Verification of the Linux Keyboard Driver

My goal is to explore SMACK as a static analysis tool in the verification of an important Linux device driver: the keyboard. There are a few complex properties that are manifest in keyboard input that must be properly interpreted in order for the keyboard to function properly. Examples may include simultaneous key presses, key combinations, special functions based on key press duration, key remapping, multifunction keys, key combinations that are capable of generating interrupts, etc. I would like to explore how the code is verified with SMACK and what code conditions cause verification to fail.

My first step is to install and use SMACK to verify a simple “hello world” or similar program, and learn how to use SMACK and some of its features. My next goal would be to obtain copies of the source code for the Linux keyboard driver(s) and attempt to verify the code. If verification fails, I intend to analyze the cause(s) and attempt to identify code conditions that lead to failure. If verification succeeds, I intend to experiment with altering the code to cause failure and explore the verification process. My last step would be to organizing and interpreting the various results of verification attempts and reporting on interesting properties that were manifested.

There has been similar work done in the past with other verification tools. For example, one paper titled “**Sound formal verification of linux's USB BP keyboard driver”** was published in 2012 and can be found at <http://dl.acm.org/citation.cfm?id=2260747>. The tool used was VeriFast, and the authors reported that the driver was successfully verified. If verification with SMACK fails, I may attempt to alter SMACK so that verification may succeed.