Incremental Forest Draft

Yiming Wu

August 5, 2014

1 Model of Incremental Forest

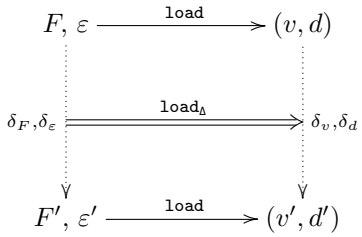
Incremental Forest aims to improve efficiency of Forest. It enables Forest to respond to changes on file systems and environment. The Incremental Forest detects changes and apply them only to the corresponding metadata and representation instead of reload the whole file system again.

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 changes δ_v and δ_d that will be use to update representation and metadata in memory.



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 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	$::= \texttt{String} \mid t_1 \times t_2 \mid t\ \texttt{set} \mid \texttt{Maybe}\ t \mid () \mid \texttt{Bool}$
Environment ε	$::= \emptyset \mid \varepsilon, x \mapsto v$

incremental Forest definitions

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

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 $\texttt{add}(v) \mid \texttt{del}(v) \mid \texttt{mod}(v, \delta_1)$ is for applying changes for elements in the set from the comprehension specification; `id` is for no changes.

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 `id`.

For Forest environment, $\varepsilon : Vars \mapsto Vals$, it maps variables to values. Environment update $\delta_\varepsilon : Vars \mapsto Vals \times \delta_v$ maps from variables to product of values and its change. Thus for any variable x, we have:

$$\begin{aligned} \varepsilon(x) &= v \\ \delta_\varepsilon(x) &= (\varepsilon(x), \delta_x) \end{aligned}$$

For example, in Adaptive Forest, environment always contain an item r which maps to the current path. We define environment before update ε_{old}, alias *old* ε; environment after update ε_{new}, alias *new* ε. Given any updates δ_ε:

$$\begin{aligned} \varepsilon_{old}(x) &= v \\ &\quad\ where\ \delta_\varepsilon(x) = (v, \delta_v) \\ \varepsilon_{new}(x) &= v' \\ &\quad\ where\ \delta_\varepsilon(x) = (v, \delta_v) \\ &\quad\ and\ v' = \delta_v v \end{aligned}$$

We also define equivalance of two environments ε_1 and ε_2

$$\varepsilon_1 = \varepsilon_2$$

$$where \quad \texttt{dom}(\varepsilon_1) = \texttt{dom}(\varepsilon_2) \ \wedge\ \forall x \in \texttt{dom}(\varepsilon_1), \quad \varepsilon_1(x) = \varepsilon_2(x)$$

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

3 Semantic of Incremental Forest

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

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

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

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

$$\begin{array}{rrl} \texttt{addFile}(r', \omega')\rhd & (\rightsquigarrow \omega', \rightsquigarrow (True, a)) & if\ \varepsilon_{new}(r) = r' \\ \rhd & (\emptyset, \emptyset) & otherwise \\ \texttt{rmvFile}(r')\rhd & (\rightsquigarrow \text{“”}, \rightsquigarrow (False, default)) & if\ \varepsilon_{new}(r) = r' \\ \rhd & (\emptyset, \emptyset) & otherwise \\ \texttt{addAttri}(r', a')\rhd & (\emptyset, \pi_2(\rightsquigarrow a')) & if\ \varepsilon_{newtheorem}(r) = r' \\ \rhd & (\emptyset, \emptyset) & otherwise \\ \texttt{id}\rhd & (\emptyset, \emptyset) & if\ \varepsilon_{new} = \varepsilon_{old} \\ \rhd & (\rightsquigarrow \omega', \rightsquigarrow (Ture, a)) & if\ \varepsilon_{new} \neq \varepsilon_{old}\ \wedge\ \varepsilon_{new}, k \vdash \texttt{load}\ F \rhd (\omega', (True, a)) \end{array}$$

In incremental Forest, δ_ε contains the current path value in the variable r. When an update occurs in the file system, incremental Forest will change the representation and metadata if the update happens at the current path.

In the rest of the section, we will use the $\varepsilon_{old}(r)$ or simply r to get the old path.

Combination of Updates

$$\frac{\delta_\varepsilon, k \vdash \texttt{load}_\Delta(F, v, d)\ \delta_{F_1} \rhd (\delta_{v_1}, \delta_{d_1}) \quad \delta_\varepsilon, k \vdash \texttt{load}_\Delta(\delta_{F_1} F,\ \delta_{v_1} v,\ \delta_{d_1} d)\ \delta_{F_2} \rhd (\delta_{v_2}, \delta_{d_2})}{\delta_\varepsilon, k \vdash \texttt{load}_\Delta(F, v, d)\ (\delta_{F_1} \cdot \delta_{F_2}) \rhd ((\delta_{v_2} \cdot \delta_{v_1}), (\delta_{d_2} \cdot \delta_{d_1}))}$$

$$\frac{\delta_\varepsilon{}' = (\delta_\varepsilon, x \mapsto (m, \delta_m)) \quad (\delta_\varepsilon, x \mapsto (m, \delta_m)), k \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)}{\delta_\varepsilon{}', k \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \rhd (\delta_v, \delta_d)}$$

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.

$e :: s$

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

Incremental $e :: s$

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

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. We solve this by always applying the new path in the new environment to the path variable r. A tricky part here is we don't change the path variable. Instead, we add a new path variable into the $\delta_e nvironment$ so that Incremental Forest will not produce any extra environment change itself.

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

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

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

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

If reader find it difficult to understand $(\delta_{v_1} \otimes \delta_{v_2}, \rightsquigarrow valid_{12} \otimes (\delta_{d_1} \otimes \delta_{d_2}))$, it can be viewed as apply corresponding change to $((v_1, v_2), (b, (d_1, d_2)))$ and recheck the correctness of new metadata $(d_1{}', d_2{}')$

$s = \texttt{P}(e)$

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

Incremental $s = P(e)$

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

$$\frac{\varepsilon_{old} \neq \varepsilon\ \wedge\ b = [\![e]\!]^{\varepsilon_{new}}_{Bool}}{\delta_\varepsilon, P(e) \vdash \texttt{load}\ (F, v, d)\ \delta_F \rhd (\texttt{id}, \pi_1(\rightsquigarrow b))}$$

If nothing has changed in the environment, Incremental Forest will spare the labor of testing the bool expresiion. Otherwise, Incremental Forest will test the expression under new environment.

Incremental $s = s?$

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

$$\frac{r' = \varepsilon_{new}(r) \quad r \in \texttt{dom}(F) \quad r' \notin \texttt{dom}(\delta_F\ F)}{\delta_\varepsilon, s? \vdash \texttt{load}_\Delta(F, v, d)\ u \rhd (\rightsquigarrow Nothing, \rightsquigarrow (True, Nothing))}$$

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

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

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

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

$$\frac{\begin{array}{c}[\![e]\!]^{\varepsilon}_{\{Path\}} = \{n_1, n_2, \ldots, n_k\} \\ \forall i \in \{1, 2, \ldots, k\}, (\varepsilon, r \mapsto r/n_i, s) \vdash \texttt{load}\ F \rhd (v_i, d_i) \\ vs\ \texttt{maps}\ n_i \to v_i,\ ds\ \texttt{maps}\ n_i \to d_i,\ b = \bigwedge_{i=1}^{k} valid(d_i)\end{array}}{\varepsilon, [s \mid x \in e] \vdash \texttt{load}\ F \rhd (vs, (b, ds))}$$

To make Forest incremental, we make some changes in static comprehension specification. First, we abandon arbitrary type of comprehension and restrict it to be set of filenames n. Second, we write the loading path for every filename explicitly as r/n_i while static Forest exploit $e :: s$ implicitly to help it accomplish that. Third, 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 filename n_i.

$$vs(n) = v$$
$$ds(n) = v$$

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.

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

$$\frac{\begin{array}{c}\Delta_v = \texttt{id},\ \Delta_d = \texttt{id} \\ \forall n_i \in [\![e]\!]^{\varepsilon_{old}}_{\{Path\}} \cup [\![e]\!]^{\varepsilon_{new}}_{\{Path\}} \quad \Delta_v = \Delta_v \cdot \texttt{changeVOf}(n_i),\ \Delta_d = \Delta_d \cdot \texttt{changeDOf}(n_i) \\ valid_{all}(ds') = \bigwedge_{i=1}^{l} valid(d'_i),\ d'_i \in ds'\end{array}}{\delta_\varepsilon, r, [s \mid x \in e] \vdash \texttt{load}_\Delta\ (F, vs, ds)\ \delta_F \rhd (\Delta_v, valid_{all} \leftarrow \Delta_d)}$$

Since changes on the filename set is arbitrary, here we design a Δ_v to record all the changes on representation and Δ_d to record all the changes on metadata. Incremental Forest will iterate all of the filenames, find out how should we change the representation and metadata of each file and combine them together. The rule to determine the change of a single file is dicribed by `changeVOf` and `changeDOf`

$\boxed{\texttt{changeVOf}\ and\ \texttt{changeDOf}}$

$$\frac{\forall n \notin [\![e]\!]^{\varepsilon_{new}}_{\{Path\}},\ v = vs(n),\ d = ds(n)}{\texttt{changeVOf}(n) = \texttt{del}((n, v)), \quad \texttt{changeDOf}(n) = \texttt{del}((n, d))}$$

If the filename n doesn't exist in new environment, Incremental Forest delete corresponding representation and metadata from the set.

$$\frac{\forall n \in [\![e]\!]^{\varepsilon_{new}}_{\{Path\}} \setminus [\![e]\!]^{\varepsilon_{old}}_{\{Path\}},\ \varepsilon_{new}, r/n, s \vdash \texttt{load}\ F \rhd (v, d)}{\texttt{changeVOf}(n) = \texttt{add}((n, v)), \quad \texttt{changeDOf}(n) = \texttt{add}((n, d))}$$

If the filename n doesn't exist in old environment, Incremental Forest add corresponding representation and metadata to the set.

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

If the filename n is preserved in both environments, Incremental Forest delta-load the file recursively.

4 Proof

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

Assumption 4.1 (Atomic Update)**.** *Incremental Forest each time takes in one atomic update between*

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

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*

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

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

Proof. By induction
Case: $s = k$

1. By definition, if $\delta_F = \texttt{addFile}(r', \omega')$
   Case: $r = r'$

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

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

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

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

$$\texttt{id}\ v = v \quad \texttt{id}\ d = d$$

   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 $\texttt{id}$ scenarios. Thus we can conclude that Soundness holds for the constant specification case.

Case: $e :: s$

1. By induction, we assume that Soundness holds for $(\delta_\varepsilon \setminus r, r \mapsto ([\![r/e]\!]^{\varepsilon_{new}}_{Path}, \emptyset)), s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \triangleright (\delta_v, \delta_d)$, which means.

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

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

$$\begin{aligned}
&\varepsilon, e :: s \vdash \texttt{load}\ F\ \triangleright (v, d) \\
&\delta_\varepsilon, e :: s \vdash \texttt{load}_\Delta(F, v, d)\ \delta_F \triangleright (\delta_v, \delta_d) \\
&\varepsilon, e :: s \vdash \texttt{load}\ F\ \triangleright (\delta_v\ v, \delta_d\ d)
\end{aligned}$$

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

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

1. By definition

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

   By induction, Soundness holds for s_1.

$$\delta_\varepsilon, s_1 \vdash \texttt{load}_\Delta(F, v_1, d_1)\ u \triangleright (\delta_{v_1}, \delta_{d_1})$$

$$\delta_{v_1}\ v_1 = v_1' \quad \delta_{d_1}\ d_1 = d_1'$$

   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

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

$$\delta_{v_2}\ v_2 = v_2' \quad \delta_{d_2}\ d_2 = d_2'$$

   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

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

   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 $(\texttt{id}, \texttt{id})$, 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 (id 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.*

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

Proof. By induction

Case: $s = k$
By definition, if $\delta_\emptyset \vdash \varepsilon_{old} = \varepsilon_{new}$

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

Incrementality holds for this case.

Case: $s = e_1 :: s_1$

By definition,

$$\frac{(\delta_\emptyset \setminus r, r \mapsto ([\![r/e_1]\!]^{\varepsilon_{new}}_{Path}, \emptyset)), s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \emptyset \rhd (\delta_v, \delta_d)}{\delta_\emptyset, e_1 :: s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \emptyset \rhd (\delta_v, \delta_d)}$$

For the new delta environment δ'_ε,

$$\frac{\delta'_\varepsilon = (\delta_\emptyset \setminus r, r \mapsto ([\![r/e_1]\!]^{\varepsilon_{new}}_{Path}, \emptyset)) \ \wedge\ \delta_\emptyset \vdash \varepsilon_{old} = \varepsilon_{new}}{\delta'_\varepsilon \vdash \varepsilon'_{old} = \varepsilon'_{new}}$$

This means $\delta'_\varepsilon \in \Delta_{id}$. By induction,

$$(\delta_\emptyset \setminus r, r \mapsto ([\![r/e_1]\!]^{\varepsilon_{new}}_{Path}, \emptyset)), s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \emptyset \rhd (\emptyset, \emptyset)$$

Thus,

$$\delta_\emptyset, e_1 :: s_1 \vdash \texttt{load}_\Delta(F, v, d)\ \emptyset \rhd (\emptyset, \emptyset)$$

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, 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.

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

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,

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

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. □