

TxForest: Composable Transactions over Filestores

Forest Team
Cornell, TUFTS
hpacheco@cs.cornell.edu

Abstract

Keywords

1. Introduction

transactional use cases: batch changes on group of files (process all the files in a directory) software upgrade (rollback) concurrent file access (multiple processes writing to the same log file) filesystem as a database (ACID guarantees)

transactional filesystems <http://www.fuzzy.cz/en/articles/transactional-file-systems> http://www.fsl.cs.sunysb.edu/docs/valor/valor_fast2009.pdf <http://www.fsl.cs.sunysb.edu/docs/amino-tos06/amino.pdf>

libraries for transactional file operations: <http://commons.apache.org/proper/commons-transaction/file/index.html> <https://xadisk.java.net/> <https://transactionalfilemgr.codeplex.com/>

tx file-level operations (copy,create,delete,move,write) schema somehow equivalent to using the unstructured universal Forest representation

but what about data manipulation: transactional maps,etc?

2. Examples

3. The Forest Language

the forest description types

a forest description defines a structured representation of a semi-structured filestore.

4. Forest Transactions

The key goal of this paper is to make Forest [1] transactional. As an embedded DSL in Haskell, we borrow the elegant software transactional memory (STM) interface from its host language.

4.1 Composable transactions

software transactional memory building blocks

```
-- running transactions
atomically :: FTM a → IO a
-- blocking
```

```
retry      :: FTM a
-- nested transactions
orElse     :: FTM a → FTM a → FTM a
-- exceptions
throw      :: Exception e ⇒ e → FTM a
catch      :: Exception e ⇒ FTM a → (e → FTM a) → FTM a
```

$FTM\ a$ denotes a transactional action that returns a value of type a . Complex transactions can be defined by composing FTM actions, and run *atomically* as an IO action. In Haskell, IO is the type of non-revocable I/O operations, including reading/writing to files.

arbitrary pure code

4.2 Transactional variables

the forest programming style draws no distinction between data represented on disk and in memory.

The transactional forest compiler generates several Haskell types and functions from every forest type declaration, aggregated as an instance of the *TxForest* class:

programmers can manipulate in-memory representations as if they were working on the filestore itself.

each user-declared forest type ty corresponds to a transactional variable that holds a representation of type rep .

```
class TxForest args ty rep where
  new :: args → FilePath → FTM fs ty
  read  :: ty → FTM rep
  writeOrElse :: ty → rep → b → (WriteErrors → FTM fs b) → FTM b
```

new creates a new forest transactional variable for the specification found in *TxForest* context, with arbitrary arguments and a root path. *read* the associated fragment of the filesystem into an in-memory representation data structure. Users can manipulate these structures as they would in regular Haskell programs, and eventually perform FS modifications by writing new representation to a transactional variable. *writes* may fail if the provided data is not a faithful representation of the filestore for the specification under consideration.

WriteErrors have nothing to do with transactional errors and account for the inconsistencies that can arise when programmer attempt to write an erroneous in-memory representation to the filestore. e.g., attempting to write conflicting data to the same file. more on this later in sec...

the *rep* of a variable may itself contain other variables. simple programming example.

Notice that we can have multiple variables (with possibly different specs) “connected” to the same fragment of a filesystem. this can cause inter

we have sort of a mismatch: transactional variables per type declaration VS fileinfo per directory/file. the justification is that, since forest always fills default data for inexistent paths, the fileinfo actually determines whether a directory/file exists or not in the real

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CONF 'yy, Month d-d, 20yy, City, ST, Country.
Copyright © 20yy ACM 978-1-xxxx-xxxx-n/yy/mm...\$15.00.
<http://dx.doi.org/10.1145/nnnnnnnn.nnnnnnn>

FS. e.g., to delete a file we need to mark its fileinfo as invalid, and to create a file we need to define clean valid fileinfo for it.

SIGPLAN International Conference on Functional Programming, ICFP '11, pages 292–306. ACM, 2011.

4.3 Validation

forest dependencies and constraints that can not be statically checked.

$validate :: TxForest\ args\ ty\ rep \Rightarrow ty \rightarrow FTM\ ValidationErrors$

4.4 Standard filesystem operations

$rm :: TxForest\ args\ ty\ rep \Rightarrow ty \rightarrow FTM\ ()$

remove a filepath by writing invalid fileinfo and default data to it. to avoid loss of information, the default data needs to be precisely the one that is generated by forest. if we are removing a directory, we need to make sure that its content is the empty list: e.g., a non-existing directory with content inside is not a valid snapshot of a FS, but a valid haskell value nonetheless. cumbersome for arbitrary specs that touch multiple files/directories. primitive operation that generates the appropriate default data.

$copyOrElse :: TxForest\ args\ ty\ rep \Rightarrow ty \rightarrow ty \rightarrow b \rightarrow ([\ WriteErrors] \rightarrow FTM\ fs\ b) \rightarrow FTM\ fs\ b$

copy a forest specification. copying a single file by hand is simple: read, copy the contents and update the fileinfo, write. copying a directory...not so much...because we have to recursively copy each child variable and update its fileinfo accordingly. primitive operation may fail, because the data that we are trying to write may not be consistent with the specification for the target arguments/path. e.g., a spec with a boolean argument that loads file x or y, with source arg = True and target arg = False.

5. Implementation

5.1 Transactional Forest

transactional semantics of STM: we log reads/writes to the filesystem instead of variables. global lock, no equality check on validation. load/store semantics of Forest with thunks, explicit laziness

transactional variables created by calling load on its spec with given arguments and root path; lazy loading, so no actual reads occur. Additionally to the representation data, each transactional variable remembers its creation-time arguments (they never change).

each transaction keeps a local filesystem version number, and a per-tvar log mapping fsversions to values, stored in a weaktable (fsversions are purgeable once a tx commits).

on writes: backup the current fslog, increment the fsversion, add an entry to the table for the (newfsversion,newvalue), run the store function for the new data and writing the modifications to the buffered FS; if there are errors, rollback to the backed-up FS and the previous fsversion.

the store function also changes the in-memory representation by recomputing the validation thunks (hidden to users) to match the new content.

write success theorem: if the current rep is in the image of load, then store succeeds

5.2 Incremental Transactional Forest

5.3 Log-structured Transactional Forest

6. Evaluation

7. Related Work

8. Conclusions

References

- [1] K. Fisher, N. Foster, D. Walker, and K. Q. Zhu. Forest: A language and toolkit for programming with filestores. In *Proceedings of the 16th ACM*

A. Forest Semantics

$$F^*(r / u) = \begin{cases} F^*(r') & \text{if } F(F^*(r) / u) = (i, \text{Link } r') \\ F^*(r) / u & \text{otherwise} \end{cases}$$

$$F^*(\cdot) = \cdot$$

$$\frac{}{r \in \cdot} \quad \frac{}{r \in r} \quad \frac{r \in r'}{r / u \in r'}$$

$$F \setminus_{\mathcal{X}} r \triangleq F|_{\{\forall r'. F^*(r') \in r\}}$$

$$F = F' = \forall r \in rs. F \setminus_{\mathcal{X}} r = F' \setminus_{\mathcal{X}} r$$

$$Err \ a = (\mathbf{M} \ \mathit{Bool}, a)$$

s	$\mathcal{R} \llbracket s \rrbracket$	$\mathcal{C} \llbracket s \rrbracket$
$\mathbf{M} \ s$	$\mathbf{M} (Err (\mathcal{R} \llbracket s \rrbracket))$	$\mathbf{M} (Err (\mathcal{C} \llbracket s \rrbracket))$
$k_{\tau_1}^{\tau_2}$	$Err (\tau_2, \tau_1)$	(τ_2, τ_1)
$e :: s$	$\mathcal{R} \llbracket s \rrbracket$	$\mathcal{C} \llbracket s \rrbracket$
$\langle x : s_1, s_2 \rangle$	$Err (\mathcal{R} \llbracket s_1 \rrbracket, \mathcal{R} \llbracket s_2 \rrbracket)$	$(\mathcal{C} \llbracket s_1 \rrbracket, \mathcal{C} \llbracket s_2 \rrbracket)$
$\{s \mid x \in e\}$	$Err [\mathcal{R} \llbracket s \rrbracket]$	$[\mathcal{C} \llbracket s \rrbracket]$
$\mathbf{P} (e)$	$Err ()$	$()$
$s?$	$Err (\mathit{Maybe} (\mathcal{R} \llbracket s \rrbracket))$	$\mathit{Maybe} (\mathcal{C} \llbracket s \rrbracket)$

$\mathcal{R} \llbracket \cdot \rrbracket$ is the internal in-memory representation type of a forest declaration; $\mathcal{C} \llbracket \cdot \rrbracket$ is the external type of content of a variables that users can inspect/modify

$$err(a) = \mathbf{do} \{ e \leftarrow \mathbf{get} \ a; (a_{err}, v) \leftarrow e; \mathbf{return} \ a_{err} \}$$

$$err(a_{err}, v) = \mathbf{return} \ a_{err}$$

$$valid(v) = \mathbf{do} \{ a_{err} \leftarrow err \ v; e_{err} \leftarrow \mathbf{get} \ a_{err}; e_{err} \}$$

$v_1 \ \Theta_1 \sim_{\Theta_2} v_2$ denotes value equivalence modulo memory addresses, under the given environments. $e_1 \ \Theta_1 \sim_{\Theta_2} e_2$ denotes expression equivalence by evaluation modulo memory addresses, under the given environments.

$v_1 \ \Theta_1 \overset{err}{\sim}_{\Theta_2} v_2$ denotes value equivalence (ignoring error information) modulo memory addresses, under the given environments.

$$\boxed{\Theta; \varepsilon; r; s \vdash \mathbf{load} \ F \Rightarrow \Theta'; v}$$

$$\boxed{s = \mathbf{M} \ s_1}$$

$$\frac{a \notin \mathbf{dom}(\Theta) \quad a_{err} \notin \mathbf{dom}(\Theta) \quad e = \varepsilon; r; \mathbf{M} \ s \vdash \mathbf{load} \ F \quad e_{err} = \mathbf{do} \{ e_1 \leftarrow \mathbf{get} \ a; v_1 \leftarrow e_1; valid \ v_1 \}}{\Theta; \varepsilon; r; \mathbf{M} \ s \vdash \mathbf{load} \ F \Rightarrow \Theta[a_{err} : e_{err}, a : e]; (a_{err}, a)}$$

$$\boxed{s = k}$$

$$\frac{a_{err} \notin \mathbf{dom}(\Theta) \quad \Theta; \mathbf{load}_k(\varepsilon, F, r) \Rightarrow \Theta'; (b, v)}{\Theta; \varepsilon; r; k \vdash \mathbf{load} \ F \Rightarrow \Theta'[a_{err} : \mathbf{return} \ b]; (a_{err}, v)}$$

$$\mathbf{load}_{\mathbf{File}}(\varepsilon, F, r) \begin{cases} \mathbf{return} \ (True, (i, u)) & \text{if } F(r) = (i, \mathbf{File} \ u) \\ \mathbf{return} \ (False, (i_{\mathbf{invalid}}, "")) & \text{otherwise} \end{cases}$$

$$\mathbf{load}_{\mathbf{Dir}}(\varepsilon, F, r) \begin{cases} \mathbf{return} \ (True, (i, us)) & \text{if } F(r) = (i, \mathbf{Dir} \ us) \\ \mathbf{return} \ (False, (i_{\mathbf{invalid}}, \{ \})) & \text{otherwise} \end{cases}$$

$$\mathbf{load}_{\mathbf{Link}}(\varepsilon, F, r) \begin{cases} \mathbf{return} \ (True, (i, r')) & \text{if } F(r) = (i, \mathbf{Link} \ r') \\ \mathbf{return} \ (False, (i_{\mathbf{invalid}}, \cdot)) & \text{otherwise} \end{cases}$$

$$\boxed{s = e :: s_1}$$

$$\frac{\Theta; \llbracket r / e \rrbracket_{Path}^{\varepsilon} \Rightarrow \Theta'; r' \quad \Theta; \varepsilon; r'; s \vdash \mathbf{load} \ F \Rightarrow \Theta''; v}{\Theta; \varepsilon; r; e :: s \vdash \mathbf{load} \ F \Rightarrow \Theta''; v}$$

$$\boxed{s = \langle x : s_1, s_2 \rangle}$$

$$\frac{\Theta; \varepsilon; r; s_1 \vdash \mathbf{load} \ F \Rightarrow \Theta_1; v_1 \quad \Theta_1; \varepsilon[x \mapsto v_1]; r; s_2 \vdash \mathbf{load} \ F \Rightarrow \Theta_2; v_2 \quad e_{err} = \mathbf{do} \{ b_1 \leftarrow valid(v_1); b_2 \leftarrow valid(v_2); \mathbf{return} \ (b_1 \wedge b_2) \}}{\Theta; \varepsilon; r; \langle x : s_1, s_2 \rangle \vdash \mathbf{load} \ F \Rightarrow \Theta_2[a_{err} : e_{err}]; (a_{err}, (v_1, v_2))}$$

$$s = P e$$

$$\frac{a_{err} \notin \text{dom}(\Theta)}{\Theta; \varepsilon; r; P e \vdash \text{load } F \Rightarrow \Theta[a_{err} : \llbracket e \rrbracket_{Bool}^\varepsilon]; (a_{err}, ())}$$

$$s = s_1?$$

$$\frac{r \notin \text{dom}(F) \quad a_{err} \notin \text{dom}(\Theta)}{\Theta; \varepsilon; r; s? \vdash \text{load } F \Rightarrow \Theta[a_{err} : \text{return } True]; (a_{err}, \text{Nothing})}$$

$$\frac{r \in \text{dom}(F) \quad a_{err} \notin \text{dom}(\Theta') \quad \Theta; \varepsilon; r; s \vdash \text{load } F \Rightarrow \Theta'; v}{\Theta; \varepsilon; r; s? \vdash \text{load } F \Rightarrow \Theta[a_{err} : \text{valid}(v)]; (a_{err}, \text{Just } v)}$$

$$s = \{s_1 \mid x \in e\}$$

$$\frac{a_{err} \notin \text{dom}(\Theta) \quad \Theta; \llbracket e \rrbracket_{\{\tau\}}^\varepsilon \Rightarrow \Theta'; \{t_1, \dots, t_k\} \quad \Theta'; \forall i \in \{1, \dots, k\}. \text{do } \{v_i \leftarrow \varepsilon[x \mapsto t_i]; r; s \vdash \text{load } F; \text{return } \{t_i \mapsto v_i\}\} \Rightarrow \Theta''; vs \quad e_{err} = \forall i \in \{1, \dots, k\}. \text{do } \{b_i \leftarrow \text{valid}(vs(t_i)); \text{return } (\bigwedge b_i)\}}{\Theta; \varepsilon; r; \{s \mid x \in e\} \vdash \text{load } F \Rightarrow \Theta''[a_{err} : e_{err}]; (a_{err}, vs)}$$

$$\Theta; \varepsilon; r; s \vdash \text{store } F v \Rightarrow \Theta'; (F', \phi')$$

$$s = M s_1$$

$$\frac{\Theta(a) = e \quad \Theta; e \Rightarrow \Theta'; (a_{err}, v) \quad \Theta'; \varepsilon; r; s \vdash \text{store } F v \Rightarrow \Theta''; (F', \phi')}{\Theta; \varepsilon; r; M s \vdash \text{store } F a \Rightarrow \Theta''; (F', \phi')}$$

$$s = k$$

$$\frac{\Theta; \text{store}_k(\varepsilon, F, r, (d, v)) \Rightarrow \Theta'; (F', \phi)}{\Theta; \varepsilon; r; k \vdash \text{store } F (a_{err}, (d, v)) \Rightarrow \Theta'; (F', \phi)}$$

$$\text{store}_{\text{File}}(\varepsilon, F, r, (i, u)) \begin{cases} \text{return } (F[r := (i, \text{File } u)], \lambda F'. F'(r) = (i, \text{File } u)) & \text{if } i \neq i_{\text{invalid}} \\ \text{return } (F[r := \perp], \lambda F'. F'(r) \neq (-, \text{File } -)) & \text{if } i = i_{\text{invalid}} \wedge F(r) = (-, \text{File } -) \\ \text{return } (F, \lambda F'. F'(r) \neq (-, \text{File } -)) & \text{if } i = i_{\text{invalid}} \wedge F(r) \neq (-, \text{File } -) \end{cases}$$

$$\text{store}_{\text{Dir}}(\varepsilon, F, r, (i, \{u_1, \dots, u_n\})) \begin{cases} \text{return } (F[r := (i, \text{Dir } \{u_1, \dots, u_n\})], \lambda F'. F'(r) = (i, \text{Dir } \{u_1, \dots, u_n\})) & \text{if } i \neq i_{\text{invalid}} \\ \text{return } (F[r := \perp], \lambda F'. F'(r) \neq (-, \text{Dir } -)) & \text{if } i = i_{\text{invalid}} \wedge F(r) = (-, \text{Dir } -) \\ \text{return } (F, \lambda F'. F'(r) \neq (-, \text{Dir } -)) & \text{if } i = i_{\text{invalid}} \wedge F(r) \neq (-, \text{Dir } -) \end{cases}$$

$$\text{store}_{\text{Link}}(\varepsilon, F, r, (i, r')) \begin{cases} \text{return } (F[r := (i, \text{Link } r')], \lambda F'. F'(r) = (i, \text{Link } r')) & \text{if } i \neq i_{\text{invalid}} \\ \text{return } (F[r := \perp], \lambda F'. F'(r) \neq (-, \text{Link } -)) & \text{if } i = i_{\text{invalid}} \wedge F(r) = (-, \text{Link } -) \\ \text{return } (F, \lambda F'. F'(r) \neq (-, \text{Link } -)) & \text{if } i = i_{\text{invalid}} \wedge F(r) \neq (-, \text{Link } -) \end{cases}$$

$$s = e :: s_1$$

$$\frac{\Theta; \llbracket e \rrbracket_{\text{Path}}^\varepsilon \Rightarrow \Theta'; r' \quad \Theta'; \varepsilon; r'; s \vdash \text{store } F v \Rightarrow \Theta''; (F', \phi')}{\Theta; \varepsilon; r; e :: s \vdash \text{store } F v \Rightarrow \Theta''; (F', \phi')}$$

$$s = \langle x : s_1, s_2 \rangle$$

$$\frac{\Theta; \varepsilon; r; s_1 \vdash \text{store } F v_1 \Rightarrow \Theta_1; (F_1, \phi_1) \quad \Theta_1; \varepsilon[x \mapsto v_1]; r; s_2 \vdash \text{store } F v_2 \Rightarrow \Theta_2; (F_2, \phi_2) \quad \phi = \lambda F'. \phi_1(F') \wedge \phi_2(F')}{\Theta; \varepsilon; r; \langle x : s_1, s_2 \rangle \vdash \text{store } F (a_{err}, (v_1, v_2)) \Rightarrow \Theta_2; (F_1 \uplus F_2, \phi)}$$

$$s = P e$$

$$\frac{\phi = \lambda F'. \text{True}}{\Theta; \varepsilon; r; P e \vdash \text{store } F (a_{err}, ()) \Rightarrow \Theta; (F, \phi)}$$

$$s = s_1?$$

$$\frac{\phi = \lambda F'. r \notin \text{dom}(F')}{\Theta; \varepsilon; r; s? \vdash \text{store } F (a_{err}, \text{Nothing}) \Rightarrow \Theta; (F[r := \perp], \phi)}$$

$$\frac{\Theta; \varepsilon; r; s \vdash \text{store } F \ v \Rightarrow \Theta'; (F_1, \phi_1) \quad \phi = \lambda F'. \phi_1(F') \wedge r \in \text{dom}(F')}{\Theta; \varepsilon; r; s? \vdash \text{store } F \ (a_{err}, \text{Just } v) \Rightarrow \Theta; (F_1, \phi)}$$

$$s = \{s_1 \mid x \in e\}$$

$$\frac{\Theta; \llbracket e \rrbracket_{\{\tau\}}^{\varepsilon} \Rightarrow \Theta'; ts \quad vs = \{t_1 \mapsto v_1, \dots, t_k \mapsto v_k\} \quad \phi = \lambda F'. ts = \{t_1, \dots, t_k\} \wedge \bigwedge \phi_i(F') \quad \Theta'; \forall i \in \{1, \dots, k\}. \text{do } \{(F_i, \phi_i) \leftarrow \varepsilon[x \mapsto v_i]; r; s \vdash \text{store } F \ v_i; \text{return } (F_1 \uplus \dots \uplus F_k, \phi)\} \Rightarrow \Theta''; F' \ \phi'}{\Theta; \varepsilon; r; \{s \mid x \in e\} \vdash \text{store } F \ (a_{err}, vs) \Rightarrow \Theta; (F', \phi')}$$

Proposition 1 (Load Type Safety). *If $\Theta; \varepsilon; r; s \vdash \text{load } F \Rightarrow \Theta'; v'$ and $\mathcal{R}[\llbracket s \rrbracket] = \tau$ then $\vdash v : \tau$.*

Theorem A.1 (LoadStore). *If*

$$\begin{aligned} \Theta; \varepsilon; r; s \vdash \text{load } F \Rightarrow \Theta'; v \\ \Theta''; \varepsilon; r; s \vdash \text{store } F \ v' \Rightarrow \Theta'''; (F', \phi') \\ v \sim_{\Theta''}^{err} v' \end{aligned}$$

then $F = F'$ and $\phi'(F')$.

Theorem A.2 (StoreLoad). *If*

$$\begin{aligned} \Theta; \varepsilon; r; s \vdash \text{store } F \ v \Rightarrow \Theta'; (F', \phi') \\ \Theta'; \varepsilon; r; s \vdash \text{load } F \Rightarrow \Theta''; v' \end{aligned}$$

then $\phi'(F')$ iff $v \sim_{\Theta''}^{err} v'$

stronger than the original forest theorem: store validation only fails for impossible cases (when representation cannot be stored to the FS without loss)

weaker in that we don't track consistency of inner validation variables; equality of the values is modulo error information. in a real implementation we want to repair error information on storing, so that it is consistent with a subsequent load.

the error information is not stored back to the FS, so the validity predicate ignores it.

B. Forest Incremental Semantics

Note that:

- We have access to the old filesystem, since filesystem deltas record the changes to be performed.
- We do not have access to the old environment, since variable deltas record the changes that already occurred.

$$\delta_F ::= \text{addFile}(r, u) \mid \text{addDir}(r) \mid \text{addLink}(r, r') \mid \text{rem}(r) \mid \text{chgAttrs}(r, i) \mid \delta_{F_1}; \delta_{F_2} \mid \emptyset$$

$$\begin{aligned} \delta_v &::= M_{\delta_a} \delta_{v_1} \mid \delta_{v_1} \otimes \delta_{v_2} \mid \{t_i \mapsto \delta_{\perp v_i}\} \mid \delta_{v_1} ? \mid \emptyset \mid \Delta \\ \delta_{\perp v} &::= \perp \mid \delta_v \end{aligned}$$

$$\Delta_v ::= \emptyset \mid \Delta$$

$$\begin{aligned} (\text{addFile}(r', u)) \searrow_F r &\triangleq \text{if } F^*(r') \in F^*(r) \text{ then } \text{addFile}(r', u) \text{ else } \emptyset \\ (\text{addDir}(r')) \searrow_F r &\triangleq \text{if } F^*(r') \in F^*(r) \text{ then } \text{addDir}(r') \text{ else } \emptyset \\ (\text{addLink}(r', r'')) \searrow_F r &\triangleq \text{if } F^*(r') \in F^*(r) \text{ then } \text{addLink}(r', r'') \text{ else } \emptyset \\ (\text{rem}(r')) \searrow_F r &\triangleq \text{if } F^*(r') \in F^*(r) \text{ then } \text{rem}(r') \text{ else } \emptyset \\ (\text{chgAttrs}(r', i)) \searrow_F r &\triangleq \text{if } F^*(r') \in F^*(r) \text{ then } \text{chgAttrs}(r', i) \text{ else } \emptyset \\ (\delta_{F_1}; \delta_{F_2}) \searrow_F r &\triangleq \delta_{F_1} \searrow_F r; \delta_{F_2} \searrow_{F_1} r \text{ where } F_1 = (\delta_{F_1} \searrow_F r) \ F \\ \emptyset \searrow_F r &\triangleq \emptyset \end{aligned}$$

$$\Theta; v \xrightarrow{\delta_v} \Theta'; v'$$

the value delta maps v to v'

monadic expressions only read from the store and perform new allocations; they can't modify existing addresses.

For any expression application $e \ \Theta = (\Theta', v)$, we have $\Theta = \Theta \cap \Theta'$.

errors are computed in the background

$$\frac{a' \notin \text{dom}(\Theta)}{\Theta; \delta_a; \Delta_e \vdash a : e \Rightarrow \Theta[a' : e]; (a', \Delta)} \quad \frac{}{\Theta; \emptyset; \Delta_e \vdash a : e \Rightarrow \Theta[a : e]; (a, \Delta)} \quad \frac{}{\Theta; \emptyset; \emptyset \vdash a : e \Rightarrow \Theta; (a, \emptyset)}$$

$$\Theta; \varepsilon; \Delta_e; r; s \vdash \text{load}_{\Delta} F \ v \ \delta_F \ \delta_v \Rightarrow \Theta'; (v', \Delta'_v)$$

$$\frac{\Delta_\varepsilon|_{fv(s)} = \emptyset \quad \delta_F \searrow_F r = \emptyset}{\Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash \text{load}_\Delta F v \delta_F \emptyset \Rightarrow \Theta; (v, \emptyset)}$$

$$\frac{\Theta; \varepsilon; r; s \vdash \text{load} (\delta_F F) \Rightarrow \Theta'; v'}{\Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash \text{load}_\Delta F v \delta_F \delta_v \Rightarrow \Theta'; (v', \Delta)}$$

$$s = M s_1$$

$$\frac{\Theta(a) = e \quad \Theta; e \Rightarrow \Theta'; (a_{err}, v) \quad \Theta'; \varepsilon; \Delta_\varepsilon; r; s \vdash \text{load}_\Delta F v \delta_F \delta_v \Rightarrow \Theta''; (v', \Delta_v) \quad v = v'}{\Theta; \varepsilon; \Delta_\varepsilon; r; M s \vdash \text{load}_\Delta F a \delta_F (M_\emptyset (\emptyset \otimes \delta_v)) \Rightarrow \Theta''; (a, \emptyset)}$$

$$\frac{\Theta(a) = e \quad \Theta; e \Rightarrow \Theta'; (a_{err}, v) \quad \Theta'; \varepsilon; \Delta_\varepsilon; r; s \vdash \text{load}_\Delta F v \delta_F \delta_v \Rightarrow \Theta_1; (v', \Delta_v) \quad \Theta_1; \delta_{a_{err}}; \Delta_v \vdash a_{err} : \text{valid } v' \Rightarrow \Theta_2; (a'_{err}, \Delta_{a_{err}}) \quad \Theta_2; \delta_a; \Delta_{a_{err}} \vdash a : \text{return } (a'_{err}, v') \Rightarrow \Theta_3; (a', \Delta_a)}{\Theta; \varepsilon; \Delta_\varepsilon; r; M s \vdash \text{load}_\Delta F a \delta_F (M_{\delta_a} (\delta_{a_{err}} \otimes \delta_v)) \Rightarrow \Theta_3; (a', \Delta_a)}$$

$$s = e :: s_1$$

$$\frac{\Delta_\varepsilon|_{fv(s)} = \emptyset \quad \Theta; \llbracket r / e \rrbracket_{Path}^\varepsilon \Rightarrow \Theta'; r' \quad \Theta'; \varepsilon; \Delta_\varepsilon; r'; e :: s \vdash \text{load}_\Delta F v \delta_F \delta_v \Rightarrow \Theta''; (v', \Delta_v)}{\Theta; \varepsilon; \Delta_\varepsilon; r; e :: s \vdash \text{load}_\Delta F v \delta_F \delta_v \Rightarrow \Theta''; (v', \Delta_v)}$$

$$s = \langle x : s_1, s_2 \rangle$$

$$\frac{\Theta; \varepsilon; \Delta_\varepsilon; r; s_1 \vdash \text{load}_\Delta F v_1 \delta_F \delta_{v_1} \Rightarrow \Theta_1; (v'_1, \Delta_{v_1}) \quad \Theta_1; \varepsilon[x \mapsto v'_1]; \Delta_\varepsilon[x \mapsto \Delta_{v_1}]; r; s_2 \vdash \text{load}_\Delta F v_2 \delta_F \delta_{v_2} \Rightarrow \Theta_2; (v'_2, \Delta_{v_2}) \quad \Theta_2; \delta_{a_{err}}; (\Delta_{v_1} \wedge \Delta_{v_2}) \vdash a_{err} : \text{do } \{b_1 \leftarrow \text{valid } v'_1; b_2 \leftarrow \text{valid } v'_2; \text{return } (b_1 \wedge b_2)\} \Rightarrow \Theta'; (a'_{err}, \Delta_{a_{err}})}{\Theta; \varepsilon; \Delta_\varepsilon; r; \langle x : s_1, s_2 \rangle \vdash \text{load}_\Delta F (a_{err}, (v_1, v_2)) \delta_F (\delta_{a_{err}} \otimes (\delta_{v_1} \otimes \delta_{v_2})) \Rightarrow \Theta'; ((a'_{err}, (v'_1, v'_2)), \Delta_{a_{err}})}$$

$$s = P e$$

$$\frac{\Delta_\varepsilon|_{fv(e)} = \emptyset}{\Theta; \varepsilon; \Delta_\varepsilon; r; P e \vdash \text{load}_\Delta F v \delta_F \emptyset \Rightarrow \Theta; (v, \emptyset)}$$

$$s = s_1?$$

$$\frac{r \notin \text{dom}(\delta_F F) \quad \Theta; \delta_{a_{err}}; \delta_v \vdash a_{err} : \text{return True} \Rightarrow \Theta'; (a'_{err}, \Delta_{a_{err}})}{\Theta; \varepsilon; \Delta_\varepsilon; r; s? \vdash \text{load}_\Delta F (a_{err}, \text{Nothing}) \delta_F (\delta_{a_{err}} \otimes \delta_v) \Rightarrow \Theta'; ((a_{err}, \text{Nothing}), \Delta_{a_{err}})}$$

$$\frac{r \in \text{dom}(\delta_F F) \quad \Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash \text{load}_\Delta F v \delta_F \delta_v \Rightarrow \Theta'; (v', \Delta_v) \quad \Theta; \delta_{a_{err}}; \Delta_v \vdash a_{err} : \text{valid}(v') \Rightarrow \Theta'; (a'_{err}, \Delta_{a_{err}})}{\Theta; \varepsilon; \Delta_\varepsilon; r; s? \vdash \text{load}_\Delta F (a_{err}, \text{Just } v) \delta_F (\delta_{a_{err}} \otimes \delta_v?) \Rightarrow \Theta'; ((a_{err}, \text{Just } v'), \Delta_{a_{err}})}$$

$$s = \{s \mid x \in e\}$$

$$\frac{\Theta; \llbracket e \rrbracket_{\tau}^\varepsilon \Rightarrow \Theta_1; \{t_1, \dots, t_k\} \quad \Theta_1; \forall i \in \{1, \dots, k\}. \text{do } \{(v_i, \Delta_{v_i}) \leftarrow \varepsilon; \Delta_\varepsilon; r; s \vdash_x \text{load}_\Delta F v s \delta_F \delta_{v_s}; \text{return } (\{t_i \mapsto v_i\}, \Delta_{v_i})\} \Rightarrow \Theta_2; (v s', \Delta_{v s}) \quad \Theta_2; \delta_{a_{err}}; \Delta_{v s} \vdash a_{err} : \forall i \in \{1, \dots, k\}. \text{do } \{b_i \leftarrow \text{valid}(v s'(t_i)); \text{return } (\bigwedge b_i)\} \Rightarrow \Theta'; (a'_{err}, \Delta_{a_{err}})}{\Theta; \varepsilon; \Delta_\varepsilon; r; \{s \mid x \in e\} \vdash \text{load}_\Delta F (a_{err}, v s) \delta_F (\delta_{a_{err}} \otimes \delta_{v s}) \Rightarrow \Theta'; ((a'_{err}, v s'), \Delta_{a_{err}})}$$

$$\frac{t \in \text{dom}(v s) \quad \Theta; \varepsilon[x \mapsto t]; \Delta_\varepsilon[x \mapsto \emptyset]; r; s \vdash \text{load}_\Delta F v s(t) \delta_F \delta_{v s}(t) \Rightarrow \Theta'; (v', \Delta_v)}{\Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash_x \text{load}_\Delta F (t, v s) \delta_F \delta_{v s} \Rightarrow \Theta'; (v', \Delta_v)}$$

$$\frac{t \notin \text{dom}(v s) \quad \Theta; \varepsilon; r; s \vdash \text{load} (\delta_F F) \Rightarrow \Theta'; v'}{\Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash_x \text{load}_\Delta F (t, v s) \delta_F \delta_{v s} \Rightarrow \Theta'; (v', \Delta)}$$

$$\Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash \text{store}_\Delta F v \delta_F \delta_v \Rightarrow \Theta'; (F', \phi')$$

$$\frac{\Delta_\varepsilon|_{fv(s)} = \emptyset \quad \delta_F \searrow_F r = \emptyset \quad \Theta; \varepsilon; r; s \vdash \text{sense } v \Rightarrow r s \quad \phi = \lambda F'. F = F' \quad \text{rs}}{\Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash \text{store}_\Delta F v \delta_F \emptyset \Rightarrow \Theta; (F, \phi)}$$

$$\frac{\Theta; \varepsilon; r; s \vdash \text{store} (\delta_F F) v \Rightarrow \Theta'; (F', \phi')}{\Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash \text{store}_\Delta F v \delta_F \delta_v \Rightarrow \Theta'; (F', \phi')}$$

$$s = M s_1$$

$$\frac{\Theta(a) = e \quad \Theta; e \Rightarrow \Theta'; (a_{err}, v) \quad \Theta'; \varepsilon; \Delta_\varepsilon; r; s \vdash \mathbf{store}_\Delta F v \delta_F \delta_v \Rightarrow \Theta''; (F', \phi')}{\Theta; \varepsilon; \Delta_\varepsilon; r; M s \vdash \mathbf{store}_\Delta F a \delta_F (M_{\delta_a} (\delta_{a_{err}} \otimes \delta_v)) \Rightarrow \Theta''; (F', \phi')}$$

$$s = e :: s_1$$

$$\frac{\Delta_\varepsilon|_{fv(s)} = \emptyset \quad \Theta; \llbracket r / e \rrbracket_{Path}^\varepsilon \Rightarrow \Theta'; r' \quad \Theta'; \varepsilon; \Delta_\varepsilon; r'; e :: s \vdash \mathbf{store}_\Delta F v \delta_F \delta_v \Rightarrow \Theta''; (F', \phi')}{\Theta; \varepsilon; \Delta_\varepsilon; r; e :: s \vdash \mathbf{store}_\Delta F v \delta_F \delta_v \Rightarrow \Theta''; (F', \phi')}$$

$$s = \langle x : s_1, s_2 \rangle$$

$$\frac{\Theta; \varepsilon; \Delta_\varepsilon; r; s_1 \vdash \mathbf{store}_\Delta F v_1 \delta_F \delta_{v_1} \Rightarrow \Theta_1; (F'_1, \phi'_1) \quad \Theta_1; \varepsilon[x \mapsto v_1]; \Delta_\varepsilon[x \mapsto \delta_{v_1}]; r; s_2 \vdash \mathbf{store}_\Delta F v_2 \delta_F \delta_{v_2} \Rightarrow \Theta_2; (F'_2, \phi'_2) \quad \phi = \lambda F'. \phi'_1(F'_1) \wedge \phi'_2(F'_2)}{\Theta; \varepsilon; \Delta_\varepsilon; r; \langle x : s_1, s_2 \rangle \vdash \mathbf{store}_\Delta F (a_{err}, (v_1, v_2)) \delta_F (\delta_{a_{err}} \otimes (\delta_{v_1} \otimes \delta_{v_2})) \Rightarrow \Theta_2; ((F_1 \uparrow F_2), \phi)}$$

$$s = P e$$

$$\frac{\phi = \lambda F'. \mathbf{return} \text{ True}}{\Theta; \varepsilon; \Delta_\varepsilon; r; P e \vdash \mathbf{store}_\Delta F v \delta_F \delta_v \Rightarrow \Theta; (F, \phi)}$$

$$s = s_1?$$

$$\frac{r \notin \text{dom}(\delta_F F) \quad \phi = \lambda F'. r \notin \text{dom}(F')}{\Theta; \varepsilon; \Delta_\varepsilon; r; s? \vdash \mathbf{store}_\Delta F (a_{err}, \text{Nothing}) \delta_F (\delta_{a_{err}} \otimes \emptyset) \Rightarrow \Theta; (F, \phi)} \quad \frac{r \in \text{dom}(\delta_F F) \quad \Theta; \varepsilon; \Delta_\varepsilon; r; s \vdash \mathbf{store}_\Delta F v \delta_F \delta_v \Rightarrow \Theta'; (F_1, \phi_1) \quad \phi = \lambda F'. \phi_1(F') \wedge e \in \text{dom}(F')}{\Theta; \varepsilon; \Delta_\varepsilon; r; s? \vdash \mathbf{store}_\Delta F (a_{err}, \text{Just } v) \delta_F (\delta_{a_{err}} \otimes \delta_v?) \Rightarrow \Theta'; (F_1, \phi)}$$

$$s = \{s \mid x \in e\}$$

$$\frac{\Theta; \llbracket e \rrbracket_{\tau}^\varepsilon \Rightarrow \Theta'; ts \quad vs = \{t_1 \mapsto v_1, \dots, t_k \mapsto v_k\} \quad \phi = \lambda F'. ts = \{t_1, \dots, t_k\} \wedge \bigwedge \phi_i(F') \quad \Theta_1; \forall t_i \in \text{dom}(vs). \text{do } \{(F_i, \phi_i) \leftarrow \varepsilon[x \mapsto t_i]; \Delta_\varepsilon[x \mapsto \emptyset]; r; s \vdash \mathbf{store}_\Delta F vs(t_i) \delta_F \delta_{vs(t_i)}; \mathbf{return} (F_1 \uparrow \dots \uparrow F_k, \phi)\} \Rightarrow \Theta_2; (F', \phi')}{\Theta; \varepsilon; \Delta_\varepsilon; r; \{s \mid x \in e\} \vdash \mathbf{store}_\Delta F (a_{err}, vs) \delta_F (\delta_{a_{err}} \otimes \delta_{vs}) \Rightarrow \Theta_2; (F', \phi')}$$

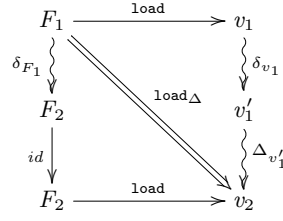
$$\Theta; \varepsilon; r; s \vdash \mathbf{sense} v \Rightarrow rs \quad \text{“Sensitivity of a forest specification in respect to a representation”}$$

$$\frac{\Theta(a) = e \quad \Theta; e \Rightarrow \Theta'; v \quad \Theta'; \varepsilon; r; s \vdash \mathbf{sense} v \Rightarrow rs}{\Theta; \varepsilon; r; M s \vdash \mathbf{sense} a \Rightarrow rs} \quad \frac{\Theta; \varepsilon; r; s \vdash \mathbf{sense} v \Rightarrow rs}{\Theta; \varepsilon; r; e :: s \vdash \mathbf{sense} v \Rightarrow \{r\} \cup rs} \quad \frac{\Theta; \varepsilon; r; s_1 \vdash \mathbf{sense} v_1 \Rightarrow rs_1 \quad \Theta; \varepsilon[x \mapsto v_1]; r; s_2 \vdash \mathbf{sense} v_2 \Rightarrow rs_2}{\Theta; \varepsilon; r; \langle x : s_1, s_2 \rangle \vdash \mathbf{sense} (a_{err}, (v_1, v_2)) \Rightarrow rs_1 \cup rs_2} \quad \frac{}{\Theta; \varepsilon; r; P e \vdash \mathbf{sense} v \Rightarrow \{\}} \quad \frac{\Theta; \varepsilon; r; s? \vdash \mathbf{sense} (a_{err}, \text{Nothing}) \Rightarrow \{r\} \quad \Theta; \varepsilon; r; s \vdash \mathbf{sense} v \Rightarrow rs}{\Theta; \varepsilon; r; s? \vdash \mathbf{sense} (a_{err}, \text{Just } v) \Rightarrow \{r\} \cup rs} \quad \frac{vs = \{t_1 \mapsto v_1, \dots, t_k \mapsto v_k\} \quad \forall i \in \{1, \dots, k\}. \Theta; \varepsilon[x \mapsto t_i]; r; s \vdash \mathbf{sense} v_i \Rightarrow r_i}{\Theta; \varepsilon; r; \{s \mid x \in e\} \vdash \mathbf{sense} (a_{err}, vs) \Rightarrow \bigcup r_i}$$

Theorem B.1 (Incremental Load Soundness). *If*

$$\begin{aligned} \Theta; \varepsilon; r; s \vdash \mathbf{load} F_1 &\Rightarrow \Theta_1; v_1 \\ \Theta_1; v_1 &\xrightarrow{\delta_{v_1}} \Theta_2; v'_1 \\ \Theta_2; \varepsilon'; \Delta_\varepsilon; r; s \vdash \mathbf{load}_\Delta F_1 v'_1 \delta_{F_1} \delta_{v_1} &\Rightarrow \Theta_3; (v_2, \Delta_{v'_1}) \\ \Theta_1; \varepsilon'; r; s \vdash \mathbf{load} (\delta_{F_1} F_1) &\Rightarrow \Theta_4; v_3 \end{aligned}$$

then $v_2 \sim_{\Theta_3}^{err} v_3$ and $\text{valid}(v_2) \sim_{\Theta_3}^{err} \text{valid}(v_3)$.



Lemma 1 (Incremental Load Stability). $\Theta; \varepsilon; \Delta_\varepsilon; r; \mathbf{M} s \vdash \text{load}_\Delta F' a \delta_F (\mathbf{M}_\emptyset \delta_v) \Rightarrow \Theta'; (a, \Delta_a)$

Theorem B.2 (Incremental Store Soundness). *If*

$$\Theta; \varepsilon; r; s \vdash \text{store } F v_1 \Rightarrow \Theta_1; (F_1, \phi_1)$$

$$\Theta_1; v_1 \xrightarrow{\delta_{v_1}} \Theta_2; v_2$$

$$\Theta_2; \varepsilon'; \Delta_\varepsilon; r; s \vdash \text{store}_\Delta F_1 v_2 \delta_{F_1} \delta_{v_1} \Rightarrow \Theta_3; (F_2, \phi_2)$$

$$\Theta_2; \varepsilon'; r; s \vdash \text{store } (\delta_{F_1} F_1) v_2 \Rightarrow \Theta_4; (F_3, \phi_3)$$

then $F_2 = F_3$ and $\phi_2(F_2) = \phi_3(F_3)$.

