Developing a Usability Oriented ACL2 Test Suite

Ryan Ralston[1]

University of Oklahoma, Norman OK 73019, USA,
rlralston@gmail.com

Abstract. Much of the work in advancing the adoption of automated theorem provers (ATP) in software development is in making ATPs more accessible to larger groups of users. Accessibility has been enhanced with new development environments, adding features to manage the complexity, and teaching the use of the tools at the undergraduate level. This research investigates the ACL2 system and workshop libraries, which contain over 60,000 theorems, to demonstrate how usability changes over time, as hardware and software improve. The goal is to estimate usability by measuring ATP characteristics that are likely to affect usage behaviors. In particular, this work focuses on elapsed time in completing proofs of theorems.

Keywords: software verification, automated theorem proving, usability

1 Motivation

Automated theorem provers (ATP) are potentially viable tools for applying formal methods to verifying executable software. Some of the most common ATPs implement their logic into functional languages but work has been done to model object oriented code in ATP logics to verify properties. Jinja is a Java-like system that implements a compiler and virtual machine with formal semantics that are type safe. The proof is machine-checked in Isabella/HOL [1]. In similar work, Spec# is a compiler and runtime for a superset of C# that includes syntax for specifying properties of methods. The Spec# system creates runtime conditions from the specifications and proves correctness with the ATPs Simplify and Zap [2, 3].

ACL2 implements a side-effect free Common Lisp based programming language and logic. It has been used to implement a model of the Java virtual machine [4, 5]. The M5 model created in this work also demonstrates how a powerful ATP can still be difficult to use in a industrial setting. One of the lemmas introduced in M5 is annotated in the source code with a note that it takes over 500 seconds for ACL2 to complete its proof. It is unlikely non-experts[1] will have the confidence and patience to succeed in verifying software properties that require such long computations. Fortunately, ACL2 version 6.0 can prove that

[1] The M5 model and lemma were created by one of the developers of ACL2.

lemma on the system described in Section 2 in 90 seconds. Improvements of this magnitude have an impact on ATP usability.

The existing work shows the feasibility in verifying properties of industrial software, including software written in imperative languages, by converting to a functional model and applying an existing ATP. That step, alone, is daunting, but the results also show the tremendous amount of expertise needed to use the ATP once converted. This paper describes early research creating a biased theorem test suite that can be used to judge improvements in usability.

1.1 Related Work

The package "Thousands of Problems for Theorem Provers" (TPTP) is a test set of theorems that can be used to measure performance during the development of ATPs. The theorems are intended to be solved with specified, minimal sets of axioms [6]. The TPTP library is used to compare ATPs in the CASC competition [7]. The research reported in this paper seeks metrics that demonstrate a user's ability to configure the ATP to prove meaningful properties of software.

ATP have been integrated into existing development environments. The language features added in Spec# let users express properties for an underlying ATP to verify [3]. Translating specifications to a C#-like syntax should make new users more comfortable using an ATP, but limits expressibility that would be valuable with experience. A DrScheme extension, Dracula, embedded ACL2 into the development environment to be used in undergraduate computer science courses [8]. One of the instructor findings in a first year course was "the difficulty of generating proofs in ACL2 does not always scale down with the complexity of the problem" [8]. This is indicative of the learning curve necessary to gain value in using ATPs in software development. The ACL2 Sedan integrates ACL2 into Eclipse [9]. The Sedan simplifies the learning curve by being an Eclipse plug-in while leaving the expressibility unchanged. Additionally, a 3D visualization of ACL2 proof trees has been created to replace the default text representations [10]. So far, case studies do not indicate it being useful to non-expert users, but it remains a promising idea.

Finally, ACL(p) is an extension of ACL2 that parallelizes the theorem prover. The author is motivated to create an ACL2 that provides feedback on proofs as quickly as possible. The research categorizes ACL2 theorems into 4 groups. The first group consists of short proofs that would not see much improvement and a threshold is 5 seconds is used to create this group. The last group is theorems with high potential speed-up when running as parallel processes [11]. It is anticipated that most users will experience a variety of answer times between incredibly short and exceedingly long. One of the goals of the research that is the basis of this paper is to create a test set that includes both.

2 Data

ACL2 theorems are collected in *books*. The system is distributed with a set of common books, which will be referred to as the *system books* in the rest of this

paper. In addition, the ACL2 community holds workshop events regularly. The work presented at these events is collected and distributed in the *workshop books*. Combined there are over 60,000 theorems in the system and workshop books.

The books generate certification files that summarize the proof results. The summaries include the number of rules used in a proof, the prover steps counted, the total time in seconds to prove a theorem, and the prove time in seconds which ignores time devoted to non-proving tasks such as writing output. In this paper, ACL2 data is gathered from a system configured avoid generating much output during a proof. Therefore, the analysis investigates total time because the other non-proving tasks are harder for users to control. The difference between the two numbers is generally small anyway.

The ACL2 books are a valuable data set for researching ATP usability. The theorems in the books contain user artifacts such as the use of hints. Many of the books are commented with stories explaining intent. The theorems proved in the system books are a standard set that is considered useful to other users and workshop books are supplementary material to research interests so it is assumed the books are non-trivial theorems. The theorems were created over several years, beginning in 1999.

From the books data, we can investigate user experience based on answer time and complexity. Using the workshop books, we can also investigate temporal changes to usability based on the year of the workshop. This research subscribes to the belief explained by Larry Wos in research problem #23 that reducing the answer time increases the usability of an ATP [12]. Schumann explained that most ATPs implement answer time cutoffs because allowing long-running proofs reduces usability for an unlikely gain. He writes that most systems set the cutoff at a few seconds with anything over a minute being rare [13]. ACL2 does not force a cutoff, but it is our intuition that most users have an internal cutoff when using the system. Once the cutoff is passed, the user will stop the current task and try to configure the system differently.

2.1 System Descriptions

The raw data was gathered on 3 machines. One machine is an AMD Phenom II X6 1090T 3.20 GHz Windows 7 PC with 16.0 GB RAM. Of the machines, it is the only desktop computer so it will be referenced as the desktop throughout the paper. The other two machines are laptops. The high performance laptop is a Dell XPS L511Z laptop with an Intel Core i5 - 2410M 2.3 GHz processor and 6.00 GB of RAM. It is also running Windows 7. The low performance laptop is a ten-year-old Gateway Pentium 4 2 GHz with 256 MB RAM. It uses Windows XP Service Pack 2 as an operating system. The low performance laptop is included in the gathering because many of the original theorems in the books and workshop were created over 10 years ago.

The books were tested using ACL2 version 6.0 on all machines. The desktop and high performance laptop use an ACL2 built on 64-bit Clozure Common Lisp (CCL) 1.9. The low performance laptop uses the 32-bit CCL 1.9. To control for effects caused by the underlying Common Lisp distribution, an additional set of

Computer	**Lisp**	**t ≤ 0.5 s**	**0.5 s < t ≤ 1.0 s**	**1.0 s < t ≤ 5.0 s**	**t > 5.0 s**
Desktop	CCL	98.0%	1.3%	1.0%	0.2%
Desktop	SBCL	97%	1.4%	1.1%	0.2%
High Performance	CCL	97%	1.5%	1.3%	0.2%
Low Performance	CCL	93%	2.8%	3.2%	0.8%

Table 1. System Results by Computer

data was gathered on the desktop using ACL2 built with Steel Bank Common Lisp (SBCL).

2.2 Results

Table 1 breaks the system books results into buckets based on time. The majority of the results are less than half a second. Even on the low performance computer, which takes almost 2 hours longer to finish proving the system books, 96% of the results are less than 1 second. Comparing performance using the entire suite is made difficult by a significant ceiling effect. With distributions like this, it is common to calculate a skewness by dividing the difference between the data's mean and mode by the standard deviation. A positive skewness indicates the right tail of the probability density function is longer than the left but the distribution is concentrated on the left-side. The skewness of the desktop using CCL is 78.

	1999	2000	2002	2003	2004	2006	2007	2009	2011
Theorems	2864	1310	1204	2348	2112	1108	558	644	65
Rule Count μ	16.2	12.6	14.7	15.7	14.4	12.3	21.4	21.8	25.4
Prover Steps μ	37700	20700	28800	23200	20800	5940	25100	60400	2213
Prove Time μ	0.22	0.17	0.18	0.14	0.36	0.04	0.42	0.53	0.13
Prove Time max	49.4	53.3	105	51.1	62.4	6.47	30.6	56.4	3.60

Table 2. Workshop Results by Year

The summary breakdowns of workshop books is shown in Table 2. We had expected performance to change over years in the workshop books as hardware and ACL2 improved. Instead, we see very static performance across average prove time, maximum prove time, and average prover steps counted. But it should be noticed that in 2007 and 2009, the number of rules counted in the summary goes up and the number of theorems goes down. The average prove times are also higher but are still less than half a second and the maximum prove times does not increase. Conceivably, ACL2 usability is improving and successful users are able to prove their desired theorems with fewer lemmas.

3 Test Suite

A goal of this research is constructing a test suite that over-emphasizes long-running theorems. By making improvements to the answer times on long-running theorems, it is anticipated users can achieve their intended results with fewer lemmas. The test suite is also intended to represent usability, and part of ACL2 usability is organizing theorems into books and proving those lemmas prior to their associated theorems. We want the test suite to incorporate lemmas, since over time we would expect their need reduced, and to include short running proofs because it is the common use case.

Our test suite was constructed by accepting a Lisp file if it contained a theorem with a proof that took over a certain threshold time on the desktop using CCL. The resulting files were pruned for duplicates that exist, such as workshop books that were promoted to system books. Three thresholds were tested and the suites generated were compared by size of the suite and reduction in skewness. The three thresholds were 1, 5, and 10 seconds. The 1 second suite only reduced the skewness from 78 to 40. The 5 second suite reduced the skewness to 22 and the 10 seconds reduced it to 19, but had fewer than 1000 theorems. The 5 second suite was selected because it reduced the skew almost as much as the 10 second cutoff, and it included 8.7% of the theorems. The total time of the test suite is 57.6% of the time on the desktop using CCL.

The files in the system test suite and workshop test suite are described in Table 3 and Table 4 respectively. The rest of this paper is interested in showing how the test suite can illustrate performance improvements using the highest performing and lowest performing computers. The workshop test suite has not successfully been tested on the low performance laptop, so the remaining discussion is only using the system test suite.

The theorem performance in the test set were paired across all systems. The desktop using CCL outperformed the other systems on almost all theorems so it was selected as a baseline for analysis. The theorems were sorted by prove time on the desktop. The difference between the baseline prove time and each of the corresponding theorems were calculated. In Figure 1, the x-axis is the index in the sorted theorem list. The y-axis is the difference between the baseline system. The more time consuming theorems show greater differences between systems. The high performance laptop and desktop with SBCL both scored similarly. The difference from the baseline computer being relatively small except for the longest running theorems. The low performance laptop begins deviating much earlier. It is not visible in the graph, but the greatest difference is 649 seconds.

The nature of the curve suggests a power law relationship between time and some measure of complexity. The theorem summaries include prover steps counted, which is the best available metric for complexity in the data. The paired theorems were graphed on a loglog curve with prover steps across the x-axis and time across the y-axis. Figure 2 shows the curves for all systems. Using the graph, it is plain to see a difference between the low performance laptop and the other systems.

Book	File	Theorems
unicode	utf8-decode	76
taspi	fringes-guards	42
taspi	bdd-functions	82
taspi	btrees-bdds	13
taspi	fringes-taspi	47
system	convert-normalized-term-to-pairs	4
rtl	drnd	110
rtl	fadd	72
rtl	lextra	56
rtl	lior	55
rtl	setbits-proofs	20
ordinals	ordinal-isomorphism	46
models	find-k!	67
models	theorems-a-and-b	34
models	tmi-reductions	90
models	universal-never-returns	5
misc	dijkstra-shortest-path	126
misc	reverse-by-separation	23
defexec	fpst	63
data-structures	memtree	76
countereg-gen	splitnat	87
countereg-gen	switchnat	21
concurrent-programs	inv-persists	122
concurrent-programs	stutter2	54
coi	eric-meta	100
coi	meta	87
coi	base	61
coi	erase	35
coi	finite	112
coi	gacc2	63
coi	gacc3	172
coi	gax	81
coi	ram	127
coi	ram2	20
coi	ram2b	57
coi	tr-path-connection	92
coi	wrap	80
coi	multicons	13
coi	pm	117
coi	loglist	39
coi	super-ihs	416
arithmetic-5	logand-helper	8
arithmetic-5	logand	70
arithmetic-3	mod-expt-fast	7

Table 3. System Books in Test Suite

Book	Year	File	Theorems
embedded	1999	Proof-Of-Correctness-OneCycle	326
ivy	1999	close	26
ivy	1999	prop-subsume	5
ivy	1999	pull-pulls	22
ivy	1999	pull-sound	30
ivy	1999	sk-step-sound	21
ivy	1999	sk-useless	9
ivy	1999	sk-xbuild	50
ivy	1999	solution2	5
mu-calculus	1999	semantics	19
ste	1999	assertion	28
ste	1999	fundamental	9
ste	1999	inference	31
ste	1999	lemma-4	34
ste	1999	run	33
ste	1999	state	36
lusk-mccune	2000	stepproc1	39
lusk-mccune	2000	stepproc2	14
sumners1	2000	cdeq-phase2	36
cowles-flat	2002	flat-ackermann	16
cowles-flat	2002	flat-reverse	14
cowles-gamboa-van-baalen_matrix	2003	matrix	527
gamboa-cowles-van-baalen	2003	kalman-proof	78
moore_rockwell	2003	memory-taggings	91
ray-matthews-tuttle	2003	concrete-ltl	8
tsong	2003	shim	24
legato	2004	generic-theory-alternative-induction-mult	8
legato	2004	generic-theory-loop-invariant-mult	8
legato	2004	proof-by-generalization-mult	13
ruiz-et-al	2004	q-dag-unification	185
schmaltz-borrione	2004	collect_msg_book	23
schmaltz-borrione	2004	local_trip_book	34
schmaltz-borrione	2004	routing_local_lemmas	64
sumners-ray	2004	records	53
schmaltz	2007	GeNoC	45
schmaltz	2007	doubleY-routing	81
schmaltz	2007	getting_rid_of_mod	10
schmaltz	2007	routing_local_lemmas	28
hardin	2009	deque-thms	57
pierre-clavel-leveugle	2009	ATM-TMR	33
verbeek-schmaltz	2009	GeNoC-misc	127
verbeek-schmaltz	2009	GeNoC-scheduling	21
verbeek-schmaltz	2009	simple	71
verbeek-schmaltz	2009	XYRouting	49

Table 4. Workshop Books in Test Suite

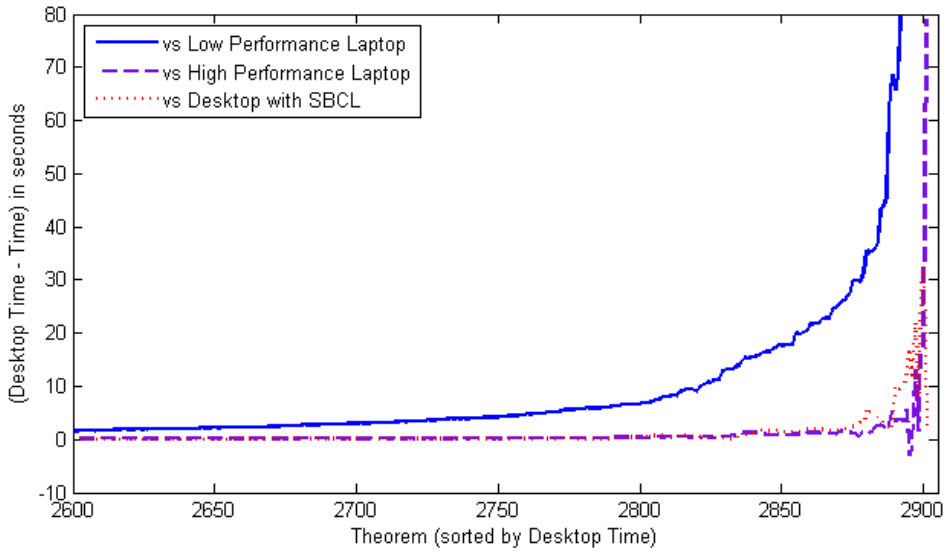

Fig. 1. Difference in Time from Desktop with CCL

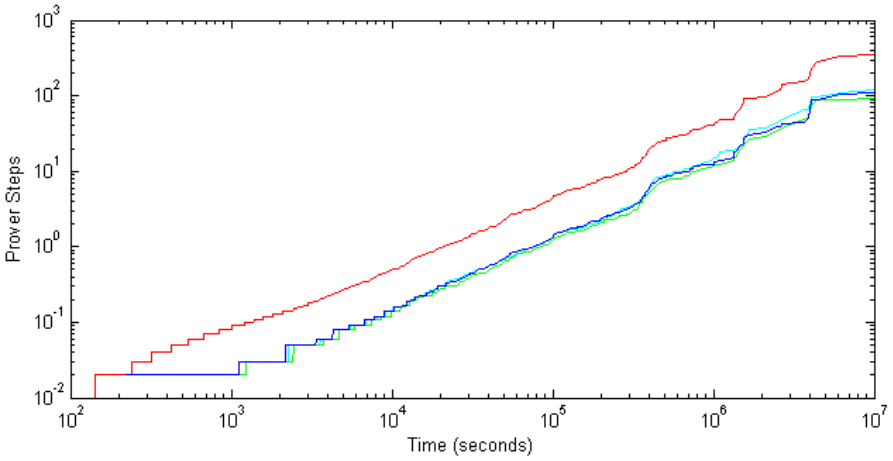

Fig. 2. Prover Steps vs Time loglog Graph

The comparison can be expanded using the power law relationship. The low performance system has a slope of 0.84 and an intercept of -2.5. The desktop with CCL has a slope of 0.80 and an intercept of -4.0. Since performance approaches 0 seconds, the intercept will approach -inf. General increases in performance should reduce the intercept. Improvements that favor increased performance

on simple theorems will increase the slope. Similarly, improvements that favor increased performance on complex theorems will decrease the slope.

4 Conclusions

It is difficult to learn to use ATPs, in general, and ACL2, specifically. To incorporate the advantages of software verification more conveniently in industry, the tools should be more usable. While efforts are being made to improve education and tools, none of these efforts are modeling effective user behavior and applying the results to improving the system. This test set was created based on the assumption that a system capable of solving complex problems quickly is more usable. The test set reduces the ceiling effect without ignoring that complex proofs are generally proved after proving simpler lemmas. It was demonstrated that time and prover steps have a power law relationship, which can be used compare performance differences on a loglog graph.

There is a small difference between the system and workshop books, but not as significant as expected. The performance over the years in the workshop books showed results that were more static than expected. There is some evidence in the results from 2007 and 2009 that less work may be required to prove theorems, if it is assumed that the results from most workshops stem from comparable investments of user effort. It may also be that new users adopting the system over the years has changed the average user behavior submitting books to the workshops. Upcoming work will search for insight into the change into those last 2 years.

Immediate future work will investigate the effect of specific features in ACL2 on the answer times and complexity of the proofs. A common user will guide the theorem prover with hints. Many successful users limit the search space for ACL2 by encapsulating lemmas where they are needed or disabling theorems preemptively. The system also has several macros that expand into common theorem structures that are currently ignored in this analysis. These user configuration options improve usability for the users that are aware of how to use them. The books, having been written by skilled users, will use the available options more efficiently than an average user. If future work identified common, effective configuration options, new and average users could be helped up the ACL2 learning curve with focused documentation or user tools that recommended the options.

The desktop with CCL outperformed the high performance laptop and the desktop with SBCL, but there was not a clear difference between the desktop with SBCL and the laptop. The biggest performance difference between the desktop and high performance laptop is the RAM. Based on anecdotal observation, SBCL uses significantly more memory than CCL. Using the test suite, we could search for the effect of memory usage on answer times.

The most significant constraint on this research is access to high fidelity user data. The books lack any information about the time and effort invested by a user to configure source code to make it possible for ACL2 to accept a theorem. The books are also created by a skilled set of users so their use of the system is

likely more efficient than most users. A better investigation could be conducted with data gathered directly from a user's interaction with ACL2, but data of this kind is difficult to obtain. One approach could be to implement a data gathering system with a basic data input protocol. The ACL2 IDEs, such as the aforementioned ACL2 Sedan and Dracula and the newer Proof Pad [14], could be modified to let users optionally send updates to the data gathering server as they work.

All three of these IDEs have been used to teach ACL2 to undergraduate students. An influx of novice users' data would balance the majority of the currently existing data, which reflects use by experts. The novice users would also provide data that could lead towards understanding the ACL2 learning curve and how it could be improved.

The goal of this research project is understanding what effects usability and using that knowledge to improve ATP usability. Successfully admitted, long-running theorems may be anomalies of ACL2 or anomalies of user behavior. Either way, the test set represents meaningful theorems that can be proven in non-trivial user time. Improving the performance on this test set will either make the theorems less anomalous to ACL2 or increase the number of users that would find similar theorems. In both cases, the system should be considered more usable.

References

1. Klein, G., Nipkow, T.: A machine-checked model for a Java-like language, virtual machine and compiler. ACM Transactions on Programming Languages and Systems **28**(4) (2006) 619–695
2. Barnett, M., Leino, K.R.M., Schulte, W.: The spec# programming system: An overview. In: Proceedings of the 2004 International Conference on Construction and Analysis of Safe, Secure, and Interoperable Smart Devices, Springer (2004) 49–69
3. Barnett, M., yuh Evan Chang, B., Deline, R., Jacobs, B., Leino, K.R.: Boogie: A modular reusable verifier for object-oriented programs. In: Formal Methods for Components and Objects: 4th International Symposium. Volume 4111 of Lecture Notes in Computer Science., Springer (2006) 364–387
4. Moore, J.S.: Proving theorems about Java-like byte code. In Olderog, E.R., Steffen, B., eds.: Correct System Design. Volume 1710 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1999) 139–162
5. Moore, J.S., Porter, G.: The apprentice challenge. ACM Transactions on Programming Languages and Systems **24**(3) (May 2002) 193–216
6. Sutcliffe, G.: The TPTP problem library and associated infrastructure: The FOF and CNF parts, v3.5.0. Journal of Automated Reasoning **43**(4) (2009) 337–362
7. Sutcliffe, G., Suttner, C.: The state of CASC. AI Communications **19**(1) (2006) 35–48
8. Eastlund, C., Vaillancourt, D., Felleisen, M.: ACL2 for freshmen: First experiences. In: Proceedings of the 7th International Workshop on the ACL2 Theorem Prover and Its Applications. (November 2007)

9. Chamarthi, H.R., Dillinger, P., Manolios, P., Vroon, D.: The ACL2 sedan theorem proving system. In: Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Berlin, Heidelberg, Springer-Verlag (2011) 291–295
10. Bajaj, C., Khandelwal, S., Moore, J., Siddavanahalli, V.: Interactive symbolic visualization of semi-automatic theorem proving. Technical Report TR-03-37, University of Texas at Austin (August 2003)
11. Rager, D.L.: Parallelizing an Interactive Theorem Prover: Functional Programming and Proofs with ACL2. PhD thesis, University of Texas at Austin (December 2012)
12. Wos, L.: Automated reasoning - 33 basic research problems. Prentice Hall (1988)
13. Schumann, J.: Automated theorem proving in software engineering. Springer (2001)
14. Eggensperger, C.: Proof pad: A modern development environment for the ACL2 theorem prover. Master's thesis, The University of Oklahoma (May 2013)