Incremental Forest Draft

Yiming Wu

August 24, 2014

1 Model of Incremental Forest

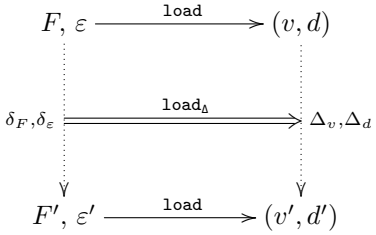
Incremental Forest aims to improve efficiency of Forest. It enables Forest to respond to changes on file systems and environment. The Incremental Forest detects changes and apply them only to the corresponding metadata and representation instead of reload the whole file system again.

In static Forest, we only have `load` function that loads certain file system F under path r into memory as representation v and metadata d. The original Forest model can be abstracted like:

$$F, \varepsilon \xrightarrow{load} (v, d)$$

This model fails to meet the requirement of real world application for that it has to reload the whole file system again if either file system or environment changes. This makes implementation cumbersome.

To improve efficiency of Forest, we introduce a delta-load function $\texttt{load}_\Delta$ that takes in changes(delta) on file system and enviroment, δ_F and δ_ε respectively. Then $\texttt{load}_\Delta$ generates a squence of changes $\Delta_v = \delta_{v1} \cdot \delta_{v2} \ldots \delta_{vn}$ and $\Delta_d = \delta_{d1} \cdot \delta_{d2} \ldots \delta_{dn}$ that will be use to update representation and metadata in memory.



$$\Delta_v = \delta_{v1} \cdot \delta_{v2} \ldots \delta_{vn} \quad \Delta_d = \delta_{d1} \cdot \delta_{d2} \ldots \delta_{dn}$$

The square figure above shows a abstracted model of Incremental Forest. Some parameters like specification s are hidden for simplicity. The (v', d') is the result of loading the new file system again. In Incremental Forest, we guarantee that we will get same in-memory representation if we apply the sequence of changes Δ_v and Δ_d to original (v, d).

In the following section,we will first give out syntax and semantic of Incremental Forest. Then we will propose and prove properties and theorum of correctness of Incremental Forest.

2 Syntax definition

In this section, we will first discuss definition of Incremental Forest syntax. Then we will introduce incremental semantic under different specifications.

Static Forest definitions

Specification s	$::= k_{\pi_2}^{\pi_1} \mid e :: s \mid \langle x : s_1, s_2 \rangle \mid \{s \mid x \in e\} \mid s = P(e) \mid s?$
Type t	$::= \texttt{String} \mid t_1 \times t_2 \mid t\ \texttt{Map} \mid \texttt{Maybe}\ t \mid () \mid \texttt{Bool}$
Environment ε	$::= \emptyset \mid \varepsilon, x \mapsto v$

incremental Forest definitions

Change δ	$::= \rightsquigarrow v \mid \delta_1 \cdot \delta_2 \mid \pi_1\,\delta_1 \mid \pi_2\,\delta_2 \mid \delta_1? \mid f \leftarrow \delta_1$ $\mid \texttt{add}(n, v) \mid \texttt{del}(n) \mid \texttt{mod}(n, \delta_1) \mid \emptyset$
Update on files δ_F	$::= \texttt{addFile}(r, \omega) \mid \texttt{rmvFile}(r) \mid \texttt{chgAttri}(r, a') \mid \delta_1 \cdot \delta_2 \mid \emptyset$
Update on environments δ_ε	$::= \emptyset \mid \delta_\varepsilon, \delta_x \mapsto \delta_v$

Static Forest definition can be found in previous Forest paper. In incremental syntax, we define changes on in-memory data δ, updates on file system δ_F and updates on environment δ_ε.

In syntax of changes δ, $\rightsquigarrow v$ means overwriting the value with v; $\delta_1 \cdot \delta_2$ means combination of changes; $\pi_1\,\delta_1$ means apply the change δ_1 to the first element of the pair, $\pi_2\,\delta_1$ means apply the change δ_1 to the second element of the pair; $\delta_1?$ means apply the change and find out either this file system that is undefined at the current path, or a file system containing the current path and satisfying the specification; the $\delta_1 \otimes \delta_2$ equals to $\pi_1\,\delta_1 \cdot \pi_2\,\delta_2$, which can be applied to change two elements of a pair separately; The other three $\texttt{add}(n, v) \mid \texttt{del}(n, v) \mid \texttt{mod}(n, \delta_1)$ is for applying changes for elements in the map from the comprehension specification; $\emptyset$ is for empty sequence, which means doing nothing. Besides these syntax, we define syntax sugar $\otimes$:

$$\delta_1 \otimes \delta_2 = \pi_1\,\delta_1 \cdot \pi_2\,\delta_2$$

We can also define some algebraric operation on changes δ.

$$\delta \cdot \rightsquigarrow v \Rightarrow \rightsquigarrow v$$
$$\delta \cdot \emptyset \Rightarrow \delta$$
$$\emptyset \cdot \delta \Rightarrow \delta$$

Updates on file system is abstracted to adding a file with content ω to path r as $\texttt{addFile}(r, \omega)$; deleting a file from path r as $\texttt{rmvFile}(r)$; changing attributes of a file as $\texttt{chgAttri}(r, a')$; combinations of updates as $\delta_1 \cdot \delta_2$ and finally, doing nothing as $\emptyset$.

For Forest environment, $\varepsilon : \mathit{Vars} \mapsto \mathit{Vals}$, it maps variables to values. Environment update $\delta_\varepsilon : \epsilon \mapsto \epsilon$ maps from environment to new environment. The mapping rule for δ_ε is:

$$\delta_\varepsilon(\varepsilon') = \varepsilon''$$

$$\mathit{where}\ \forall x \in \texttt{dom}(\varepsilon''), x \in \texttt{dom}(\varepsilon') \quad \delta_x(\varepsilon'(x)) = \varepsilon''(x)$$

In Increamental Forest, There is always a variable $\cdot$ that denotes the current path r. We will use r to represent current path as a syntax sugar in this paper.

$$r = \varepsilon(\cdot)$$

After we define all the syntax of Incremental Forest, we can move on to the semantic part.

3 Semantic of Incremental Forest

We will discuss semantic of Incremental Forest under different specification in this section.

$\boxed{s ::= k^{\pi_1}_{\pi_2}}$

$$\frac{F(r) = (a, \texttt{File}(\omega))}{\varepsilon, r, k \vdash \texttt{load}\ F \rhd (\omega, (True, a))}$$

$$\varepsilon, \delta_\varepsilon, r, k \vdash \texttt{load}_\Delta(F, v, d) :$$

$$\begin{array}{rrr}
\texttt{addFile}(r', \omega')\rhd & (\rightsquigarrow \omega', \rightsquigarrow True) & if\ r = r' \\
\texttt{addFile}(r', \omega')\rhd & (\emptyset, \emptyset) & otherwise \\
\texttt{rmvFile}(r')\rhd & (\rightsquigarrow \text{“”}, \rightsquigarrow False) & if\ r = r' \\
\texttt{rmvFile}(r')\rhd & (\emptyset, \emptyset) & otherwise
\end{array}$$

When an update occurs in the file system, Incremental Forest will check whether the update happens at the path it is focusing on. If the updates happens on the current path r, then Incremental Forest will overwrite the representation and metadata. Otherwise, it will do nothing.

$\boxed{Combination\ of\ Updates}$

$$\frac{\begin{array}{c}\varepsilon, \delta_\varepsilon, r, s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_{F_1} \rhd (\delta_{v_1}, \delta_{d_1}) \\ \varepsilon, \delta_\varepsilon, r, s \vdash \texttt{load}_\Delta(\delta_{F_1} F,\ \delta_{v_1} v,\ \delta_{d_1} d)\ \delta_{F_2} \rhd (\delta_{v_2}, \delta_{d_2})\end{array}}{\delta_\varepsilon, s \vdash \texttt{load}_\Delta(F, v, d)\ (\delta_{F_1} \cdot \delta_{F_2}) \rhd ((\delta_{v_2} \cdot \delta_{v_1}), (\delta_{d_2} \cdot \delta_{d_1}))}$$

In Incremental Forest, if an update can be splited into separate updates, the result stays the same between loading separate updates step by step or load them as one update.

$\boxed{e :: s}$

$$\frac{r' = [\![r/e]\!]^\varepsilon_{Path} \quad \varepsilon, r', s \vdash \texttt{load}\ F \rhd (v, d)}{\varepsilon, r, e :: s \vdash \texttt{load}\ F \rhd (v, d)}$$

$\boxed{Incremental\ \ e :: s}$

$$\frac{\begin{array}{c}[\![r/e]\!]^\varepsilon_{Path} = [\![r/e]\!]^{\delta_\varepsilon(\varepsilon)}_{Path} \\ \varepsilon, \delta_\varepsilon, [\![r/e]\!]^{\delta_\varepsilon(\varepsilon)}_{Path}, s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)\end{array}}{\varepsilon, \delta_\varepsilon, r, e :: s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)}$$

$$\frac{\begin{array}{c}[\![r/e]\!]^\varepsilon_{Path} \neq [\![r/e]\!]^{\delta_\varepsilon(\varepsilon)}_{Path} \\ \delta_\varepsilon(\varepsilon), [\![r/e]\!]^{\delta_\varepsilon(\varepsilon)}_{Path}, s \vdash \texttt{load}(\delta_F F)\ u \rhd (v, d)\end{array}}{\varepsilon, \delta_\varepsilon, r, e :: s \vdash \texttt{load}_\Delta(F, v, d)\ u \rhd (\rightsquigarrow v, \rightsquigarrow d)}$$

In Incremental Forest, path specification $e :: s$ will create a new path but may not result in the same path in old environment and new environment. When the r/e evaluates to the same path in both new and old environment, we will recursively use delta load function. Once r/e differs between the new and old environment, then we will directly load the data, because we are now definitely loading a new file.

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

$$\frac{\begin{array}{c}\varepsilon, r, s_1 \vdash \texttt{load}\ F \rhd (v_1, d_1) \\ (\varepsilon, x \mapsto v_1, x_{md} \mapsto d_1), r, s_2 \vdash \texttt{load}\ F \rhd (v_2, d_2) \\ b = \texttt{valid}(d_1) \wedge \texttt{valid}(d_2)\end{array}}{\varepsilon, r, \langle x : s_1, s_2 \rangle \vdash \texttt{load}\ F \rhd ((v_1, v_2), b)}$$

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

$$\frac{\begin{array}{c}\varepsilon, r, \langle x : s_1, s_2 \rangle \vdash \texttt{load}\ F \rhd ((v_1, v_2), b) \\ \varepsilon, \delta_\varepsilon, r, s_1 \vdash \texttt{load}_\Delta(F, v_1, d_1)\ \delta_F \rhd (\delta_{v_1}, \delta_{d_1}) \\ (\varepsilon, x \mapsto v_1, x_{md} \mapsto d_1), (\delta_\varepsilon, \delta_x \mapsto \delta_{v_1}, x_{md} \mapsto \delta_{d_1}), r, s_2 \vdash \texttt{load}_\Delta(F, v_2, d_2)\ \delta_F \rhd (\delta_{v_2}, \delta_{d_2})\end{array}}{\delta_\varepsilon, \langle x : s_1, s_2 \rangle \vdash \texttt{load}\ (F, (v_1, v_2), b)\ u \rhd (\delta_{v_1} \otimes \delta_{v_2}, \delta_{d_1} \otimes \delta_{d_2})}$$

For dependant pair, Increamental Forest will delta load both the representation and metadata, and apply the changes to the dependant pair.

$\boxed{s = \mathtt{P}(e)}$

$$\frac{b = [\![e]\!]^{\varepsilon}_{Bool}}{\varepsilon, r, \mathtt{P}(e) \vdash \mathtt{load}\ F \rhd ((), (b, ()))}$$

$\boxed{Incremental \quad s = P(e)}$

$$\frac{\delta_{\varepsilon}(\varepsilon) \mid_{fv(e)} = \emptyset \vee [\![e]\!]^{\varepsilon}_{Bool} = [\![e]\!]^{\delta_{\varepsilon}(\varepsilon)\varepsilon}_{Bool}}{\varepsilon, \delta_{\varepsilon}, r, P(e) \vdash \mathtt{load}\ (F, v, d)\ \delta_F \rhd (\emptyset, \emptyset)}$$

$$\frac{[\![e]\!]^{\varepsilon}_{Bool} \neq [\![e]\!]^{\delta_{\varepsilon}(\varepsilon)\varepsilon}_{Bool}}{\delta_{\varepsilon}, r, P(e) \vdash \mathtt{load}\ (F, v, d)\ \delta_F \rhd (\emptyset, \pi_1(\rightsquigarrow b))}$$

If free variables in expression e hasn't changed or the expression e evaluates to the same value in new and old environment, Incremental Forest will do nothing. Otherwise, Incremental Forest will test the expression under new environment.

$\boxed{s = s_1?}$

$$\frac{r \notin \mathtt{dom}(F)}{\varepsilon, r, s_1? \vdash \mathtt{load}\ F \rhd (Nothing, True)}$$

$$\frac{r \in \mathtt{dom}(F) \quad \varepsilon, r, s_1 \vdash \mathtt{load}\ F \rhd (v_1, d_1)}{\varepsilon, r, s_1? \vdash \mathtt{load}\ F \rhd (Just\ v_1, Just\ d_1)}$$

$\boxed{Incremental \quad s = s_?}$

$$\varepsilon, r, s_1 \vdash \mathtt{load}\ F \rhd (v, d)$$

$$\frac{r \notin \mathtt{dom}(\delta_F F) \quad v = Nothing}{\varepsilon, \delta_{\varepsilon}, r, s_1? \vdash \mathtt{load}_{\Delta}(F, v, d)\ \delta_F \rhd (\emptyset, \emptyset)}$$

$$\frac{r \notin \mathtt{dom}(\delta_F F) \quad v = Just\ v' \quad \varepsilon, r, s_1 \vdash \mathtt{load}\ F \rhd (v_1, d_1)}{\varepsilon, \delta_{\varepsilon}, r, s_1? \vdash \mathtt{load}_{\Delta}(F, v, d)\ \delta_F \rhd (\rightsquigarrow v_1, \rightsquigarrow d_1)}$$

$$\frac{r \in \mathtt{dom}(\delta_F F) \quad v = Nothing \quad \varepsilon, r, s_1 \vdash \mathtt{load}\ F \rhd (v_1, d_1)}{\varepsilon, \delta_{\varepsilon}, r, s_1? \vdash \mathtt{load}_{\Delta}(F, v, d)\ \delta_F \rhd (\rightsquigarrow v_1, \rightsquigarrow d_1)}$$

$$\frac{r \in \mathtt{dom}(\delta_F F) \quad v = Just\ v' \quad \varepsilon, \delta_{\varepsilon}, r, s_1 \vdash \mathtt{load}_{\Delta}(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)}{\varepsilon, \delta_{\varepsilon}, r, s_1? \vdash \mathtt{load}_{\Delta}(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)}$$

Incremental Forest will check whether the current path r is defined under the new file system $\delta_F\ F$ and perform the same operation as static Forest does.

$\boxed{s = [s \mid x \in e]}$

$$\frac{\begin{array}{c}[\![e]\!]^{\varepsilon}_{\{\tau\}} = \{t_1, t_2, \ldots, t_k\} \\ \forall i \in \{1, 2, \ldots, k\}, (\varepsilon, x \mapsto t_i), r, s \vdash \mathtt{load}\ F \rhd (v_i, d_i) \\ vs\ \mathtt{maps}\ t_i \to v_i,\ ds\ \mathtt{maps}\ t_i \to d_i\end{array}}{\varepsilon, [s \mid x \in e] \vdash \mathtt{load}\ F \rhd (vs, ds)}$$

To make Forest incremental, we change the data structure of vs and ds as `map` so that we can easily find representation v_i and metadata d_i of certain item t.

$$vs(t) = v$$
$$ds(t) = d$$

All these changes are made to preserve the atomic property of Incremental Forest so that we can get all the information we need in one operation.

$\boxed{\textit{Incremental} \quad s = [s \mid x \in e]}$

$$\frac{\begin{array}{c} T = [\![e]\!]^{\varepsilon}_{\{\tau\}} \quad T' = [\![e]\!]^{\delta_\varepsilon(\varepsilon)}_{\{\tau\}} \\ Dv = \Pi\ \delta^v_i, Dd = \Pi\ \delta^d_i \quad \textit{where } \forall t_i \in T \setminus T', \quad \delta^v_i = \texttt{del}(t_i), \delta^d_i = \texttt{del}(t_i) \\ Av = \Pi\ \delta^v_j, Ad = \Pi\ \delta^d_j \\ \textit{where } \forall t_j \in T' \setminus T, \quad (\varepsilon, x \mapsto t_j), r, s \vdash \texttt{load}(\delta_F F) \rhd (v_j, d_j) \quad \delta^v_j = \texttt{add}(t_j, v_j), \delta^d_j = \texttt{add}(t_j, d_j) \\ Mv = \Pi\ \delta^v_k, Ad = \Pi\ \delta^d_k \\ \textit{where } \forall t_k \in T' \cap T, \quad (\varepsilon, x \mapsto t_i), \delta_\varepsilon, r, s \vdash \texttt{load}_\Delta(F, vs(t_i), ds(t_i))\delta_F \rhd (\delta_v, \delta_d) \quad \delta^v_k = \texttt{mod}(t_i, \delta_v), \delta^d_k = \texttt{mod}(t_i, \delta_d) \end{array}}{\varepsilon, \delta_\varepsilon, r, [s \mid x \in e] \vdash \texttt{load}_\Delta\ (F, vs, ds)\ \delta_F \rhd (Dv \cdot Av \cdot Mv,\ Dd \cdot Ad \cdot Md)}$$

Changes on the evaluation of expression e is arbitrary, thus we need to categorize these changes to three types.

1. If the item t_i doesn't exist in new environment, Incremental Forest delete corresponding representation and metadata from the map.
2. If the item t_j doesn't exist in old environment but appears in the new environment, Incremental Forest add corresponding representation and metadata to the map.
3. If the item t_k is maintained in both environments, Incremental Forest delta load the file recursively.

These changes can be viewed as three different sets where Dv, Av and Mv are changes on representation map vs, and Dd, Ad and Md are changes on metadata map ds. We combine them as a sequence of changes.

4 Proof

In this section, we will discuss about assumptions and theorems of Incremental Forest.

Assumption 4.1 (Update Sequence)**.** *Incremental Forest takes in a sequence of updates consists of*

$$\texttt{addFile}(r, \omega) \mid \texttt{rmvFile}(r) \mid \texttt{chgAttri}(r, a')$$

Or empty sequence $\emptyset$*.*

Complex updates can be constructed from atomic updates.

Assumption 4.2 (Filesystem is a tree)**.** *Incremental Forest views symbolic link as a normal file. not necessarily needed.*

Theorem 4.1 (Soundness)**.** *Incremental Forest satisfied the rule that*

$$\begin{aligned}
&\varepsilon, s \vdash \texttt{load}\ F\ \rhd (v, d) \\
&\delta_\varepsilon, s \vdash \texttt{load}\ (F, v, d)\ \delta_F \rhd (\delta_v, \delta_d) \\
&\varepsilon_{new}, s \vdash \texttt{load}\ (\delta_F\ F)\ \rhd (v', d')
\end{aligned}$$

$$v' = \delta_v v, \quad d' = \delta_d d$$

Proof. By induction
Case: $s = k$

1. By definition, if $\varepsilon_{old}(r) \neq \varepsilon_{new}(r)$,

$$\begin{aligned}
&\varepsilon, s \vdash \texttt{load}\ F\ \rhd (v, d) \\
&\delta_\varepsilon, s \vdash \texttt{load}\ (F, v, d)\ \delta_F \rhd (\rightsquigarrow \omega', \rightsquigarrow (True, a)) \\
&\varepsilon_{new}, s \vdash \texttt{load}\ (\delta_F\ F)\ \rhd (\omega', (True, a))
\end{aligned}$$

$$\rightsquigarrow \omega'\ v = \omega' \quad \rightsquigarrow (True, a)\ d = (True, a)$$

   The Soundness Theorem holds for the $\varepsilon_{old}(r) \neq \varepsilon_{new}(r)$ case here.

2. By definition, if $\delta_F = \texttt{addFile}(r', \omega')$, here we use r to represent $\varepsilon_{old}(r)$
   Case: $r = r'$

$$\begin{aligned}
&\varepsilon, s \vdash \texttt{load}\ F\ \rhd (v, d) \\
&\delta_\varepsilon, s \vdash \texttt{load}\ (F, v, d)\ \delta_F \rhd (\rightsquigarrow \omega', \rightsquigarrow (True, a)) \\
&\varepsilon_{new}, s \vdash \texttt{load}\ (\delta_F\ F)\ \rhd (\omega', (True, a))
\end{aligned}$$

$$\rightsquigarrow \omega'\ v = \omega' \quad \rightsquigarrow (True, a)\ d = (True, a)$$

   The Soundness Theorem holds for the $r = r'$ case here.
   Case: $r \neq r'$

$$\begin{aligned}
&\varepsilon, s \vdash \texttt{load}\ F\ \rhd (v, d) \\
&\delta_\varepsilon, s \vdash \texttt{load}\ (F, v, d)\ \delta_F \rhd (\emptyset, \emptyset) \\
&\varepsilon_{new}, s \vdash \texttt{load}\ (\delta_F\ F)\ \rhd (v, d)
\end{aligned}$$

$$\emptyset\ v = v \quad \emptyset\ d = d$$

   The Soundness Theorem holds for the $r \neq r'$ case here.
   With same method, we can easily prove that Soundness holds for $\texttt{rmvFile}(r')$, $\texttt{addAttri}(r', a')$ and $\emptyset$ scenarios.
   Thus we can conclude that Soundness holds for the constant specification case.

Case: $e :: s$

1. When the generated path $[\![r/e]\!]^{\varepsilon_{new}}_{Path}$ keeps the same in both old and new environment, by induction, we assume that Soundness holds for $(\delta_\varepsilon \setminus r, r \mapsto ([\![r/e]\!]^{\varepsilon_{new}}_{Path}, \emptyset)), s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)$, which means.

$$(\varepsilon, r \mapsto [\![r/e]\!]^{\varepsilon_{new}}_{Path}), s \vdash \texttt{load}\ F\ \rhd (v, d)$$
$$(\delta_\varepsilon \setminus r, r \mapsto ([\![r/e]\!]^{\varepsilon_{new}}_{Path}, \emptyset)), s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)$$
$$(\varepsilon, r \mapsto [\![r/e]\!]^{\varepsilon_{new}}_{Path}), s \vdash \texttt{load}\ F\ \rhd (\delta_v\ v, \delta_d\ d)$$

   By definition of $e :: s$ static load and incremental load, we will have

$$\varepsilon, e :: s \vdash \texttt{load}\ F\ \rhd (v, d)$$
$$\delta_\varepsilon, e :: s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)$$
$$\varepsilon, e :: s \vdash \texttt{load}\ F\ \rhd (\delta_v\ v, \delta_d\ d)$$

2. When the generated path changes, by induction, we assume that Soundness holds for $(\delta_\varepsilon \setminus r, r \mapsto (r, [\![r/e]\!]^{\varepsilon_{new}}_{Path})), s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)$, which means.

$$(\varepsilon, r \mapsto [\![r/e]\!]^{\varepsilon_{new}}_{Path}), s \vdash \texttt{load}\ F\ \rhd (v, d)$$
$$(\delta_\varepsilon \setminus r, r \mapsto ([\![r/e]\!]^{\varepsilon_{new}}_{Path}, \emptyset)), s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)$$
$$(\varepsilon, r \mapsto [\![r/e]\!]^{\varepsilon_{new}}_{Path}), s \vdash \texttt{load}\ F\ \rhd (\delta_v\ v, \delta_d\ d)$$

   By definition of $e :: s$ static load and incremental load, we will have

$$\varepsilon, e :: s \vdash \texttt{load}\ F\ \rhd (v, d)$$
$$\delta_\varepsilon, e :: s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)$$
$$\varepsilon, e :: s \vdash \texttt{load}\ F\ \rhd (\delta_v\ v, \delta_d\ d)$$

   Thus we can conclude that Soundness holds for the focus specification case.

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

1. By definition

$$\varepsilon, \langle x : s_1, s_2 \rangle \vdash \texttt{load}\ F\ \rhd ((v_1, v_2), (b, (d_1, d_2)))$$
$$\delta_\varepsilon, \langle x : s_1, s_2 \vdash \texttt{load}\ (F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)$$
$$\varepsilon_{new}, \langle x : s_1, s_2 \rangle \vdash \texttt{load}\ (\delta_F\ F)\ \rhd ((v_1', v_2'), (b', (d_1', d_2')))$$

   By induction, Soundness holds for s_1.

$$\delta_\varepsilon, s_1 \vdash \texttt{load}_\Delta(F, v_1, d_1)\ u \rhd (\delta_{v_1}, \delta_{d_1})$$
$$\delta_{v_1}\ v_1 = v_1' \quad \delta_{d_1}\ d_1 = d_1'$$

   Because the delta environment $(\delta_\varepsilon, x \mapsto (v_1, \delta_{v_1}), x_{md} \mapsto (d_1, \delta_{d_1}))$ will keep the record of update of v_1 and d_1, Soundness also holds for s_2 so that

$$(\delta_\varepsilon, x \mapsto (v_1, \delta_{v_1}), x_{md} \mapsto (d_1, \delta_{d_1})), s_2 \vdash \texttt{load}_\Delta(F, v_2, d_2)\ u \rhd (\delta_{v_2}, \delta_{d_2})$$
$$\delta_{v_2}\ v_2 = v_2' \quad \delta_{d_2}\ d_2 = d_2'$$

   To apply changes on data, we use $\otimes$ to apply two changes. The result should be the same with (v_1', v_2') and (d_1', d_2').

$$\delta_{v_1} \otimes \delta_{v_2}\ (v_1, v_2) = (v_1', v_2') \quad \delta_{v_1} \otimes \delta_{d_2}\ (d_1, d_2) = (d_1', d_2')$$

   Finally $valid_{12} \leftarrow$ will validate the two metadata and change the original one. The result of it goes the same with the b'. Thus we can conclude that Soundness holds for the dependant pair specification case.

Case: $s = P(e)$

1. By definition

$$\varepsilon, P(e) \vdash \texttt{load}\ F\ \rhd ((), (b, ()))$$
$$\delta_\varepsilon, P(e) \vdash \texttt{load}\ (F, v, d)\ \delta_F \rhd (\delta_v, \pi_1(\rightsquigarrow b'))$$
$$\varepsilon_{new}, P(e) \vdash \texttt{load}\ (\delta_F\ F)\ \rhd ((), (b'', ()))$$

   In the equations,

$$b' = [\![e]\!]^{\varepsilon_{new}}_{Bool} \quad b'' = [\![e]\!]^{\varepsilon_{new}}_{Bool}$$

   The b' and b'' holds the same value. Thus we can conclude that Soundness holds for the predicate specification case.

Case: $s = s_1?$

1. When $r \notin \texttt{dom}(F) \quad r' \notin \texttt{dom}(\delta_F\ F)$

$$\varepsilon, s_1? \vdash \texttt{load}\ F\ \triangleright (Nothing, (True, Nothing))$$
$$\varepsilon_{new}, s_1? \vdash \texttt{load}\ (\delta_F\ F)\ \triangleright (Nothing, (True, Nothing))$$

   The delta load function will generate $(\emptyset, \emptyset)$, which satisfied the Soundness.

2. When $r \in \texttt{dom}(F) \quad r' \notin \texttt{dom}(\delta_F\ F)$

$$\varepsilon, s_1? \vdash \texttt{load}\ F\ \triangleright (Just\ v, (b, Just\ d))$$
$$\varepsilon_{new}, s_1? \vdash \texttt{load}\ (\delta_F\ F)\ \triangleright (Nothing, (True, Nothing))$$

   The delta load function will generate $(\rightsquigarrow Nothing, \rightsquigarrow (True, Nothing))$, which satisfied the Soundness.

3. When $r \notin \texttt{dom}(F) \quad r' \in \texttt{dom}(\delta_F\ F)$

$$\varepsilon, s_1? \vdash \texttt{load}\ F\ \triangleright (Nothing, (True, Nothing))$$
$$\varepsilon_{new}, s_1? \vdash \texttt{load}\ (\delta_F\ F)\ \triangleright (Just\ v, (b, Just\ d))$$

   The delta load function will generate $(\rightsquigarrow Just\ v \rightsquigarrow (b, Just\ d))$, which satisfied the Soundness.

4. When $r \in \texttt{dom}(F) \quad r' \in \texttt{dom}(\delta_F\ F)$

$$\varepsilon, s_1? \vdash \texttt{load}\ F\ \triangleright (Just\ v, (b, Just\ d))$$
$$\varepsilon_{new}, s_1? \vdash \texttt{load}\ (\delta_F\ F)\ \triangleright (Just\ v', (b', Just\ d'))$$

   By induction, Soundness holds for

$$\delta_\varepsilon, s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \triangleright (\delta_v, \delta_d)$$
$$v' = \delta_v\ v, \quad d' = \delta_d\ (b, d)$$

   Thus we can conclude that Soundness holds for the optional specification case.

Case: $s = [s \mid x \in e]$

$$\varepsilon, s_1? \vdash \texttt{load}\ F\ \triangleright (vs, (b, ds))$$
$$\varepsilon_{new}, s_1? \vdash \texttt{load}\ (\delta_F\ F)\ \triangleright (vs', (b', ds'))$$

For comprehension, we will inspect the change of each single element of `map` vs and ds. We look into these elements by their key value: filename n. Then we will prove that the soundness of delete/add/modify each element can eventually composed together to prove the Soundness for updates on the whole map.

1. When $n \in [\![e]\!]^{\varepsilon_{old}}_{\{Path\}} \setminus [\![e]\!]^{\varepsilon_{new}}_{\{Path\}}$

$$(n, vs(n)) \in vs \wedge (n, vs'(n)) \notin vs', (n, ds) \in ds \wedge (n, ds'(n)) \notin ds'$$

   Incremental Forest will delete these elements from the vs' and ds', so Soundness holds for this case.

2. When $n \in [\![e]\!]^{\varepsilon_{new}}_{\{Path\}} \setminus [\![e]\!]^{\varepsilon_{old}}_{\{Path\}}$

$$(n, vs(n)) \notin vs \wedge (n, vs'(n)) \in vs', (n, ds) \notin ds \wedge (n, ds'(n)) \in ds'$$

   Incremental Forest will add these elements to the vs' and ds', so Soundness holds for this case.

3. When $n \in [\![e]\!]^{\varepsilon_{old}}_{\{Path\}} \cap [\![e]\!]^{\varepsilon_{new}}_{\{Path\}}$

$$(n, vs(n)) \in vs \wedge (n, vs'(n)) \in vs', (n, ds) \in ds \wedge (n, ds'(n)) \in ds'$$

   Then Incremental Forest will delta load corresponding v and d in the way that satisfies Soundness.

$$(\delta_\varepsilon, r \mapsto (r, \rightsquigarrow r/n)), s \vdash \texttt{load}_\Delta(F, v, d)\ u \triangleright (\delta_v, \delta_d)$$

   So that

$$\delta_v\ vs(n) = vs'(n), \quad \delta_d\ ds(n) = ds'(n)$$

   Thus, the modification using δ_v and δ_d holds the Soundness for this single element.

4. For these filename categories, we can define them as

$$\begin{aligned} W_{del} &= [\![e]\!]^{\varepsilon_{old}}_{\{Path\}} \setminus [\![e]\!]^{\varepsilon_{new}}_{\{Path\}} \\ W_{add} &= [\![e]\!]^{\varepsilon_{new}}_{\{Path\}} \setminus [\![e]\!]^{\varepsilon_{old}}_{\{Path\}} \\ W_{keep} &= [\![e]\!]^{\varepsilon_{old}}_{\{Path\}} \cap [\![e]\!]^{\varepsilon_{new}}_{\{Path\}} \\ W_{all} &= [\![e]\!]^{\varepsilon_{old}}_{\{Path\}} \cup [\![e]\!]^{\varepsilon_{new}}_{\{Path\}} \end{aligned}$$

   These categories hold the relation that

$$\begin{aligned} W_{del} \cap W_{add} \cap W_{keep} &= \emptyset \\ W_{del} \cap W_{add} \cap W_{keep} &= W_{all} \end{aligned}$$

   This means that each filename and its data should be updated once and only once correctly. The updates on them won't effect each other, and make every element of the `map` deleted or added or become the new data it should be. Thus we can conclude that Soundness holds for the Comprehension specification case.

Conclusion
By induction, we can conclude that Incremental Forest satisfies Soundness. □

Definition 4.1 ($\emptyset$ Operation)**.** We define some $\emptyset$ operations that will produce exactly $\emptyset$ change

$$\begin{aligned}
\emptyset \cdot \emptyset &= \emptyset && (1)\\
\pi_1 \emptyset &= \emptyset && (2)\\
\pi_2 \emptyset &= \emptyset && (3)\\
\emptyset \otimes \emptyset &= \emptyset && (4)\\
\texttt{mod}(n, \emptyset) &= \emptyset && (5)\\
valid_{12} \leftarrow \emptyset &= \emptyset && (6)\\
valid_{all} \leftarrow \emptyset &= \emptyset && (7)
\end{aligned}$$

The last two definitions are not graceful as expected. Actually they are unnecessary in real semantic, but we enforce them here to prove Incrementality.

Theorem 4.2 (Incrementality)**.** *If there is no update in file system nor environment, Incremental Forest will return $\emptyset$ update.*

$$\frac{\Delta_\emptyset = \{\delta_\varepsilon \mid \delta_\varepsilon \vdash \varepsilon_{old} = \varepsilon_{new}\} \quad \delta_\emptyset \in \Delta_\emptyset}{\delta_\emptyset, r, s \vdash \texttt{load}_\Delta\ (F, v, d)\ \emptyset \rhd (\emptyset, \emptyset)}$$

Proof. By induction

Case: $s = k$
By definition, if $\delta_\emptyset \vdash \varepsilon_{old} = \varepsilon_{new}$, Incremental Forest will use the last rule in const specification

$$\delta_\emptyset, r, k \vdash \texttt{load}_\Delta\ (F, v, d)\ \emptyset \rhd (\emptyset, \emptyset)$$

Incrementality holds for this case.

Case: $s = e :: s_1$

By definition, if nothing have changed in the environment, we only consider the $[\![r/e]\!]^{\varepsilon_{old}}_{Path} = [\![r/e]\!]^{\varepsilon_{new}}_{Path}$ situation

$$\frac{(\delta_\emptyset \setminus r, r \mapsto ([\![r/e]\!]^{\varepsilon_{new}}_{Path}, \emptyset)), s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \emptyset \rhd (\delta_v, \delta_d)}{\delta_\emptyset, e :: s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \emptyset \rhd (\delta_v, \delta_d)}$$

For the new delta environment δ'_ε,

$$\frac{\delta'_\varepsilon = (\delta_\emptyset \setminus r, r \mapsto ([\![r/e]\!]^{\varepsilon_{new}}_{Path}, \emptyset)) \ \wedge\ \delta_\emptyset \vdash \varepsilon_{old} = \varepsilon_{new}}{\delta'_\varepsilon \vdash \varepsilon'_{old} = \varepsilon'_{new}}$$

This means $\delta'_\varepsilon \in \Delta_\emptyset$. By induction,

$$(\delta_\emptyset \setminus r, r \mapsto ([\![r/e]\!]^{\varepsilon_{new}}_{Path}, \emptyset)), s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \emptyset \rhd (\emptyset, \emptyset)$$

Thus,

$$\delta_\emptyset, e :: s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \emptyset \rhd (\emptyset, \emptyset)$$

Incrementality holds for this case.

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

By definition,

$$\frac{\begin{array}{c}\delta_\emptyset, s_1 \vdash \texttt{load}_\Delta(F, v_1, d_1)\ \emptyset \rhd (\delta_{v_1}, \delta_{d_1})\\ (\delta_\emptyset, x \mapsto (v_1, \delta_{v_1}), x_{md} \mapsto (d_1, \delta_{d_1})), s_2 \vdash \texttt{load}_\Delta(F, v_2, d_2)\ \emptyset \rhd (\delta_{v_2}, \delta_{d_2})\\ valid_{12}(d_1{}', d_2{}') = valid(d_1{}') \wedge valid(d_2{}')\end{array}}{\delta_\emptyset, \langle x : s_1, s_2 \rangle \vdash \texttt{load}\ (F, (v_1, v_2), (b, (d_1, d_2)))\ \emptyset \rhd (\delta_{v_1} \otimes \delta_{v_2}, valid_{12} \leftarrow (\delta_{d_1} \otimes \delta_{d_2}))}$$

By induction, we have

$$\delta_\emptyset, s_1 \vdash \texttt{load}_\Delta(F, v_1, d_1)\ \emptyset \rhd (\emptyset, \emptyset)$$

In the second line, we make $\delta_{v_1} = \emptyset$ and $\delta_{d_1} = \emptyset$.

$$\frac{\delta'_\varepsilon = (\delta_\emptyset, x \mapsto (v_1, \emptyset), x_{md} \mapsto (d_1, \emptyset))}{\delta'_\varepsilon \vdash \varepsilon'_{old} = \varepsilon'_{new}}$$

Then we have $\delta'_\varepsilon \in \Delta_\emptyset$, by induction,

$$\delta'_\varepsilon, s_2 \vdash \texttt{load}_\Delta(F, v_1, d_1)\ \emptyset \rhd (\emptyset, \emptyset)$$

By definition, we have

$$\delta_\emptyset, \langle x : s_1, s_2 \rangle \vdash \texttt{load}\ (F, (v_1, v_2), (b, (d_1, d_2)))\ \emptyset \rhd (\emptyset \otimes \emptyset, valid_{12} \leftarrow (\emptyset \otimes \emptyset))$$

Because of the $\emptyset$ operations, finally we have

$$\delta_\emptyset, \langle x : s_1, s_2 \rangle \vdash \texttt{load}\ (F, (v_1, v_2), (b, (d_1, d_2)))\ \emptyset \rhd (\emptyset, \emptyset)$$

Incrementality holds for this case.

Case: $s = P(e)$

By definition, when $\varepsilon_{old} = \varepsilon_{new}$,

$$\frac{\varepsilon_{old} = \varepsilon_{new}}{\delta_\varepsilon, P(e) \vdash \texttt{load}\ (F, v, d)\ \delta_F \rhd (\emptyset, \emptyset)}$$

Incrementality holds for this case.

Case: $s = s_1?$

By definition, since $\varepsilon_{old} = \varepsilon new$, there is only two situations intead of four. When the path doesn't exist in the file system.

$$\frac{r' = \varepsilon_{new}(r) \quad r \notin \texttt{dom}(F) \quad r' \notin \texttt{dom}(\delta_F\ F)}{\delta_\varepsilon, s? \vdash \texttt{load}_\Delta(F, v, d)\ u \rhd (\emptyset, \emptyset)}$$

When the path exists in the file system, by induction

$$\frac{r' = \varepsilon_{new}(r) \quad r \in \texttt{dom}(F) \quad r' \in \texttt{dom}(\delta_F\ F), \quad \delta_\emptyset, s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\emptyset, \emptyset)}{\delta_\varepsilon, s? \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\emptyset, \emptyset)}$$

Incrementality holds for this case.

Case: $s = [s \mid x \in e]$

When $\delta_\emptyset \in \Delta_\emptyset$, no file is added or deleted in comprehension maps vs and ds. Only *mod* case is considered here. By definition and induction,

$$\frac{\begin{array}{c}\forall n \in [\![e]\!]^{\varepsilon_{new}}_{\{Path\}} \cap [\![e]\!]^{\varepsilon_{old}}_{\{Path\}},\ v = vs(n),\ d = ds(n) \\ (\delta_\emptyset, r \mapsto (r/n, \emptyset)), s \vdash \texttt{load}_\Delta(F, v, d)\ u \rhd (\emptyset, \emptyset)\end{array}}{\texttt{changeVOf}(n) = \texttt{mod}((n, v), \pi_2\emptyset), \quad \texttt{changeDOf}(n) = \texttt{mod}((n, d), \pi_2\emptyset)}$$

By $\emptyset$ operations, changes on every single file is $\emptyset$, thus the record of changes Δ_v and Δ_d has the form:

$$\begin{array}{rcll}\Delta_v & = & \emptyset \cdot \emptyset \cdots \cdot \emptyset & \Rightarrow \quad \Delta_v = \emptyset \\ \Delta_d & = & \emptyset \cdot \emptyset \cdots \cdot \emptyset & \Rightarrow \quad \Delta_d = \emptyset\end{array}$$

Thus,

$$\frac{\delta_{id}, r, [s \mid x \in e] \vdash \texttt{load}_\Delta\ (F, vs, ds)\ \emptyset \rhd (\emptyset, valid_{all} \leftarrow \emptyset)}{\delta_{id}, r, [s \mid x \in e] \vdash \texttt{load}_\Delta\ (F, vs, ds)\ \emptyset \rhd (\emptyset, \emptyset)}$$

Incrementality holds for this case.

Conclusion
By induction, we can conclude that Incremental Forest satisfies Incrementality. □