On the Turing Completeness of MS PowerPoint

Tom Wildenhain

Introduction

As many users are well-aware, Microsoft PowerPoint ® offers unparalleled presentation editing tools, enabling the creation of professional, animation-laden slides with minimal effort (Source: Microsoft).[1] However, only more experienced (and desperate) PowerPoint aficionados fully utilize PowerPoint's advanced AutoShape, Hyperlink, and Transition tools for the purposes of image editing, video production, and game design. Given PowerPoint's versatility and cross-platform compatibility (offering Microsoft branded support for mobile devices and the two commercially relevant desktop operating systems), some have asked whether any other applications are necessary at all, or if all computational tasks can be accomplished through the creation of dedicated .pptx files. This research aims to definitively answer these questions in the affirmative through the creation of a PowerPoint Turing Machine (PPTXTM), proving PowerPoint to be exponentially more capable than competing slideshow editing software.

Creation Process

As the primary goal of this research is to prove the unnecessity of all non-PowerPoint software, no external applications were used in the creation of the TuringMachine.pptx file. Furthermore, VBScript and Macros were not used, as they have limited cross-platform support and are considered security threats, which may decrease the attractiveness of using .pptx files as an alternative to software. Thus, every AutoShape, Animation, and Hyperlink was painstakingly added by hand, which took a meager 10 hours due to PowerPoint's superior editing tools. Programming the finished TM file, however, takes minutes and can be done through an intuitive process.

Functionality and Operation

Like most Turing Machines, the PPTXTM file (available here[2]) consists of a tape, tape head, and states (instruction cards). Unlike most Turing Machines, they are made entirely of AutoShapes and On-Click Animations, enabling a more user friendly and visually pleasing Turing Machine experience (on just a single slide!). To program the file, the developer need only delete the correct portions of the instruction punch cards that determine the machine's

[1] https://products.office.com/en-us/powerpoint

[2] http://tomwildenhain.com/PowerPointTM/PowerPointTM.pptx

behavior based on its current state and the character on the tape. Once programmed, the PowerPoint application can be saved and distributed to interested users.[3]

To run a programmed Turing Machine, the user must open the file in the latest Microsoft-supported PowerPoint editor and enter slideshow mode.[4] They may then write the desired input on the tape by clicking the corresponding buttons. Clicking the run button begins the computation, with the machine starting at state 0. With each step, the PowerPoint highlights a region of the screen in vibrant, PowerPoint orange, which the user must click to continue execution. The user cannot click regions that are not highlighted, ensuring that the computation continues properly. When execution halts, the user can read the result from the tape. The PPTXTM can be easily modified to have separate accept/reject states based on the requirements of the application.

Turing Completeness

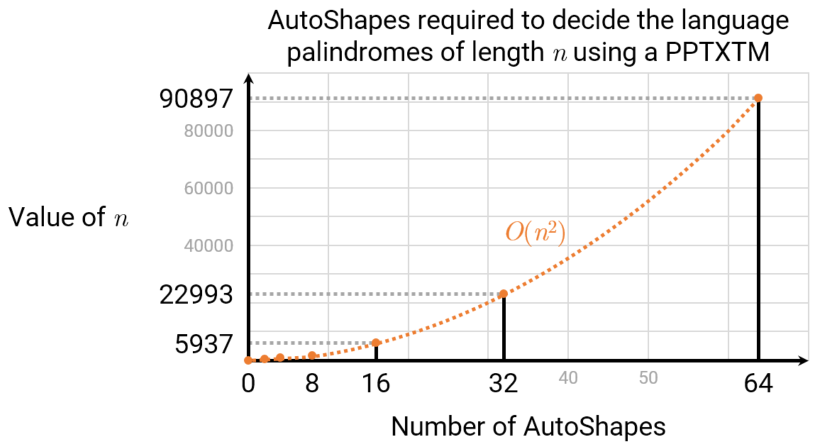
Critics of the PPTXTM may point out that the machine only has a finite-length tape and is therefore not a true Turing Machine. While this is true, it should also be noted that all physical systems have finite memory, and thus the capabilities of the PPTXTM are no less than that of any other Turing Complete language running on physical hardware. Furthermore, many other popular languages (like C) which are commonly referred to as Turing Complete actually use finitely sized pointers and therefore have bounded memory. But to truly understand what differentiates the PPTXTM from a less-capable PowerPoint Deterministic Finite State Automata (DFA) we must study the PPTXTM's Asymptotic AutoShape Complexity.

AutoShape and Animation Complexity

The 8 state PPTXTM with a tape alphabet of size four and 8 cells of memory requires 1669 animations and around 700 AutoShapes to function. More generally, for a given alphabet size a, a PPTXTM with n states and m tape cells uses $O(n^2 + m^2)$ animations and $O(nm)$ AutoShapes. In contrast, a PowerPoint DFA implemented using hyperlinks could require an exponential number of AutoShapes to achieve similar functionality. For example, a DFA to decide the language of palindromes of length at most m must memorize the entire first half of the string, which requires $a^{\frac{m}{2}}$ separate slides, each containing at least a AutoShapes. Thus, the PPTXTM is significantly more efficient than the DFA implementation. As most other (inferior) slideshow editing programs lack the On Click Animations PowerPoint offers, software presentations

[3] Assuming potential users of your product have PowerPoint or are interested enough in your application that they are motivated to obtain it.
[4] Running PPTXTMs in old or non-Microsoft approved slideshow viewers may lead to undefined behavior.

created using them must be implemented using hyperlinks making them exponentially larger than PPTXTMs with the same features.[5]



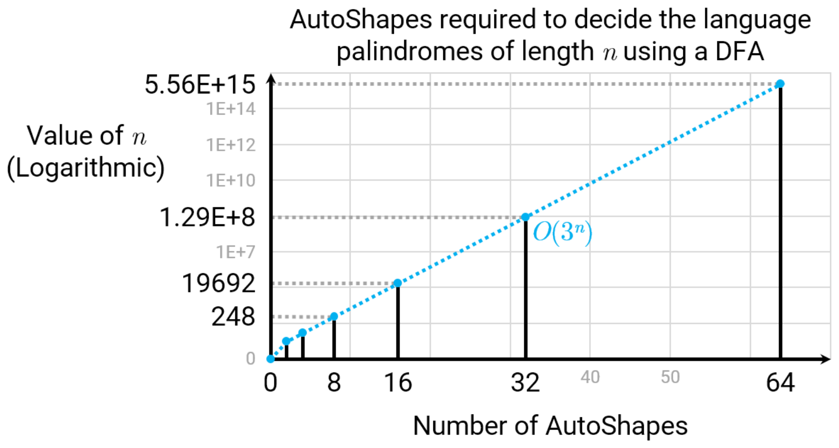


Figure 1. Graphs comparing the asymptotic AutoShape complexity of a PPTXTM with that of a DFA for deciding palindromes (alphabet size 3). These graphs were made using PowerPoint.

Advantages over Conventional Languages

The PPTXTM offers several advantages over other Turing Complete languages. Its ability to be programmed entirely using a GUI rather than confusing text-based languages could make it easier for novice programmers to learn. In addition, since execution requires the user to click before each step, no debugger is needed; stepping through code happens automatically. Installation of the IDE for PowerPoint development is painless and requires minimal setup; a credit card and Microsoft account are the only barriers to becoming an authentic PowerPoint

[5] https://discussions.apple.com/thread/6989563?start=0&tstart=0

developer.[6] PowerPoint's sandboxed "Protected View" means that PowerPoint applications can be safely shared and run, and since .pptx files are not generally viewed as dangerous, they can be easily downloaded on or emailed to any supported device. The PowerPoint IDE is also (surprisingly) stable when working with large projects; despite dealing with thousands of elements, the development of the PPTXTM file never crashed PowerPoint, though PowerPoint does appear to have some memory leaks when working with animations.[7] Of course the primary advantage of PowerPoint development is the ease with which aesthetically pleasing presentations can be created thanks to the built-in themes and styles.[8]

Implications of Turing Completeness

PowerPoint's ability to emulate arbitrary code, while offering many advantages, also has some less favorable repercussions, putting its app in violation of the iOS App Store Guidelines, which state that "Apps that create alternate desktop/home screen environments or simulate multi-app widget experiences will be rejected." [9] As proven through this research, PowerPoint files can emulate arbitrary applications and thus may be considered "apps," so the iOS PowerPoint app's open dialog might be considered a multi-app home screen environment. Furthermore, it is conceivable that an alternative App Store for iOS apps could be created which solely distributes executable PowerPoint files for every task. In fact, stores are already in existence for desktop-optimized PowerPoint applications, and it is just a matter of time before they begin to adopt Apple's platform.[10] Thus it is crucial that Microsoft act quickly and prevent execution of On Click Animations on iOS devices before its apps are removed for violating Apple's terms.

Future Work

While the PPTXTM proves the theoretical possibility of PowerPoint development, research needs to be done in making the software creation process more practical. I am currently investigating the issues of scalability and encapsulation, and have developed techniques for dividing complicated applications into multiple PowerPoint files that link to each other. Work also needs to be done in PowerPoint application optimization. There is a lot of potential here to exploit PowerPoint's automatic buffering of the next slide, which through careful slide

[6] Though some may consider "authentic PowerPoint developer" to be an oxymoron.
[7] It is recommended that you reopening your PowerPoint file each time you add more than 100 animations.
[8] Aesthetically displeasing presentations are equally possible:
http://www.pcworld.com/article/161912/powerpoint_hell_dont_let_this_happen_to_your_next_presentation.html
[9] Apple App Store Terms:
https://developer.apple.com/app-store/review/guidelines/
[10] An example of a PowerPoint app store:
http://people.uncw.edu/ertzbergerj/ppt_games.html

placement may be used to greatly increase application performance. With enough advances in these areas, it is increasingly likely that every application will one day be run within Microsoft PowerPoint.

This document was typeset in PowerPoint.