Challenge Problems for the ACL2 Community

David Hardin
First, the good news...

ACL2 has been shown to scale to industrial problems
- Microprocessor verification
- Operating system kernel verification
- Verifying compiler
- Many more

The use of ACL2 has been accepted by certification authorities

The world is beginning to appreciate executable formal specifications

Security is being taken much more seriously by digital system designers

New techniques are leveraging ACL2’s proof automation and pushing it to new heights (depths?)
Some challenge problems that seem within reach

- Formally verified virtualization system for a commercially popular microprocessor
- Verified cross-domain systems
- Verified user mode networking stack
- Verified secure middleware
- Verified full JVM implementation
- Verified complex embedded real-time control systems
- Verifiable language system that would combine the best of Java, ML, Lisp, C#, etc., and that could take full advantage of modern multi-chip, multi-core computing systems
  - Including verified abstract data types
- “21st century CLInc stack”
Some challenges for ACL2 itself

- ACL2 should provide much better support for reasoning about “real-world” Lisp programs
- ACL2 still doesn’t know enough about computer arithmetic
- Integration with other tools – HOL connection is promising, but we need more
- Functional languages are inherently parallelizable, yet ACL2’s support for parallelism is limited
- Lisp Development Environments were cutting edge 20 years ago; now, they are way behind the times
- ACL2 is still too difficult for non-logicians to use; ACL2’s support for parallelism is limited
- Some problems are inherently higher order
So now, let’s look ahead 5 years...
Our intrepid formal methods guy, Guy, heads to work, driving a car with a formally verified engine control system. He can afford a nice car because he has profit sharing, and his employer makes lots of money on formal methods.
Guy downloads a parallel proof dispatch/visualization system released the night before by an Australian developer. The downloaded code is inspected by a bytecode verifier that has been proven correct.
Guy attends a design review for a security product prototype, based on a formally verified microprocessor design. The prototype is ready within weeks, and works as anticipated.
Over lunch, Guy has an idea on how to extend a previously verified product to a new domain. He realizes that he can incrementally verify the new functionality while reusing most of the existing proofs. He adds his new functionality to the architectural-level model, imports it into his proof system, and reverifies a key property. His employer is happy.
At the end of the day, Guy heads to the CHAIRS (Confluence of HOL, ACL2, Isabelle, and Refutation-based Systems) Workshop. At the airport, he checks out the spec for the V language, a formally verifiable language environment that is the hot new successor to Java/C++/C#/etc.
Meanwhile, a graduate student in New Mexico works on a massive verified V application in his dorm room along with other Internet-based developers. He has never freed live memory, suffered a buffer overflow attack, made a pointer arithmetic mistake, or had an undetected array bounds error.