Verifying filesystems in the ACL2 theorem prover: an application to FAT32

Mihir Parang Mehta* and Warren A. Hunt, Jr.

University of Texas at Austin, Department of Computer Science,
2317 Speedway, Austin, TX 78712, USA
{mihir,
hunt}@cs.utexas.edu
http://www.cs.utexas.edu

Abstract. We describe an effort to formally verify the FAT32 filesystem, based on a specification put together from Microsoft's published specification and the Linux kernel source code. We detail the proof approach we used and its pros and cons. We describe how this work is applicable to filesystems in general, and enumerate possible future applications of these techniques.

Keywords: interactive theorem proving, filesystems

1 Overview

Filesystems are ubiquitous in computing, and they have been of interest to the formal verification community for nearly as long as it has existed.

Here, we detail an effort to advance the state of the art by means of modelling the FAT32 filesystem at the binary level, and validating this model both through theorem proving and through co-simulation with the kernel implementation of FAT32. We begin with an overview of the model and the properties proved with examples; we proceed to a high-level explanation of the proof techniques used; and further we offer some insights about the low-level issues encountered while working the proofs. We end with some statistics pertaining to the magnitude of the proof effort and the running time of the proofs.

2 The models

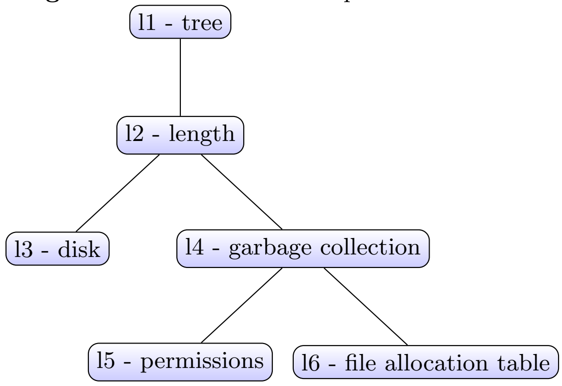
At this point in development, we have six models of the filesystem, here referred to as `l1` through `l6` (see table 1). Each new model *refines* a previous model, adding some features and complexity, and thereby approaching closer

* Please note that the LNCS Editorial assumes that all authors have used the western naming convention, with given names preceding surnames. This determines the structure of the names in the running heads and the author index.

Table 1. Models and their features

`l1`	The filesystem is represented as a tree, with leaf nodes for regular files and non-leaf nodes for directories. The contents of regular files are represented as strings stored in the nodes of the tree; the storage available for these is unbounded.
`l2`	A single element of metadata, *length*, is stored within each regular file.
`l3`	The contents of regular files are divided into blocks of fixed size. These blocks are stored in an external "disk" data structure; the storage for these blocks remains unbounded.
`l4`	The storage available for blocks is now bounded. An allocation vector data structure is introduced to help allocate and garbage collect blocks.
`l5`	Additional metadata for file ownership and access permissions is stored within each regular file.
`l6`	The allocation vector is replaced by a file allocation table, per the official FAT specification.

Fig. 1. Refinement relationships between models



to a model which is binary compatible with FAT32. These refinement relationships are shown in figure 1. Broadly, we characterise the filesystem operations we offer as either *write* operations, which do modify the filesystem, or *read* operations, which do not. In each model, we have been able to prove *read-over-write* properties which show that write operations have their effects made available immediately for reads at the same location, but also that they do not affect reads at other locations.

The first read-after-write theorem states that immediately following a write of some text at some location, a read of the same length at the same location yields the same text. The second read-after-write theorem states that after a write of some text at some location, a read at any other location returns exactly what it would have returned before the write.

By composing these properties, we can reason about executions involving multiple reads and writes.

3 Proof methodology

In *l1*, our simplest model, the read-over-write properties were, of necessity, proven from scratch, with the use of some rather complicated induction schemes. For reference, code listing 42 shows the induction scheme used for `l1-read-after-write-2`.

In each subsequent model, the read-over-write properties are proven as corollaries of equivalence proofs which establish the correctness of read and write operations in the respective model with respect to a previous model. A representation of such an equivalence proof follows.

4 Some proof details

As the models grow more complex, with the addition of more auxiliary data the "sanity" criteria for filesystem instances become more complex. For instance, in `l4`, the predicate `l4-fs-p` is defined to be the same as `l3-fs-p`, which recursively defines the shape of a valid filesystem. However, a "sane" filesystem requires also that each disk index assigned to a regular file be marked as *used* in the allocation vector, and that it be distinct from other disk indices assigned to files across the filesystem. These properties are invariants to be maintained across write operations; they simplify the verification of read-after-write properties by ensuring that write properties do not create an "aliasing" situation in which a regular file's contents can be modified through a write to a different regular file.

These properties, in the form of the predicates `indices-marked-listp` and `no-duplicatesp`, are packaged together into the `l4-stricter-fs-p` predicate, for which a listing follows.It is interesting to note that disabling l6-wrchs brought down the certification time for `l6` from 84 seconds to 64 seconds.

```
(defun l4-stricter-fs-p (fs alv)
  (declare (xargs :guard t))
```

```
(and (l4-fs-p fs)
     (boolean-listp alv)
     (let ((all-indices (l4-list-all-indices fs)))
          (and (no-duplicatesp all-indices)
               (indices-marked-p all-indices alv)))))
```

4.1 Citations

For citations in the text please use square brackets and consecutive numbers: [1], [2], [4] – provided automatically by LaTeX's `\cite` ... `\bibitem` mechanism.

4.2 Page Numbering and Running Heads

There is no need to include page numbers. If your paper title is too long to serve as a running head, it will be shortened. Your suggestion as to how to shorten it would be most welcome.

5 LNCS Online

The online version of the volume will be available in LNCS Online. Members of institutes subscribing to the Lecture Notes in Computer Science series have access to all the pdfs of all the online publications. Non-subscribers can only read as far as the abstracts. If they try to go beyond this point, they are automatically asked, whether they would like to order the pdf, and are given instructions as to how to do so.

Please note that, if your email address is given in your paper, it will also be included in the meta data of the online version.

6 BibTeX Entries

The correct BibTeX entries for the Lecture Notes in Computer Science volumes can be found at the following Website shortly after the publication of the book: `http://www.informatik.uni-trier.de/~ley/db/journals/lncs.html`

Acknowledgments. The heading should be treated as a subsubsection heading and should not be assigned a number.

7 The References Section

In order to permit cross referencing within LNCS-Online, and eventually between different publishers and their online databases, LNCS will, from now on, be standardizing the format of the references. This new feature will increase the visibility of publications and facilitate academic research considerably. Please base your references on the examples below. References that don't adhere to this style will be reformatted by Springer. You should therefore check your references thoroughly when you receive the final pdf of your paper. The reference section must be complete. You may not omit references. Instructions as to where to find a fuller version of the references are not permissible.

We only accept references written using the latin alphabet. If the title of the book you are referring to is in Russian or Chinese, then please write (in Russian) or (in Chinese) at the end of the transcript or translation of the title.

The following section shows a sample reference list with entries for journal articles [1], an LNCS chapter [2], a book [3], proceedings without editors [4] and [5], as well as a URL [6]. Please note that proceedings published in LNCS are not cited with their full titles, but with their acronyms!

References

1. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
2. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information, `http://www.ncbi.nlm.nih.gov`

Appendix: Springer-Author Discount

LNCS authors are entitled to a 33.3% discount off all Springer publications. Before placing an order, the author should send an email, giving full details of his or her Springer publication, to `orders-HD-individuals@springer.com` to obtain a so-called token. This token is a number, which must be entered when placing an order via the Internet, in order to obtain the discount.

8 Checklist of Items to be Sent to Volume Editors

Here is a checklist of everything the volume editor requires from you:

- [ ] The final LaTeX source files

- [ ] A final PDF file

- ☐ A copyright form, signed by one author on behalf of all of the authors of the paper.
- ☐ A readme giving the name and email address of the corresponding author.