UNIVERSITY OF GLASGOW

Safety Critical Systems AX Report (H)

Andrei-Mihai Nicolae
2147392n@student.gla.ac.uk

March 3, 2017

CONTENTS

1 Introduction

Artificial Intelligence has been one of the hot topics of this decade as it is surrounding our lives more as days go by. We can see it in objects ranging from aircrafts to smart home controllers.

However, as AI has become so powerful, it may come with a cost for the majority of us. Because of the large number of risks when letting Artificial Intelligence and Machine Learning robots/devices perform certain actions for us (Mannino [2015]), we need to devote a huge amount of effort in designing better safety-oriented architectures, well-crafted and thorough testing as well as regular check-ups and revisions. Therefore, we need to see a considerable increase in tools that assess and mitigate risks when introducing AI into systems of any type. Such a tool will be discussed in further detail in this report.

1.1 AI-Driven Technologies

The importance of introducing AI into a field with safety as a top priority is crucial. Because there is no human involved, the goal of the whole AI community is to let the machines actually take our place in performing certain actions, so that our tasks would be simplified. As good examples where it has become more and more developed, here is a list of examples:

- Transportation (driverless cars, subways)
- Game playing machines (Deepmind's Go playing machine that beat the en-titre champion) Silver [2016]
- Medical robotics
- Manufacturing machines
- Education

As the list can go on, we can see how AI spans throughout most of the major aspects of our lives, thus the need of careful monitoring its development.

1.2 How AI Can Go Wrong

Going past the many fields driven by Artificial Intelligence nowadays, we need to also take a close look at how many technologies have proven to be very prone to failure.

One interesting case is something that happened only a few months ago with an Uber driverless car going through a red light in front of San Francisco's Museum of Modern Art (Wakabayashi [2015]).
As it was recorded on camera, the car just rushes through a busy street on red light. This is a clear sign of how developers are not placing enough testing and robust checks before launching such a safety critical systems into an open environment.

We can see that the cause of the previous example could have been harmful for us humans. However, there have been cases where AI was involved and it was even deadly. Such an event is the killing of a Volkswagen employee who was grabbed and killed by one of the manufacturing robots in the plant (Dockterman [2015]). Moreover, a robot in one Silicon Valley mall struck a child on the head by mistake (Rocha [2016]).

In conclusion, after discussing various developments in the Artificial Intelligence world and how these safety critical systems can fail drastically, a tool for assessing and mitigating such risks will be presented in the rest of this report.

2 Tool

After extensive research, I came to the conclusion that for such an application the best technique that should be used is Model Checking adapted specifically for AI systems.

2.1 Reasons behind Choice

Firstly, it's needed to be pointed out that various techniques (e.g. Fault Tree Analysis, Effects and Criticality Analysis) work as well for some sub-fields of AI-driven systems, but I believe that Model Checking is eventually the optimal choice. This is because of various reasons which will be exposed below.

2.2 AI is Young

Because of the relatively young age of AI and ML in mainstream technology, there is quite a significant amount of improvements needed to be implemented. As shown before, there are many cases where they failed and lead to possible catastrophic outcomes.

Model Checking is a solution to the problem: having a model system, check automatically (and exhaustively) if that particular system meets some given specification. I believe this is a very good approach to the tool needed to be implemented because the model checking applies to finite state systems (Wah [2009]). As AI is young, we need to make sure (even at the cost of time and not so optimal design decisions) that the system behaves correctly and gives the correct output regardless of the situation.
As such, a technique such as Model Checking would automatically check all possible combinations and determine whether the system is prepared to use AI/ML safely or not.

2.3 State of the Art Usages of the Model

There are many cases in the technology world where manufacturers and government bodies adopted Model Checking for AI-driven systems, thus making me more inclined in having confidence in the technique's effectiveness.

One such example is how NASA used a SPIN-based Model Checker to verify the safety properties of autonomous planning systems. In their case, such an APS would decide what a rover or spacecraft should perform next (Cucullu [2005]). Therefore, they needed to assure that the APS would always pick a safe choice that would not require human supervision eventually. Therefore, they came up with a solution that would replace the current empirical testing at that time (proven inefficient) with a Model Checker SPIN implementation using the Promela programming language. It proved to be very successful, being able to develop over 3 billion plans for their finite-state system. Even though this is very time-consuming and it is not optimal, as anyone can imagine, the decision a rover or a spacecraft system makes is of crucial importance, thus we can sacrifice cost for the benefit of confidence in the machine's abilities to perform the task correctly.

Another good example is coming from the aviation field: Rockwell Collins is an important avionics and IT company that delivers safety critical systems to governmental agencies and various aircraft manufacturers. They developed 2 systems of interest to us in this report: one is called ADGS-2100 Adaptive Display and Guidance System Window Manager - it was designed to provide pilots with heads up and down displays, as well as display management software, while the other one is FCS 5000 - a new family of Flight Control Systems.

Figure 2.1: Pilot Display Panels

After considering various techniques to assess and mitigate risk as early and efficiently as possible, they ended up with using Model Checking (Miller [2005]) for both applications. The systems are crucial, such that whichever would fail during a flight it would impose great risks to both pilots and other travellers. Therefore, they have imposed a Simulink specification us-

ing only Boolean and enum types (Miller [2010]). The company has been very pleased with the approach, having 5 autonomous components which had, in turn, 563 properties developed and checked altogether. This revealed around 100 errors in early versions of the window manager. By the end of the project, however, the developers were checking properties after the design change, reporting that they were very satisfied with the new technique for assessing and mitigating risk.

2.4 Available Tools

After showing how the tool is used by large companies and governmental agencies throughout the globe, there is a need to discuss about available tools to get started when applying the Model Checking technique. Because such systems where AI is becoming more involved are taking over the technology field, people need fast, reliable and easy to get started tools.
Such an example is the above-mentioned SPIN.

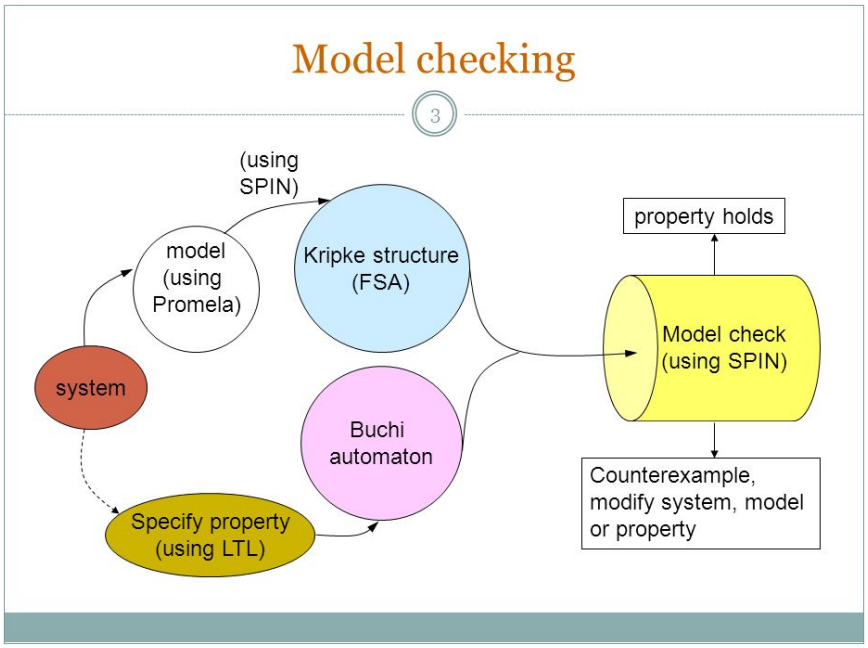

Figure 2.2: SPIN Model Checking

Not only that the model received the extremely prestigious ACM System Software Award in 2001, but it also has annual conferences held specifically for this technique. The systems need to be described in Promela language, while the properties that need to be verified are expressed as LTL (Linear Temporal Logic) formulas (Holzmann [2003]).

Apart from SPIN, NASA also made its verification tool using the Model Checking technique for the highly popular programming language Java.

Coming to a conclusion, recent studies and developments show that there are many tools available out there that can make a developer's job to put a Model Checking technique into practice quite easy.

2.5 Case Study

A specific case study that I came up with is

2.6 Implementation of the Tool

The idea I came up with was easy to implement and very efficient. As web tools are gaining more popularity among developers as days go by, the Electron framework developed by GitHub was a choice that stood out (GitHub [2017]).

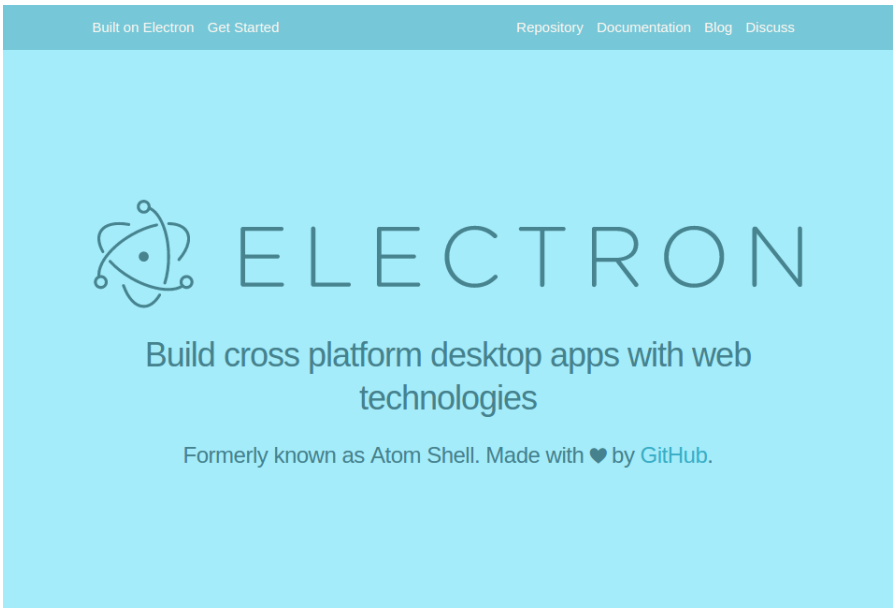

Figure 2.3: GitHub's Electron Framework

Therefore, I used only web languages (JavaScript, HTML, CSS) to build a native application on all three major OS families (macOS, Windows, Linux). This was a very important design choice because I believe that websites are not such a good choice for assessing and mitigating risks in safety critical systems. This is because of a variety of reasons:

- Security: having an offline application that cannot (if developed so) require an Internet connection eliminates the risks of man-in-the-middle attacks and many other techniques that might threaten the checking process and valuable information
- Familiarity: having an app that performs natively regardless of the specific operating system will make the developers "feel at home", becoming more productive

- System calls: the developers can take benefit from the system calls that can be used from within Electron
- Reduced costs: as websites require server-side power and hosting, they are not a valuable option for such a tool because there would be huge costs (before we mentioned how NASA had more than 3 billion plans when running the Model Checking); however, with a standalone app, we can reduce the costs drastically as we can include the whole "server" inside the app and the local machine's resources would be used to power up the huge number of plans

As we showed why a website wouldn't have been such a good idea for implementing the tool, we also need to take into consideration the limited time developers have. What I mean by that is developers can just build an app with web tools once and have it on 3 platforms from a single shot. This gives them the time that could have been wasted on building 3 native platforms with 3 different frameworks (that also would have required prior research and training).

In terms of using SPIN in my application, I have created the Electron app to give the user a visual interface where he/she could press buttons, customise variables and afterwards Promela models would be created and test systems that would be selected from local storage by the developer. My app is in a very raw form for demonstration purposes as it was not required to build a fully fledged app.

In terms of risk assessment warnings, I have selected a colour palette that would make the developer pay attention regarding the current status: vivid red, orange and green colours would denote various levels of risk. Furthermore, I have also spent extensive research into automating my app and I have found a very interesting paper (Yamada [2011]). I believe that it sounds promising and it would bring great performance and overall improved user experience to the tool.

Taking all these into account, I believe that the electronic tool I developed needs to be extended and further developer to become a viable option for software engineers mitigating and assessing risk, but this is a start.

3 Evaluation

3.1 Evaluation Technique

Talk about what evaluation techniques were used and why were they effective for this particular case.

3.2 Iterations Progress

Talk about how the soft. dev. lifecycle went on as people were evaluated and how all steps were followed thoroughly.

4 Results

5 Conclusion

References

Gordon Cucullu. Verifying autonomous planning systems: Even the best laid plans must be verified. *IEEE*, 0-4(5), 2005.

Eliana Dockterman. Robot kills man at volkswagen plant. 2015. URL http://time.com/3944181/robot-kills-man-volkswagen-plant/.

GitHub. Electron framework, 2017. URL https://electron.atom.io.

Gerard Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison Wesley, 2003.

Adriano Mannino. Artificial intelligence: Opportunities and risks. 2015. URL https://foundational-research.org/wp-content/uploads/2016/06/AI-Policy-Paper.pdf.

Steven P. Miller. Formal verification of flight critical software, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.6364rep=rep1type=pdf.

Steven P. Miller. Software model checking takes off. *ACM*, 53(2):58–64, 2010.

Veronica Rocha. Crime-fighting robot hits, rolls over child at silicon valley mall, 2016. URL http://www.latimes.com/local/lanow/la-me-ln-crimefighting-robot-hurts-child-bay-area-201

David Silver. Mastering the game of go with deep neural networks and tree search, 2016. URL https://gogameguru.com/i/2016/03/deepmind-mastering-go.pdf.

Benjamin W. Wah. *Wiley Encyclopedia of Computer Science and Engineering*. 2009.

Daisuke Wakabayashi. A lawsuit against uber highlights the rush to conquer driverless cars. 2015. URL https://www.nytimes.com/2017/02/24/technology/anthony-levandowski-waymo-uber-goo

Yutaka Yamada. Automatic generation of spin model checking code from uml activity diagram and its application to web application design. *IEEE*, 53, 2011.