How We Extended ACL2 to Verify Block Cipher Implementations
Eric Smith
ACL2 2009 Rump Session Talk

Abstract:
At FMCAD 2008, David Dill and I published the paper "Automatic Formal Verification of Block Cipher Implementations." I plan to quickly summarize that work for those who aren't familiar with it and then address in more depth some of the issues of particular interest to the ACL2 community.

While our verification approach was based on ACL2, we were unable to use ACL2's rewriter to perform the simplifications. The main reason is that ACL2's representation of terms as trees can be exponentially more expensive than an equivalent representation using directed acyclic graphs (DAGs) where shared subterms are represented only once. To perform the verification I had no choice but to write my own rewriter. It is written in ACL2 and applies ACL2 rewrite rules, but it represents terms compactly as DAGs. The rewriter can be used to symbolically simulate long JVM computations (tens of thousands of bytecodes) and to simplify the resulting large bit-vector and array terms (with tens of thousands of sub-expressions and massive sharing). ACL2's rewriter fails utterly on these examples.

I'll discuss which rewriter features I found necessary to implement (recursive rewriting of hypotheses, free variable matching from known assumptions, a version of syntaxp, a version of bind-free) and which I found I could live without, at least for our examples (type-prescription, forward chaining, linear reasoning, splitting into cases). I may also discuss performance issues (the use of ACL2 arrays and memoization) and touch on how such a rewriter might be verified.

Our block cipher verification method also relies on STP, an external, SAT-based decision procedure for bit-vectors and arrays. I'll briefly discuss the connection between ACL2 and STP, including the library of bit-vector and array functions that underlies our entire verification approach and that permits convenient translation to STP.