# *An Algebraic Approach to File Synchronization*

COMP 150 - Applied Functional Programming

September 24, 2012

## Quick Technicalities

(1) On page 176, the paper uses the notation $\pi \preceq \gamma$ to talk about paths in the filesystem. What algebraic laws apply to this relation? Do the laws justify the use of this symbol? Why or why not?

(2) The truth table on page 178 illustrates four possible values of the pair $(\pi_1 \preceq \pi_2, \pi_2 \preceq \pi_1)$. Please illustrate each entry in the table with a scenario of filesystem operations drawn from your own work.

## Algebraic laws

(3) Figure 2 on page 178 is not algebra. So what is it?

(4) Given that Figure 2 on page 178 is far simpler and easier to understand than Table 1 on page 179, what is the point of all these algebraic laws, anyway?

(5) The paper says that the laws are "sound and complete." What is meant by this claim? Were our stack laws sound and complete? What about the queue laws?

(6) How has the paper as a whole, and especially Section 4, contributed to your thoughts about when a set of algebraic laws is "good"? What do you think about "good" laws now?

## A deeper technicality

(7) The statement of the completeness theorem on page 178 seems rather technical.

   (a) What sort of alternative would you hope for?

   (b) Can you come up with an informal story or an example that explains why the completeness law is the way it is?

## Relation to Git, Darcs, Hg, and friends

(8) In Table 1 on page 179, roughly speaking, what kinds of laws are exploited by `git`?

(9) **Important:** Page 178 talks about using the laws as rewrite rules from left to right. In source-code control, are there any laws which are used in a right-to-left direction? If so, when, how, and why?

(10) **Important:** What does `git` do in the case of a conflict? Using the ideas from the paper, can you speculate about what reasoning might be used to justify `git`'s behavior? What alternatives can you imagine?

   On page 182 in Section 6, how does the discussion of conflicts compare with Git, darcs, or Mercurial? Are there usability issues? If so, how are they resolved?

(11) Also in page 182 in Section 6, the paper discusses issues about metadata. Do git, darcs, or Mercurial have similar issues? If so, how are they resolved? What are the usability problems?

(12) Page 181 talks about explicit *move* commands. There's a `git mv` command. What do you suppose it does?

## Challenge problems (for after class)

(13) How would you add "soft links" (also known as symbolic links) to the model described in this paper? In a file synchronizer, how would you want soft links to behave?

(14) If you wanted to extend the algebra to cover more of the behaviors needed in distributed source-code control, what sort of extensions would you consider?

(15) Your friend insists on adding *move* to the algebra. What happens to the algorithm described in the paper? How would you patch it?