha(\mathsf{matchCons}\ f\ (c, n, \mathsf{Cons}\ c\ [t_1 \ldots t_n])) = \alpha(\mathsf{Cons}\ c\ [t'_1 \ldots t'_n]) \sqsubseteq$$
$$\mathsf{Cons}\ c\ [\widehat{t'_1} \ldots \widehat{t'_n}] \in \widehat{\mathsf{matchCons}}\ \widehat{f}\ (c, n, \mathsf{Cons}\ c\ [\widehat{t_1} \ldots \widehat{t_n}])$$

or $\alpha(\mathsf{matchCons}\ f\ (c, n, \mathsf{Cons}\ c\ [t_1 \ldots t_n])) \sqsubseteq \mathsf{Fail} \in \widehat{\mathsf{matchCons}}\ \widehat{f}\ (c, n, \mathsf{Cons}\ c\ [\widehat{t_1} \ldots \widehat{t_n}])$.

- Case $\alpha(X) = (c, n, \texttt{Wildcard})$, then $X \subseteq \{(c, n, t) \mid t \in \texttt{Term}\}$ and $\texttt{matchCons}\ f\ (c, n, t) = \texttt{Cons}\ \ c\ [t'_1 \ldots t'_n]$ for $t'_i = f(t_i)$ and $t = \texttt{Cons}\ c\ [t_1 \ldots t_n]$ or $\texttt{matchCons}\ f\ (c, n, t) = \texttt{Fail}$. By $f \dot{\sqsubseteq} \widehat{f}$, we know that $\alpha(f(t_i)) \in \widehat{f}(\texttt{Wildcard})$ and hence we conclude that $\alpha(\texttt{matchCons}\ f\ (c, n, t)) \in \widehat{\texttt{matchCons}}\ \widehat{f}\ (c, n, \texttt{Wildcard})$.

(ii) The proof of `matchTermAgainstString` follows the same structure as `matchTermAgainstConstructor`.

(iii) The proof of `matchTermAgainstNumber` follows the same structure as `matchTermAgainstConstructor`.

(iv) To show: $\texttt{matchExplode}\ f\ g \dot{\sqsubseteq} \widehat{\texttt{matchExplode}}\ \widehat{f}\ \widehat{g}$ if $f \dot{\sqsubseteq} \widehat{f}$ and $g \dot{\sqsubseteq} \widehat{g}$. We proceed by case distinction on $\alpha(X)$,

- In case of $\alpha(X) = \texttt{Cons}\ c\ ts$, $\alpha(X) = \texttt{StringLiteral}c$, and $\alpha(X) = \texttt{NumberLiteral}n$, soundness of `matchExplode` follows from Lemma 24 and the assumption $f \dot{\sqsubseteq} \widehat{f}$ and $g \dot{\sqsubseteq} \widehat{g}$.
- In case of $\alpha(X) = \texttt{Wildcard}$, the code in $\widehat{\texttt{matchExplode}}$ is the least upper bound of cases for constructors and literals, except that $\widehat{f}$ and $\widehat{g}$ are called with `Wildcard`. This is a sound approximation, because $\widehat{f}$ and $\widehat{g}$ are monotone functions.

□

Lemma 28 (Term equality is sound)**.** The semantics of `equal` is given by the recursive equations

$$\texttt{equal}(t_1, t_2) = \begin{cases} \texttt{Cons}\ c\ [w_1 \ldots w_n] & t_1 = \texttt{Cons}\ c\ [u_1 \ldots u_n] \wedge t_2 = \texttt{Cons}\ c\ [v_1 \ldots v_n] \wedge \\ & \qquad w_1 = \texttt{equal}(u_1, v_1) \ldots \wedge w_n = \texttt{equal}(u_n, w_n) \\ \texttt{StringLiteral}\ s & t_1 = \texttt{StringLiteral}\ s \wedge t_2 = \texttt{StringLiteral}\ s \\ \texttt{NumberLiteral}\ n & t_1 = \texttt{NumberLiteral}\ n \wedge t_2 = \texttt{NumberLiteral}\ n \\ \texttt{Fail} & \text{otherwise} \end{cases}$$

and

$$\widehat{\texttt{equal}}(t_1, t_2) = \begin{cases} \texttt{Cons}\ c\ [w_1 \ldots w_n] & t_1 = \texttt{Cons}\ c\ [u_1 \ldots u_n] \wedge t_2 = \texttt{Cons}\ c\ [v_1 \ldots v_n] \wedge \\ & \qquad w_1 \in \widehat{\texttt{equal}}(u_1, v_1) \ldots \wedge w_n \in \widehat{\texttt{equal}}(u_n, w_n) \\ \texttt{StringLiteral}\ s & t_1 = \texttt{StringLiteral}\ s \wedge t_2 = \texttt{StringLiteral}\ s \\ \texttt{NumberLiteral}\ n & t_1 = \texttt{NumberLiteral}\ n \wedge t_2 = \texttt{NumberLiteral}\ n \\ t_2 \sqcup \texttt{Fail} & t_1 = \texttt{Wildcard} \\ t_1 \sqcup \texttt{Fail} & t_2 = \texttt{Wildcard} \\ \texttt{Fail} & \text{otherwise}\ . \end{cases}$$

Proof. To show: $\forall (t_1, t_2) \in X.\, \exists \widehat{t} \in \widehat{\texttt{equal}}(\alpha(X)).\, \alpha(\texttt{equal}(t_1, t_2)) \sqsubseteq \widehat{t}$. We proceed by well-founded induction on X over the height of terms and by case distinction on $\alpha(X)$.

- Case $\alpha(X) = (\mathsf{Cons}\ c[\widehat{u}_1 \ldots \widehat{u}_n], \mathsf{Cons}\ c[\widehat{v}_1 \ldots \widehat{v}_n])$, then

$$\widehat{\mathtt{equal}}(\alpha(X)) = \{\mathsf{Cons}\ c[\widehat{w}_1 \ldots \widehat{w}_n] \mid \widehat{w}_1 \in \widehat{\mathtt{equal}}(\widehat{u}_1, \widehat{v}_1) \ldots \widehat{w}_n \in \widehat{\mathtt{equal}}(\widehat{u}_n, \widehat{v}_n)\}$$
$$\cup \{\mathtt{Fail} \mid \mathtt{Fail} \in \widehat{\mathtt{equal}}(\widehat{u}_1, \widehat{v}_1) \ldots \vee \mathtt{Fail} \in \widehat{\mathtt{equal}}(\widehat{u}_n, \widehat{v}_n)\}.$$

  By analyzing α, we get

$$X \subseteq \{(\mathsf{Cons}\ c[u_1 \ldots u_n], \mathsf{Cons}\ c[v_1 \ldots v_n]) \mid \forall i.\ (u_i, v_i) \in \gamma(\widehat{u}_i, \widehat{v}_i)\}.$$

  Furthermore, for all $(\mathsf{Cons}\ c[u_1 \ldots u_n], \mathsf{Cons}\ c[v_1 \ldots v_n]) \in X$, we know by the induction hypothesis that exists an $\widehat{w}_i \in \widehat{\mathtt{equal}}(\widehat{u}_i, \widehat{v}_i)$ such that $\alpha(\mathsf{equal}(u_i, v_i)) \sqsubseteq \widehat{w}_i$ or that $\alpha(\mathsf{equal}(u_i, v_i)) \sqsubseteq \mathtt{Fail} \in \widehat{\mathtt{equal}}(\widehat{u}_i, \widehat{v}_i)$, for all $1 \leq i \leq n$. This lets us conclude that

$$\alpha(\mathsf{equal}(\mathsf{Cons}\ c[u_1 \ldots u_n], \mathsf{Cons}\ c[v_1 \ldots v_n])) \sqsubseteq \mathsf{Cons}\ c[\widehat{u}_1 \ldots \widehat{u}_n], \mathsf{Cons}\ c[\widehat{v}_1 \ldots \widehat{v}_n]) \in \widehat{\mathtt{equal}}(\alpha(X))$$

  or

$$\alpha(\mathsf{equal}(\mathsf{Cons}\ c[u_1 \ldots u_n], \mathsf{Cons}\ c[v_1 \ldots v_n])) \sqsubseteq \mathtt{Fail} \in \widehat{\mathtt{equal}}(\alpha(X))$$

- Case $\alpha(X) = (\mathtt{Wildcard}, \widehat{t_2})$, then $\widehat{\mathtt{equal}}(\mathtt{Wildcard}, \widehat{t_2}) = \{\widehat{t_2}, \mathtt{Fail}\}$ and for all $(t_1, t_2) \in X \subseteq \mathtt{Term} \times \{t_2 \in \gamma(\widehat{t_2})\}$, either $\mathsf{equal}(t_1, t_2) = t_1 = t_2$ or $\mathsf{equal}(t_1, t_2) = \mathtt{Fail}$. In the first case, $\alpha(t_2) \sqsubseteq \widehat{t_2} \in \widehat{\mathtt{equal}}(\mathtt{Wildcard}, \widehat{t_2})$. In the second case, $\alpha(\mathtt{Fail}) \sqsubseteq \mathtt{Fail} \in \widehat{\mathtt{equal}}(\mathtt{Wildcard}, \widehat{t_2})$.
- Case $\alpha(X) = (t_1, \mathtt{Wildcard})$, analogously.
- The catch all case is trivial.

□

Lemma 29 (Term construction operations are sound)**.** In particular, we prove soundness of (i) cons, (ii) `stringLiteral`, (iii) `numberLiteral`.

Proof. For (i) we show $\forall (c, ts) \in X.\ \exists \widehat{y} \in \widehat{\mathtt{cons}}(\alpha(X)).\ \alpha(\mathsf{cons}(c, ts)) \sqsubseteq \widehat{y}$. By analysis of α, we know that for every $(c, ts) \in X$, there exists an $(c, \widehat{ts}) \in \alpha(X)$ with $\alpha((c, ts)) \sqsubseteq (c, \widehat{ts})$. We conclude $\alpha(\mathsf{cons}(c, ts)) = \alpha(\mathsf{Cons}\ c\ ts) \sqsubseteq \mathsf{Cons}\ c\widehat{ts} \in \widehat{\mathtt{cons}}(c, \widehat{ts})$.

Analogously, we prove (ii) and (iii). □

Lemma 30 (mapping over subterms is sound)**.** Proof. To show: for all $t \in X$, there exists a $\widehat{t'} \in \widehat{\mathtt{mapSubTerms}}\ \widehat{f}\ \alpha(X)$, such that $\alpha(\mathtt{mapSubTerms}\ f\ t) \sqsubseteq \widehat{t'}$.

- Case $\alpha(X) = \mathtt{Cons}\ c\ ts$, $\alpha(X) = \mathtt{Stringliteral}\ s$ and $\alpha(X) = \mathtt{Numberliteral}\ n$ follows from the assumption $f \dot{\sqsubseteq} \widehat{f}$, Lemma 29 and **??**.
- Case $\alpha(X) = \mathtt{Wildcard}$, then $\alpha(\{\mathtt{mapSubTerms}\ f\ t \mid t \in X\}) \sqsubseteq \{\mathtt{Success}\,\mathtt{Wildcard}, \mathtt{Fail}\} = \widehat{\mathtt{mapSubTerms}}\ \widehat{f}\ \mathtt{Wildcard}$.

□

Lemma 31 (Converting a term from a list is sound)**.** Proof. The operation convertFromList takes two arguments, a term StringLiteral s that encodes the constructor and a second term

$$\texttt{Cons "Cons" } [x_1, \ldots \texttt{Cons "Cons" } [x_n, \texttt{Cons "Nil"[]}] \ldots],$$

that encodes the list of subterms. If the arguments have the required format, convertFromList constructs a corresponding term Cons $s[x_1 \ldots x_n]$, otherwise the operation fails.

The operation $\widehat{\texttt{convertFromList}}$ operates the same way if the terms do not contain a wildcard. However, if one of the argument terms contains a wildcard, $\widehat{\texttt{convertFromList}}$ returns {Fail, SuccessWildcard}, which over-approximates the results of convertFromList. □

Definition 32. The ordering on abstract term environments is defined by

$$\widehat{\rho} \sqsubseteq \widehat{\rho}' \quad \text{iff} \quad \mathrm{dom}(\widehat{\rho}') \subseteq \mathrm{dom}(\widehat{\rho}) \ \wedge \ \forall v \in \mathrm{dom}(\widehat{\rho}').\, \widehat{\rho}(v) \sqsubseteq \widehat{\rho}'(v).$$

Lemma 33. The ordering on abstract term environments is complete, where least upper bounds are the intersection of term environments:

$$\widehat{\rho} \sqcup \widehat{\rho}' = [v \mapsto \widehat{\rho}(v) \sqcup \widehat{\rho}'(v) \mid v \in \mathit{dom}(\widehat{\rho}) \cap \mathit{dom}(\widehat{\rho}')]$$

□

The ordering for abstract term environments was designed that the following property holds:

Lemma 34. For all $X \subseteq \texttt{Env}$, if $v \in \mathrm{dom}(\alpha(X))$ then $v \in \mathrm{dom}(\rho)$ and $\alpha(\rho)(v) \sqsubseteq \alpha(X)(v)$ for all $\rho \in X$. □

Lemma 35 (Variable lookup is sound)**.** The semantics of lookup and $\widehat{\texttt{lookup}}$ is

$$\texttt{lookup}\ f\ g\ (v, \rho, x) = \begin{cases} f(\rho(v))(\rho), & v \in \mathit{dom}(\rho) \\ g(x)(\rho), & \text{otherwise} \end{cases}$$

$$\widehat{\texttt{lookup}}\ \widehat{f}\ \widehat{g}\ (v, \widehat{\rho}, x) = \begin{cases} \widehat{f}(\widehat{\rho}(v))(\widehat{\rho}), & v \in \mathrm{dom}(\widehat{\rho}) \\ \widehat{g}(x)(\widehat{\rho}) \sqcup & \\ \quad \widehat{f}(\texttt{Wildcard})(\widehat{\rho}[v \mapsto \texttt{Wildcard}]), & \text{otherwise} \end{cases}$$

Proof. Let $(v, \widehat{\rho}, \widehat{x}) = \alpha(X)$ and $\widehat{t} = \alpha(\{\rho(v) \mid (v, x, \rho) \in X \wedge v \in \mathit{dom}(\rho)\})$, then there are two cases:

- Case $v \in \mathrm{dom}(\widehat{\rho})$, then by analysis of α, all concrete environments ρ in $(v, \rho, x) \in X$ include the variable v, i.e., $v \in \mathrm{dom}(\rho)$ and $\alpha(\rho(v)) \sqsubseteq \widehat{\rho}(v)$. Furthermore, by $f \mathrel{\dot{\sqsubseteq}} \widehat{f}$ we know that there exists a $\widehat{y} \in \widehat{f}(\widehat{t})(\widehat{\rho})$ with $\alpha(f(\rho(v))(\rho)) \sqsubseteq \widehat{y}$. We conclude $\alpha(\texttt{lookup}\ f\ g\ (v, \rho, x)) = \alpha(f(\rho(v))(\rho)) \sqsubseteq \widehat{y} \in \widehat{f}(\alpha(\rho(v)))(\alpha(\rho)) \sqsubseteq \widehat{\texttt{lookup}}\ \widehat{f}\ \widehat{g}\ \alpha(X)$.
- Case $v \notin \mathrm{dom}(\widehat{\rho})$, then for all $(v, \rho, x) \in X$ if $v \in \mathrm{dom}(\rho)$,

$$\begin{aligned} \alpha(\texttt{lookup}\ f\ g\ (v, \rho, x)) &= \alpha(f(\rho(v))(\rho)) \sqsubseteq \widehat{y} \in \widehat{f}(\widehat{t})(\widehat{\rho}) \\ &\sqsubseteq \widehat{f}(\texttt{Wildcard})(\widehat{\rho}[v \mapsto \texttt{Wildcard}]) \sqsubseteq \widehat{\texttt{lookup}}\ \widehat{f}\ \widehat{g}\ \alpha(X) \end{aligned}$$

and if $v \notin \mathrm{dom}(\rho)$, it holds $\alpha(\texttt{lookup}\ f\ g\ (v, \rho, x)) = \alpha(g(x)(\rho)) \sqsubseteq \widehat{y} \in \widehat{g}(\widehat{x})(\widehat{\rho}) \sqsubseteq \widehat{\texttt{lookup}}\ \widehat{f}\ \widehat{g}\ \alpha(X)$.

□

Lemma 36. (i) Extending, (ii) removing from environments, and (iii) left-biased union two environments is sound.

Proof.

(i) Let $(\widehat{\rho}, \widehat{t}) = \alpha(X)$ for all $X \subseteq \texttt{Env} \times \texttt{Term}$, we show for all $(\rho, t) \in X$, $\alpha(\rho[v \mapsto t]) \sqsubseteq \widehat{\rho}[v \mapsto \widehat{t}]$. That is

$$\begin{aligned}
&\mathrm{dom}(\widehat{\rho}[v \mapsto \widehat{t}]) \subseteq \mathrm{dom}(\alpha(\rho[v \mapsto t])) \Leftrightarrow \\
&\mathrm{dom}(\widehat{\rho}[v \mapsto \widehat{t}]) \subseteq \mathrm{dom}(\rho[v \mapsto t]) \Leftrightarrow \\
&\qquad \mathrm{dom}(\widehat{\rho}) \subseteq \mathrm{dom}(\rho) && \text{by Lemma 34.}
\end{aligned}$$

Furthermore, for all $w \in \mathrm{dom}(\widehat{\rho}[v \mapsto \widehat{t}])$, we have to show that

$$\begin{aligned}
\alpha(\rho[v \mapsto t])(w) \sqsubseteq \widehat{\rho}[v \mapsto \widehat{t}](w) \Leftrightarrow && \text{by } \alpha(t) \sqsubseteq \widehat{t} \\
\alpha(\rho)(w) \sqsubseteq \widehat{\rho}(w) && \text{by Lemma 34}
\end{aligned}$$

(ii) Analogously to (i).

(iii) The union of two environments has the following semantics $\texttt{union}(\rho, \rho', vs) = \rho \cup (\rho' \setminus vs \cup \mathrm{dom}(\rho))$, where vs is a set of variables. Let $(\widehat{\rho}, \widehat{\rho}', vs) = \alpha(X)$, we show that for all $(\rho, \rho', vs) \in X$, $\alpha(\texttt{union}(\rho, \rho', vs)) \sqsubseteq \widehat{\texttt{union}}(\widehat{\rho}, \widehat{\rho}', vs)$. That is $\alpha(\rho \cup (\rho' \setminus vs \cup \mathrm{dom}(\rho)))) \sqsubseteq (\widehat{\rho} \cup (\widehat{\rho}' \setminus vs \cup \mathrm{dom}(\widehat{\rho})))$. It follows,

$$\begin{aligned}
&\mathrm{dom}(\widehat{\rho} \cup (\widehat{\rho}' \setminus vs \cup \mathrm{dom}(\widehat{\rho}))) \subseteq \mathrm{dom}(\rho \cup (\rho' \setminus vs \cup \mathrm{dom}(\rho))) \Leftarrow \\
&\mathrm{dom}(\widehat{\rho}) \subseteq \mathrm{dom}(\rho) \wedge \mathrm{dom}(\widehat{\rho}' \setminus vs \cup \mathrm{dom}(\widehat{\rho})) \subseteq \mathrm{dom}(\rho' \setminus vs \cup \mathrm{dom}(\rho)) \Leftrightarrow \\
&\mathrm{dom}(\widehat{\rho}) \subseteq \mathrm{dom}(\rho) \wedge \mathrm{dom}(\widehat{\rho}') \subseteq \mathrm{dom}(\rho') && \text{by Lemma 34}
\end{aligned}$$

Furthermore, we have to show for all $w \in \mathrm{dom}(\widehat{\rho} \cup (\widehat{\rho}' \setminus vs \cup \mathrm{dom}(\widehat{\rho})))$,

$$\alpha(\rho \cup (\rho' \setminus vs \cup \mathrm{dom}(\rho)))(w) \sqsubseteq (\widehat{\rho} \cup (\widehat{\rho}' \setminus vs \cup \mathrm{dom}(\widehat{\rho})))(w).$$

- Case $w \in \mathrm{dom}(\widehat{\rho})$, then $w \in \rho$ and $\alpha(\rho)(w) \sqsubseteq \widehat{\rho}(w)$ holds because of Lemma 34.
- Case $w \in \mathrm{dom}(\widehat{\rho}' \setminus vs \cup \mathrm{dom}(\widehat{\rho}))$, then $w \in \mathrm{dom}(\rho' \setminus vs \cup \mathrm{dom}(\rho))$ and in particular $w \in \mathrm{dom}(\widehat{\rho}')$ and $w \in \mathrm{dom}(\rho')$. We have to show that $\alpha(\rho')(w) \sqsubseteq \widehat{\rho}'(w)$, which again holds because of Lemma 34.

□