The Vehicle Tutorial: Neural Network Verification with Vehicle

Matthew L. Daggitt[1], Wen Kokke[3], Ekaterina Komendantskaya[1], Robert Atkey[3], Luca Arnaboldi[2]*, Natalia Slusarz[1], Marco Casadio[1], Ben Coke[1], and Jeonghyeon Lee[1]

[1] Heriot-Watt University, Edinburgh, UK
{md2006,ek19,nds1,mc248,bc90,jl2038}@hw.ac.uk
[2] University of Birmingham, Birmingham, UK
l.arnaboldi@bham.ac.uk
[3] University of Strathclyde
{wen.kokke,robert.atkey}@strath.ac.uk

Abstract

Machine learning components, such as neural networks, gradually make their way into software; and, when the software is critically safe, the machine learning components must be verifiably safe. This gives rise to the problem of neural network verification. The community has been making rapid progress in developing methods for incorporating logical specifications into neural networks, both in training and verification. However, to truly unlock the ability to verify real-world neural network-enhanced systems we believe the following is necessary:

1. The specification should be written once and should automatically work with training and verification tools. 2. The specification should be written in a manner independent of any particular neural network training/inference platform. 3. The specification should be able to be written as a high-level property over the problem space, rather than a property over the input space (of the neural network). 4. After verification the specification should be exportable to general interactive theorem provers so that its proof can be incorporated into proofs about the larger systems around the neural network.

In this tutorial we presented Vehicle, a tool that allows users to do all of this. We provided an introduction to the Vehicle specification language, and then walked attendees through using it to express a variety of famous and not-so-famous specifications. Subsequently we demonstrate how a specification can be compiled down to i) queries for network verifiers, ii) Tensorflow graphs to be used as loss functions during training and iii) cross-compiled to Agda, a main-stream interactive theorem prover. Finally we discussed some of the technical challenges in the implementation as well as some of the outstanding problems.

Keywords: Programming Languages, Neural Network Verification, Adversarial Training, Types, Domain Specific Languages.

1 What is Vehicle?

Vehicle is a language for writing logical specifications for neural networks. At its heart is the *Vehicle specification language*, a high-level, functional language designed for writing mathematically precise specifications for neural networks networks.

For example, below is the start of a Vehicle specification for the famous ACAS Xu [9] verification challenge. This code snippet[1] declares the types of inputs and outputs of the neural network, as well as the type of the neural network itself:

*Large portion of work undertaken whilst at University of Edinburgh

[1]The full Vehicle specification for ACAS Xu is given in [5, 6, 7].

```
type InputVector = Vector Rat 5
type OutputVector = Vector Rat 5

@network
acasXu : InputVector -> OutputVector
```

Vehicle type checker ensures type safety of specifications. Relevant error messages will be produced if, for example, the code includes invalid vector indexing or uses input and output vectors that do not match the network type. The Vehicle language also makes it easy to define other auxiliary properties that ensure soundness of specifications, such as validity of inputs for a neural network, assuming the minimum and maximum permissible input values:

```
validInput : InputVector -> Bool
validInput x = forall i .
  minimumInputValues ! i <= x ! i <= maximumInputValues ! i
```

Because one can safely compose definitions of functions, even difficult specifications can be broken down to simpler subparts. For example, the below definition of ACAS Xu Property 3 (*"If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal."*) relies on the definition of the valid input we provided above:

```
@property
property3 : Bool
property3 = forall x . validInput x and
                       directlyAhead x and
                       movingTowards x =>
  not (advises clearOfConflict x)
```

Similarly, the definitions of `directlyAhead`, `movingTowards` and `clearOfConflict` are also given in this compositional manner.

There are several scenarios for Vehicle usage once the specification is written and type-checked. The specification can be automatically translated into the VNNLiB format [1] and used for verification with one of the existing solvers, such as Marabou [10]. The interested reader can find more about the type system underlying this part of Vehicle in [3].

It has been observed in the literature that neural network verification often has to include property-driven training as its integral part [11, 8, 12, 13]. Therefore, the second usage scenario for the specification is to generate a loss function for re-training the neural network in Python Tensorflow. Finally, a Vehicle specification can be automatically converted into Agda code. These different usage scenarios are shown in Figure 1, and were discussed in detail in the course of the Vehicle tutorial at FOMLAS. Currently, adversarial example search (for given a property) is not directly enabled, but is involved in training with loss functions generated by Vehicle.

2 The Vehicle Code and Tutorial Materials

Vehicle's entire code is publically available at [5]. Figure 2 shows the overall architecture of the Vehicle as a domain-specific language for neural network verification. As of recently, the Vehicle installation has been simplified to

```
pip install vehicle-lang
pip install maraboupy
```

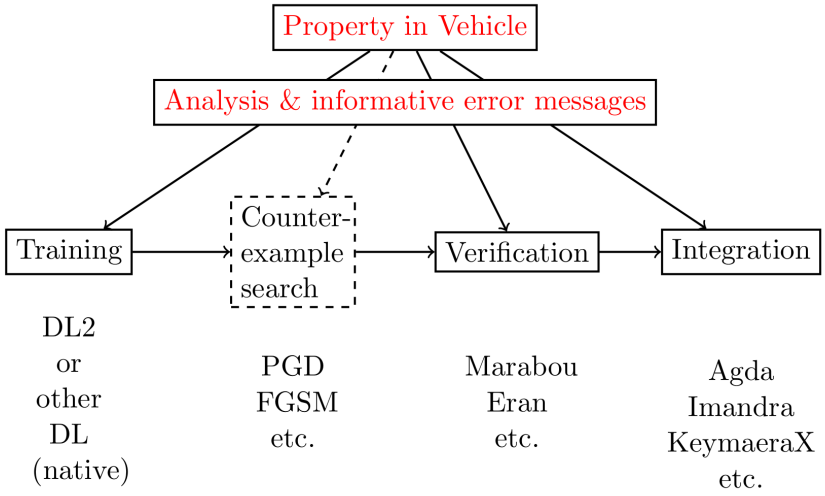

Figure 1: *A high-level overview of existing Vehicle backends.*

The second line additionally installs the verifier Marabou used by Vehicle as a Backend. Thus, only the Vehicle developers are working directly with the sources in [5].

The Complete Manual detailing the language and the backends is available in [4].

The Tutorial materials, available in html format in [6] complement the Vehicle Manual in several ways:

- Chapter 1 of the Tutorial gives a high-level discussion of the area of neural network verification and of different tools and approaches that exist in the literature, placing Vehicle as a unifying high-level language over several other existing tools;
- Chapter 2 introduces the reader to the basic syntax and command line interface of Vehicle, by means of the ACAS Xu [9] example;
- Chapter 3 discusses the popular verification problem of neural network robustness [2], and *inter alia* introduces further details of Vehicle's syntax and command line interface.
- Chapter 4 focusses on *Differentiable Logics* [13], as means of generating loss functions in Vehicle. It also explains Vehicle's interface with Tensorflow.
- Chapter 5 explains Vehicle's integration with Agda.

The theoretical part of the tutorial, given in the above Chapters, is complemented by the *Practical Exercises*. In particlar the Exercise Repository [7] contains:

- the code and networks that were used as examples within the tutorial chapters [6], as well as
- a directory with exercises offered for self-study to Vehicle tutorial attendees. The exercise directory contains specific property prototyping tasks, coming packaged with trained neural networks in ONNX format, data in IDX format and, where appropriate (semi-complete) Vehicle specifications. There is a link to relevant directory at the end of each chapter of the Vehicle tutorial. Most of the exercises come with model answers.

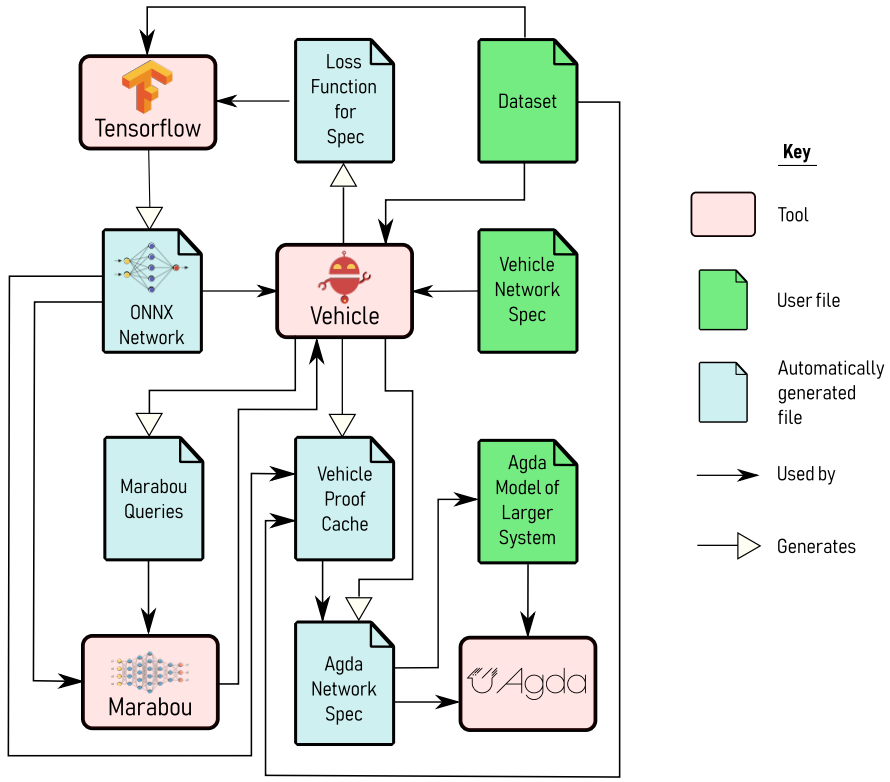

Figure 2: *Vehicle architecture.*

Some additional exercises in the exercise directory were created by Heriot-Watt university MSc students Ben Coke and Jeonghyeon Lee, as part of their MSc dissertations.

3 The Tutorial Structure at FOMLAS'23

In recognition that FOMLAS audience is generally well-acquainted with the methodology underlying neural network verification, the live FOMLAS tutorial started with detailed exposition of material in Chapters 1 and 2 (Introduction and Vehicle Syntax), however the technical content of Chapter 2 (Robustness Verification) was proposed for a practical exercise session. The practical session was followed by a live demo that covered material in Chapters 4 and 5.

4 Acknowledgements

The work was supported by the EPSRC grant EP/T026952/1, EP/T026960/1, EP/T027037/1 : *AISEC: AI Secure and Explainable by Construction.* We thank FOMLAS organisers: Guy Amir, Omri Isac, Guy Katz, Nina Narodytska for providing a venue for this tutorial.

References

[1] VNNLib format, https://vnnlib.org/, accessed on 01.12.2021

[2] Casadio, M., Komendantskaya, E., Daggitt, M.L., Kokke, W., Katz, G., Amir, G., Refaeli, I.: Neural network robustness as a verification property: A principled case study. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13371, pp. 219–231. Springer (2022)

[3] Daggitt, M.L., Atkey, R., Kokke, W., Komendantskaya, E., Arnaboldi, L.: Compiling higher-order specifications to SMT solvers: How to deal with rejection constructively. In: Krebbers, R., Traytel, D., Pientka, B., Zdancewic, S. (eds.) Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023. pp. 102–120. ACM (2023). https://doi.org/10.1145/3573105.3575674, https://doi.org/10.1145/3573105.3575674

[4] Daggitt, M.L., Kokke, W.: Vehicle Complete Documentation (2023), https://vehicle-lang.readthedocs.io/, accessed on 01.08.2023

[5] Daggitt, M.L., Kokke, W., Atkey, B., Ślusarz, N., Casadio, M.: Vehicle: the Developer Repository (2023), https://github.com/vehicle-lang/vehicle, accessed on 01.08.2023

[6] Daggitt, M.L., Kokke, W., Komendantskaya, E.: The Vehicle Tutorial (2023), https://vehicle-lang.github.io/tutorial/, accessed on 01.08.2023

[7] Daggitt, M.L., Kokke, W., Komendantskaya, E., Atkey, R., Arnaboldi, L., Slusarz, N., Casadio, M., Coke, B., Lee, J.: The Vehicle Tutorial: Repository with Practical Exercises (2023), https://github.com/vehicle-lang/tutorial, accessed on 01.08.2023

[8] Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., Vechev, M.T.: DL2: training and querying neural networks with logic. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. vol. 97, pp. 1931–1941. PMLR (2019)

[9] Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)

[10] Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The Marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)

[11] Komendantskaya, E., Kokke, W., Kienitz, D.: Continuous verification of machine learning: a declarative programming approach. In: PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020. pp. 1:1–1:3. ACM (2020). https://doi.org/10.1145/3414080.3414081, https://doi.org/10.1145/3414080.3414081

[12] van Krieken, E., Acar, E., van Harmelen, F.: Analyzing differentiable fuzzy logic operators. Artif. Intell. **302**, 103602 (2022)

[13] Slusarz, N., Komendantskaya, E., Daggitt, M.L., Stewart, R.J., Stark, K.: Logic of differentiable logics: Towards a uniform semantics of DL. In: LPAR-24: The International Conference on Logic for Programming, Artificial Intelligence and Reasoning (2023)