Instructions for Formal Verification 2020 Reports

Romain Edelmann
EPFL

Viktor Kunčak
EPFL

Abstract

This meta-circular report explains how to write a report for the Formal Verification course. To write this abstract, we have first written an initial version of it. Then, after the rest of the report was complete, we revised the abstract to make sure that it contains the most important points. As part of this revision, we have concluded that we did not set as great of an example as we hoped, but may have shown some simple latex commands and expected section structures.

2012 ACM Subject Classification Software and its engineering → Software verification and validation

Keywords and phrases formal verification, theorem proving, concurrency, decision procedure

1 Introduction

This topic of this report sits at the intersection of two areas: formal verification as the subject line, and the general topic of writing a scientific report in computer science. That said, please keep in mind that this is not a serious report whose contents is meant to be taken overly seriously, but rather a report template that shows how to use latex to typeset your report.

1.1 Formal verification.

Formal verification is computer science subject in at least two aspects: it applies rigorous analysis to *computer systems* (thus, computing is its subject), and it uses algorithms and their implementations in tools to help automate this task (thus, computing is a tool to make the task more feasible). In principle, it is possible to do formal verification manually, and early formal methods such as the Z notation focused on specification and manual proofs [**?**]. Today we expect some degree of **automation** from specification and verification process.

Among main directions for automation in formal verification are:

1. building better theorem provers, advancing SMT solvers, SAT solvers, and first-order theorem provers
2. building methods to infer inductive invariants, such as abstract interpretation
3. designing type inference mechanisms and tactics for proof assistants

Thanks in part to student projects, we were able to see aspects of all of these directions.

1.2 Report formats

Journal and conference proceedings publishers tend to specify formatting instructions for their contributions, such as Dagstuhl Publishing proceedings instructions[1] on which this very

[1] `https://submission.dagstuhl.de/documentation/authors`. It is better to not use URLs if you can provide a citation entry instead. Also, avoid lengthy footnotes in general. If the footnote is long, it distracts the reader in two ways: first, visually, because they have to skip to the bottom of the page and then go back. It is also bad in terms of readers' focus, because it often involves a digression. Digressions are best avoided, because the reader tends to forget what the main point of the text was. On top of all that, LaTeX tends to split very long foonotes across multiple pages, which makes reading them almost as bad as footnotes in books. If not, then a large footnote occupies a large portion of the bottom of a page, which does not look good. Footnotes also have too small font, so they are hard to read. In

report is based, but typically the publishers leave the content of the articles to the reviewers, editors, and mentors who provide initial advice for scientific writing.

1.3 Prior work

We would cite here the main work on which we build our solution. We should clarify why previous solutions do not solve the problem we were aiming to solve, or at least why we think we could not google the solution.

Our approach

▶ Remark 1. Scientific communication is part of a larger area of communication studied in humanities departments throughout the world, but we have not pursued these studies, so we cannot comment on the overlap of concepts learned there with the specific subject of scientific reporting, which scientists, somewhat arrogantly, tend to believe has mostly to do with science and not with reporting.

We are now ready to state the main rationale behind this report.

▶ **Proposition 2.** *Content is ultimately more important than the presentation, but adopting certain presentation conventions gives the author an easily gained advantage that may positively bias the readers and may help them accept author's ideas.*

The main challenges in writing the report sample is that it should not be too long or too confusing. Another challenge is that it may be written in a relatively short period of time, though this aspect may well be shared by actual reports.

Contributions

In summary, in this report we have taken a hopefully decent-looking non-copyrighted template for proceedings and wrote an example file.

The main contributions of this report are the following:

1. We illustrate how to use latex, though we hope that this is a skill already acquired. If not, once can consider vast LaTeX literature [**?**].
2. We suggest a list of sections.
3. We make various remarks that serve primarily to fill the space.

2 Example

In this example section, we show how to our system works. It is a good to start a section with a short sentence (such as previous one) stating the purpose of that section, before continuing with any other remarks.

An example section is very helpful. It should give the reader the feeling for *what* your work accomplishes, without going into details of *how* it does it.

We have built a verification system (once upon a time, in 2006). Figure **??** shows an example of a verification session.

---

summary, it is best to avoid footnotes, and avoid long ones entirely by restructuring the text to make the remarks appear inline or not at all.

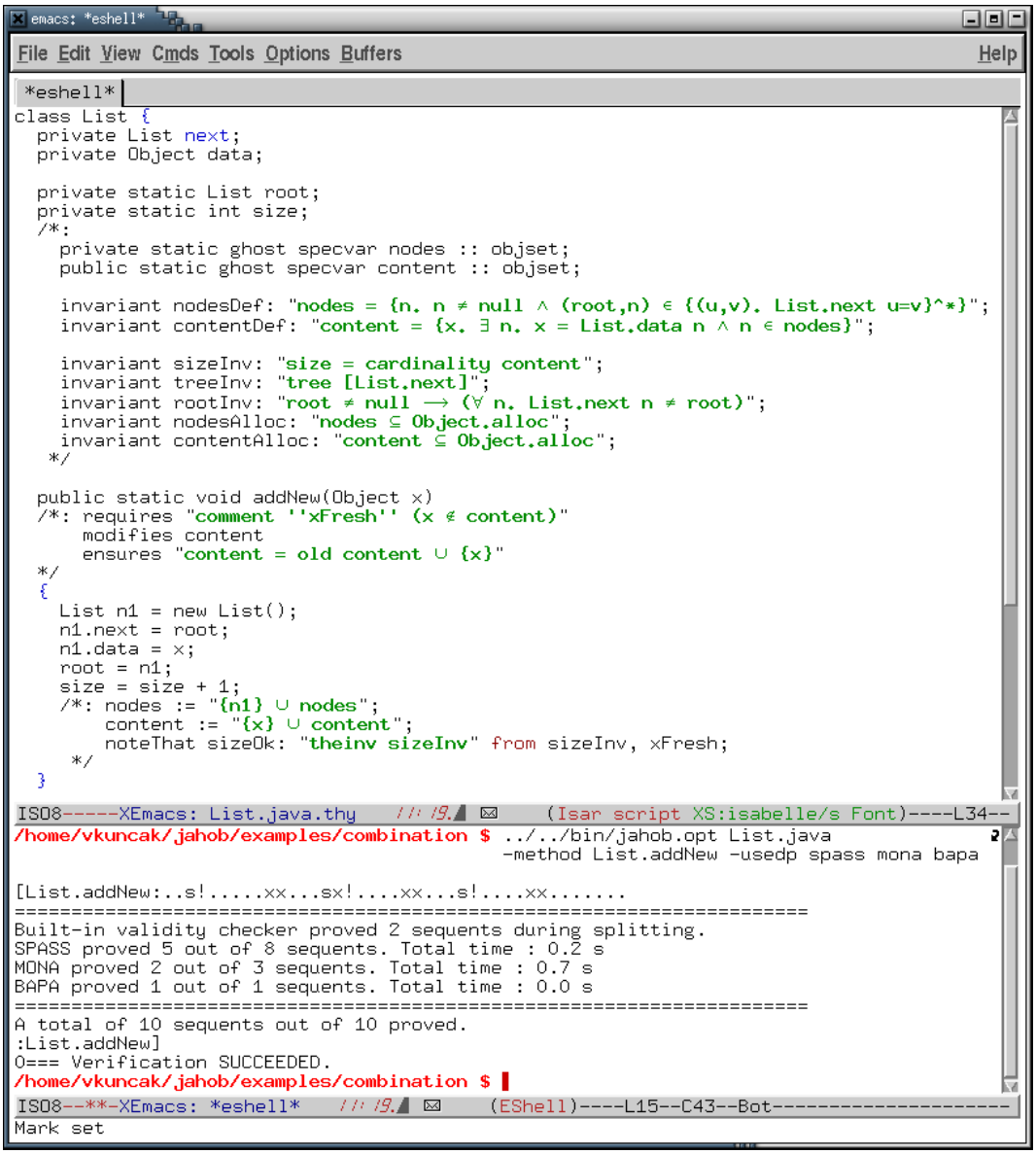

Figure 1 A screenshot of a verification session.

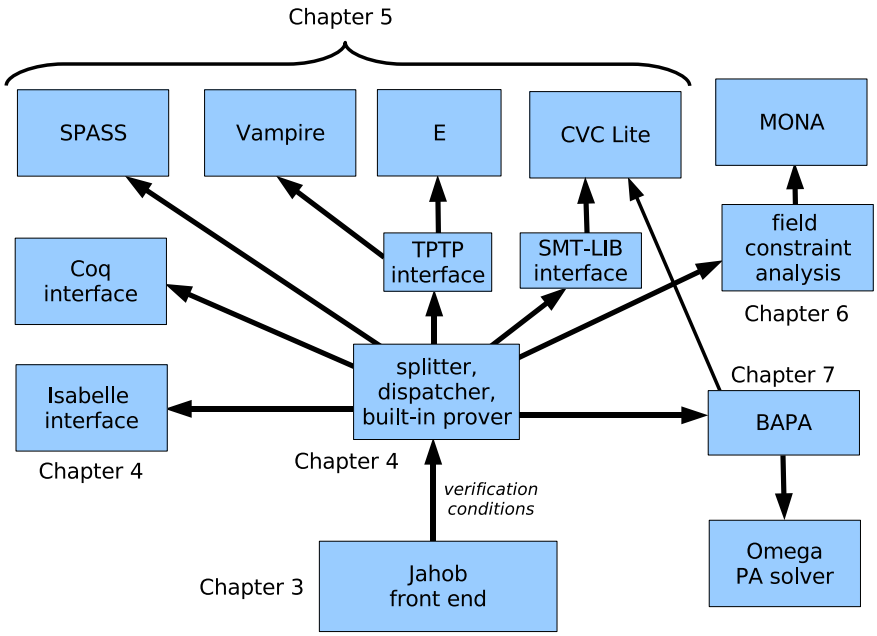

Figure 2 An Overview of the Jahob Verification System

3 Overview

By year 2006, one of the authors, together with collaborators, build a verification system called Jahob. Figure **??** shows an overview of this system. Different aspects of the system are outlined in the thesis [**?**].

4 A Solver

In this section, we explain the solver we built. This would be a sketch of a more technical section.

For the sake of illustration, Figure **??** shows a pseudo code of a SAT solver based on DPLL procedure [**?**].

Note that the above algorithm is simplified in several respects:

- It uses mathematical sets to store clauses. This is very inefficient
- It uses non-deterministic constructs such as selecting an element from the set. It thus leaves the policy on how to do actual selection under-specified.

In the rest of this report we could see how to make it more efficient (though we will, in fact, not). We present a part of the search procedure used in actual CafeSAT code in Appendix **??**.

5 Implementation

We have implemented our approach in ocaml. Our system is available in the following repository:

`https://github.com/epfl-lara/jahob`

```
def DPLL(S: Set[Clause]) : Bool =
  val S' = subsumption(UnitProp(S))
  if ∅ ∈ S' then false // unsat
  else if S' has only unit clauses then true // unit clauses give e
  else
     val L = a literal from a clause of S' where {L} ∉ S'
     DPLL(S' ∪ {{L}}) || DPLL(S' ∪ {{complement(L)}})

def UnitProp(S: Set[Clause]): Set[Clause] = // Unit Propagation (BCP)
  if C ∈ S, unit U ∈ S, resolve(U,C) ∉ S
  then UnitProp((S - {C}) ∪ {resolve(U,C)}) else S

def subsumption(S: Set[Clause]): Set[Clause] =
  if C1,C2 ∈ S such that C1 ⊆ C2
  then subsumption(S - {C2})  else S
```

Figure 3 This figure should show something new and interesting but, for illustration pruposes, it is merely a sketch of a DPPL algorithm for SAT that should be familiar from Lecture 3.

and build instructions are in file `README.md` in the top-level directory.

6 Experimental Evaluation

We have evaluated our solver on benchmarks taken from standard benchmark suites [**?**, **?**, **?**, **?**]

We have selected only those benchmarks that portray our solver in good light. To make our results look more convincing, we have added some benchmarks that our solver cannot solve (and also no other solver can solve, because they encode open conjectures in number theory).[2]

7 Related Work

In this section we argue that we are very well aware of the previous work in the area. This may be more or less realistic for a project report in a course, but is crucial for reports that reflect a longer piece of work.

Dependent types are very popular approach in programming language community [**?**]; a title that is a good reflection of the superficial nature of this very report sample. Often the idea is that some forms of type dependency are easy, and general form of type dependency allows us to prove complex properties. From this the reader should get the hint that using dependent types makes verification easy. In reality, verification is never easy; it is merely the only way to have high confidence that software is correct.

Types that refine type systems of languages such as Haskell and ocaml with predicates demonstrate an appealing combination between automation and expressive power [**?**]. They appear to have been used to prove properties somewhat simpler than those used in Stainless case studies [**?**], but have advantage in inferring properties that are not inductive yet fit the provided invariant templates.

[2] You should adopt a more convincing approach to evaluation if possible. For a short project, make it clear that you understand where your limitations are, rather than pretending they do not exist. Also, do not use footnotes.

8 Future work

In this section we point out to those aspects that we would have liked to explore had there been more time. We will be careful not to say too much that can be used against us and, where aspects are left unfinished, there were specific technical challenges that we encountered, beyond the lack of time (which is always a problem).

In terms of this report itself, there is only so much that one can do in a short amount of time in illustrating what a report should look like. Thus, we encourage students to contact us for feedback on initial versions of the report, so that we can point out to specific ways in which it can be improved.

9 Conclusions

Admittedly, it would have been possible to write a detailed treatise on how to write reports. However, we hope that the following example will already streamline the format and set some expectations on how the report should look like in its form and structure, despite the lack of substance and admittedly confusing nature of the matter.

Acknowledgements. The authors of this report would like to thank the funding agencies such as Swiss Science Foundation, as well as EPFL, for creating a wonderful environment where teaching and research can co-exist in synergy. The authors thank their parents, primary school teachers, as well as their undergraduate and doctoral advisors for improving their writing skill, as well as science camp instructors and professors at the undergraduate study for fostering their interest in logic and functional programming. The authors also thank students in the course and hope that they will consider future projects with our lab.

A Appendix: Listing of our main verified function

This appendix lists the full function whose fragments we only briefly sketched in Section **??**.

```
private[this] def search(): Results.Result = {
  val topLevelStopWatch = StopWatch("toplevelloop")
  val deduceStopWatch = StopWatch("deduce")
  val decideStopWatch = StopWatch("decide")
  val backtrackStopWatch = StopWatch("backtrack")

  topLevelStopWatch.time {
    deduceStopWatch.time {
      deduce()
    }
    if(status == Conflict)
      status = Unsatisfiable
    val timeout: Option[Int] = Settings.timeout
    var elapsedTime: Long = 0 //in ms
    while(status == Unknown) {
      val startTime = System.currentTimeMillis
      decideStopWatch.time {
        decide()
      }
      var cont = true
      while(cont) {
        deduceStopWatch.time {
```

```
          deduce()
        }

        if(status == Conflict) {
          backtrackStopWatch.time {
            backtrack()
          }
        } else {
          cont = false
        }
      }
      val endTime = System.currentTimeMillis
      elapsedTime += (endTime - startTime)
      timeout.foreach(timeout => if(elapsedTime > 1000*timeout)
        status = Timeout)
    }
  }
  status match {
    case Unknown | Conflict => sys.error("unexpected")
    case Timeout => Results.Unknown
    case Unsatisfiable => Results.Unsatisfiable
    case Satisfiable => Results.Satisfiable(model.map(pol => pol==1))
  }
}
```

It may appear that we have verified a SAT solver. In fact, we have not done so ourselves. Others, however, have successfully generated code from SAT solver formalizations inside Isabelle/HOL proof assistant [**?**]. Now we just cited a paper in an appendix. If it was important for this report, it would have been better to cite it inside the main body of the paper.

We should not use appendix just to make it look like we have a detailed report. In fact, we should instead link to a git repository. For example, the code snippet above is available from here:

https://github.com/regb/cafesat/blob/77f3c582f7b8baaf20596c08cb689508ccf0a306/src/main/scala/cafesat/sat/Solver.scala

As you may know from reading github documentation, you can refer to the current version of the file, or, like above, to a specific commit that will not change unless you rewrite history after accidentally committing a password into git.

B Appendix: Additional Information

Here we could present, for example, additional experiments that did not fit into the length of the main paper. If we do not have a page limit, the only reason to have appendices is when your report is too long for your readers and you wish to provide additional information that may be helpful for someone going into depth, but that is better presented as a report than as some files in a git repository (or is too important to be left only in a git repository).