File System Update Syntax and Properties

Yiming Wu

September 10, 2014

1 Introduction

We have discussed syntax and properties of Increamental Forest in other paper, but we have never discussed those of the updates thoroughly. We believe if we can design filesystem updates syntax carefully, we will boost Increamental Forest remarkably.

Our design of updates is based on two purposes:

- To discard redundant updates that will be covered later.
- To arrange the order of a set of updates correctly so that Increamental Forest will only need to traverse file system once.

2 Syntax of Updates

In this section, we will discuss syntax of update. Let's start with basic definitions:

$\boxed{Basic\ Definition}$

$$\begin{aligned} String\ u &::= \Sigma^* \\ Filesystem\ F &::= \{r_1 \mapsto T_1 \ldots r_n \mapsto T_n\} \end{aligned}$$

A file system F is *well-formed* if it encodes a tree with directories at the internal nodes and files and symbolic links at the leaves. More formally, F is *well-formed* if the following conditions hold:

- $\texttt{dom}(F)$ is prefixed-closed.
- $F(r) = \texttt{Dir}(u_1, u_2, \ldots, u_n)$ implies $r/u_i \in \texttt{dom}(F)$ for all i from 1 to n, and
- $F(r) = \texttt{File}\ \omega$ implies $r/u' \notin \texttt{dom}(F)$ for all u'

The syntax of updates is:

$\boxed{Syntax\ of\ Update}$

$$\begin{aligned} Elementry\ Update\quad & \rho ::= \texttt{addFile}(r, \omega) \mid \texttt{rmvFile}(r) \\ Update\ of\ Filesystem\quad & \delta ::= \rho \mid \delta_1 \cdot \delta_2 \mid \emptyset \end{aligned}$$

As we can see, syntax of updates goes mostly the same with our design in Increamental Forest paper.

$\boxed{Semantic\ of\ Update}$

$$\begin{aligned} \delta \quad &: \quad F \mapsto F \\ \emptyset\ F \quad &= \quad F \\ \texttt{addFile}(r, \omega)\ F \quad &= \quad F \cdot (r \mapsto \omega) \\ \texttt{rmvFile}(r)\ F \quad &= \quad \{r' \mapsto F(r') \mid r' \in \texttt{dom}(F) \setminus r\} \\ \delta_1 \cdot \delta_2\ F \quad &= \quad \delta_2\ (\delta_1\ F) \end{aligned}$$

The file system that Increamental Forest deals with should always be well-formed. Thus updates should be *well-formed* so that file system is always *well-formed*.

Definition 2.1 (Well-Formed Update)**.** An update δ is well-formed for file system F if $wf(F) \Rightarrow wf(\delta F)$

3 Equivalance of Updates

In real world file system, there are different sequence of updates but will have the same effect on file system. For example:

$$\texttt{addFile}(r, \omega) \cdot \texttt{rmvFile}(r) \cdot \texttt{addFile}(r, \omega') = \texttt{addFile}(r, \omega')$$

Definition 3.1 (Equivalance of Updates)**.** Two updates δ_1 and δ_2 is equivalant for file system F iff $\delta_1\ F = \delta_2\ F$. We write $\delta_1 \equiv \delta_2$.

We can prove that our *Equivalance of Updates* is an equivalant relation.

$\boxed{\textit{Equivalant Relation}}$

$$\begin{array}{ll} \delta_1 \equiv \delta_1 & \text{(REFL)} \\ \delta_1 \equiv \delta_2, \delta_2 \equiv \delta_3 \Rightarrow \delta_3 \equiv \delta_1 & \text{(ASSOC)} \\ \delta_1 \cdot (\delta_2 \cdot \delta_3) \equiv (\delta_1 \cdot \delta_2) \cdot \delta_3 & \text{(TRANS)} \end{array}$$

The first two properties are easy to prove, let's prove the third property.

TRANS.

$$\begin{array}{rcl} (\delta_1 \cdot (\delta_2 \cdot \delta_3))\ F & = & \delta_3(\delta_2(\delta_1\ F)) \\ ((\delta_1 \cdot \delta_2) \cdot \delta_3)\ F & = & \delta_3(\delta_2(\delta_1\ F)) \\ \delta_3(\delta_2(\delta_1\ F)) & = & \delta_3(\delta_2(\delta_1\ F)) \\ & \Rightarrow & \delta_1 \cdot (\delta_2 \cdot \delta_3) \equiv (\delta_1 \cdot \delta_2) \cdot \delta_3 \end{array}$$

□

Before we start defining equivalant algebraic operations, let's define some basic operations first.

$\boxed{\textit{PI Operations}}$

$$\pi(\texttt{addFile}(r, \omega')) = r$$
$$\pi(\texttt{rmvFile}(r)) = r$$
$$\pi(\delta_1 \cdot \delta_2) = \pi(\delta_1) \cup \pi(\delta_2)$$

$\boxed{\textit{Prefix and Sufix}}$

$$\begin{array}{lr} pre(.) = \emptyset & \\ pre(r/a) = pre(r) \cup \{r/a\} & \\ suf(r) = \{r\} & \textit{if } F(r) \textit{ is a } \texttt{File} \\ suf(r) = (\bigcup suf(r/u_i)) \cup \{r\} & \textit{if } F(r) = (u_1, \dots, u_n) \end{array}$$

Operation π gives out the path a basic update ρ is focusing on. Operation pre gives out the set of all the prefix of path r. Operation suf gives out the set of all the sufix of path r. With prefix and sufix operation, we can define partial order of path r.

Definition 3.2 (Partial Order of Path)**.** A path r_1 is the prefix of r_2, written as $r_1 \sqsubseteq r_2$, when $pre(r_1) \subseteq pre(r_2)$.

With these operations, we can progress to equivalent operations of update.

$\boxed{\textit{Equivalent Operations of Update}}$

$$\begin{array}{lrl} \rho_1 \cdot \rho_2 \equiv \rho_2 & \textit{if } \pi(\rho_1) = \pi(\rho_2) & (\texttt{CLOBBER}) \\ \rho_1 \cdot \rho_2 \equiv \rho_2 \cdot \rho_1 & \textit{if } \pi(\rho_1) \not\sqsubseteq \pi(\rho_2) \textit{ and } \pi(\rho_2) \not\sqsubseteq \pi(\rho_1) & (\texttt{SWAP}) \end{array}$$

We call the first operation `CLOBBER` rule, the second as `SWAP` rule. By intuition, `CLOBBER` rule will help us economize the sequence of updates that happen in one path. `SWAP` rule will help us changing the order of updates on filesystem so that we can deal with all those updates in the order we want.

Definition 3.3 (Normall Form Update)**.** A sequence of updates δ is normall form it is either

$$\begin{array}{ll} \rho & \\ \rho \cdot \delta_1 & \pi(\delta_1) \cap pre(r) = \emptyset \quad if\ \rho = \texttt{addFile}(r, \omega) \\ & \pi(\delta_1) \cap suf(r) = \emptyset \quad if\ \rho = \texttt{rmvFile}(r) \end{array}$$

Theorem 3.1 (Soundness)**.**