WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
MASTER STUDY PROGRAM: SOFTWARE ENGINEERING

Verification of Neural Networks

Supervisor:
Mădălina Erașcu

Students:
Amalia Duma Diana
Răzvan Maciovan Alexandru
Luca Mitroi
Vlad Petcu
Vlad Țîrcomnicu

TIMIŞOARA
2023
Github Link

Contents

Chapter 1

Benchmark

The assignments for this project, required the usage of one benchmark and two tools for which to analyze upon. For the purpose of our project, we have decided to utilize the ACASXU benchmark[1], along with the tools Alpha Beta Crown[2] and Marabou [3]

1.1 Acasxu

Acas stands for Airborne Collision Avoidance System. There are multiple types of Acas benchmarks, but the one this paper is based on is Xu, which is optimized for unmanned aircraft systems (UAS), issuing turn rate advisories to remote pilots[Katz et al., 2017]. The installation process for the benchmark was easy, only needing to download it from GitHub[1]. The files are divided into 3 components as followed:

Onnx

These files contain the neural network models encoded in the Open Neural Network Exchange (ONNX) format[ONNX Contributors, 2023]. They collectively are the core architecture for the "brain" of the benchmark, which dictates the verification and evaluation steps.

Vnnlib

These files contain specifications and properties that need to be verified or analyzed for the neural network models (ONNX). They contain a similar syntax to that of z3 smt-solvers.

[1]`https://github.com/ChristopherBrix/vnncomp2023_benchmarks/tree/main/benchmarks/acasxu`

[2]`https://github.com/Verified-Intelligence/alpha-beta-CROWN`

[3]`https://github.com/NeuralNetworkVerification/Marabou`

Instances.csv

This CSV file bundles the onnx and vnnlib files into groups, where vnnlib files are associated with onnx ones. There are 10 vnnlib files and 45 onnx files, and they are combined to form a total of 186 combinations. Each combination also has metadata containing how long can the selected tool run one specific combination.

The input data that can be found across the VNNLIB files, revolves around different aircraft parameters determined from sensor measurements[Kochenderfer and Chryssanthacopoulos, 2011]:

- Distance from ownership to intruder;
- Angle to intruder relative to aircraft heading direction length;
- Heading angle of intruder relative to aircraft heading direction;
- Speed of aircraft;
- Speed of intruder;
- Time until loss of vertical separation;
- Previous advisory;

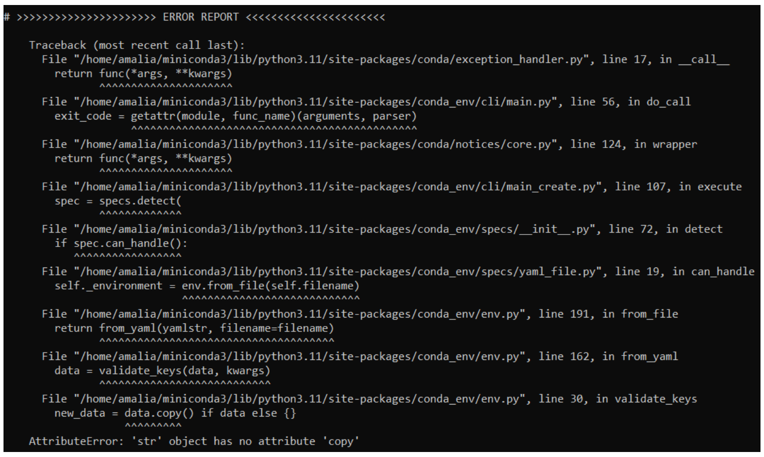
Five outputs represent the different horizontal advisories provided to the aircraft: Clear-of-Conflict (COC), weak right, strong right, weak left, or strong left. These values inputted into the neural network (the ONNX files), representing a combination of 45 deep neural networks created by mixing the time until loss of vertical separation and the previous advisory. The DNNs are fully connected, use ReLU activation functions, and have six hidden layers with a total of 300 ReLU nodes each. [Katz et al., 2017]

Chapter 2

Alpha-Beta-CROWN installation

Alpha Beta Crown serves as a neural network verifier based on an efficient linear bound propagation framework and branch and bound [huanzhang12, 2021]. Setting it up involved a two-step process: first, installing Miniconda and then the tool itself. There were no complications in the first part, as there is an executable file available on their official site that takes care of the entire setup once run. However, the latter part, involving the tool's installation, presented numerous complications.

The setup of Alpha Beta Crown followed their "Installation and Setup" section outlined on GitHub. The first command was supposed to clone their GitHub repository including "auto_LiRPA", an essential component for the tool's functionality. Despite running this command, only the tool's files were retrieved due to permission denials that restricted access to the second GitHub repository. Resolving this required an additional step: manually obtaining the missing necessary files and placing them in their indicated location. The next phase involved creating the conda environment using the specified command, which led to an error that can be observed in figure 2.1.



```
# >>>>>>>>>>>>>>>>>>>>>> ERROR REPORT <<<<<<<<<<<<<<<<<<<<<<

    Traceback (most recent call last):
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda/exception_handler.py", line 17, in __call__
        return func(*args, **kwargs)
               ^^^^^^^^^^^^^^^^^^^^^
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda_env/cli/main.py", line 56, in do_call
        exit_code = getattr(module, func_name)(arguments, parser)
                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda/notices/core.py", line 124, in wrapper
        return func(*args, **kwargs)
               ^^^^^^^^^^^^^^^^^^^^^
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda_env/cli/main_create.py", line 107, in execute
        spec = specs.detect(
               ^^^^^^^^^^^^^
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda_env/specs/__init__.py", line 72, in detect
        if spec.can_handle():
           ^^^^^^^^^^^^^^^^^
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda_env/specs/yaml_file.py", line 19, in can_handle
        self._environment = env.from_file(self.filename)
                            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda_env/env.py", line 191, in from_file
        return from_yaml(yamlstr, filename=filename)
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda_env/env.py", line 162, in from_yaml
        data = validate_keys(data, kwargs)
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^
      File "/home/amalia/miniconda3/lib/python3.11/site-packages/conda_env/env.py", line 30, in validate_keys
        new_data = data.copy() if data else {}
                   ^^^^^^^^^
    AttributeError: 'str' object has no attribute 'copy'
```

Figure 2.1: Creating conda environment error

This error consumed a considerable amount of time, as the error message initially suggested an issue with the script. After extensive investigation, the root cause was traced to the configuration file. Instead of containing the necessary configuration for the conda environment, it referenced another configuration file by name. After resolving the configuration file issue, the installation of the tool seemed successful.

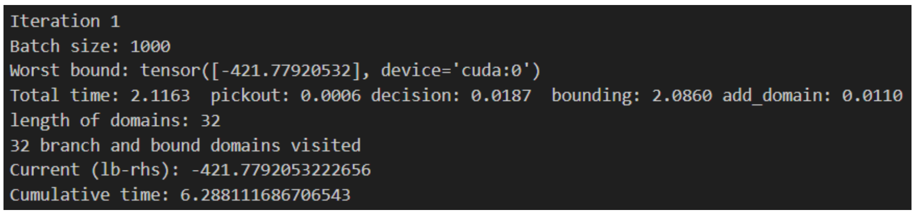
However, when attempting to run it, another problem emerged indicating a missing library, ”auto_LiRPA”. Although the necessary files existed, the error message revealed a discrepancy in the file path. It seemed that the correct location for this library was wrongly specified. With this adjustment, the setup concluded, and the tool run successfully.

Chapter 3

Acasxu verification

To ensure the reliability and safety of Acasxu, a verification was employed using Alpha-Beta-CROWN. This tool relies on data from two files: a VNNLIB file outlining the property to be verified and a file containing the neural network in an ONNX format. Its goal is to test whether the specified property holds true for the given neural network and input constraints.

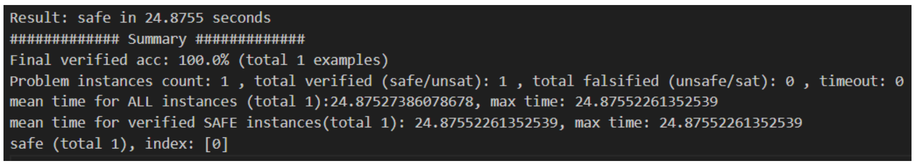
Initially, the testing focused solely on property 1 to gain insights into the output. This output contains details about configuration, attack parameters, model output, properties, verification iterations, results, and a summary. Every iteration provides data about the batch size, the worst lower bound encountered, the total time spent in that iteration, the number of domains or branches explored, the cumulative time and the current lower bound - right-hand side value. This particular value is used to asses whether the property is satisfied or not, since a negative value indicates a potential violation of the property. One example of such iteration can be observed in figure 3.1.



```
Iteration 1
Batch size: 1000
Worst bound: tensor([-421.77920532], device='cuda:0')
Total time: 2.1163  pickout: 0.0006 decision: 0.0187  bounding: 2.0860 add_domain: 0.0110
length of domains: 32
32 branch and bound domains visited
Current (lb-rhs): -421.7792053222656
Cumulative time: 6.288111686706543
```

Figure 3.1: Iteration 1

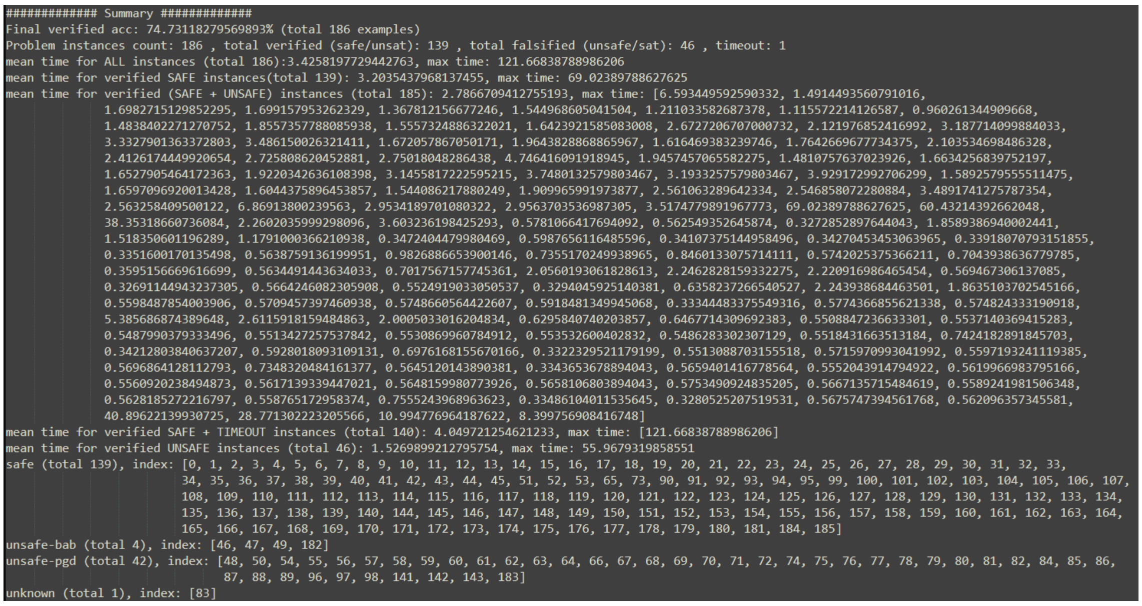
The summary provides an overview of the verification progress. In the case of the first property tested, the verification concluded in 24.8 seconds. Its summary can be seen in figure 3.2. The tool successfully checked that no safety violations were found within the specified property constraints with an accuracy of 100% and a mean time of approximately 24.87 seconds.



```
Result: safe in 24.8755 seconds
############# Summary #############
Final verified acc: 100.0% (total 1 examples)
Problem instances count: 1 , total verified (safe/unsat): 1 , total falsified (unsafe/sat): 0 , timeout: 0
mean time for ALL instances (total 1):24.87527386078678, max time: 24.87552261352539
mean time for verified SAFE instances(total 1): 24.87552261352539, max time: 24.87552261352539
safe (total 1), index: [0]
```

Figure 3.2: Summary for property 1

The next phase involved testing all properties, aiming to evaluate the neural network's behavior across various configurations and constraints. The final verified accuracy stands at approximately 74.73%. Out of the 186 instances tested, 139 were verified as safe, meaning that the property held true. There was 1 instance that resulted in a timeout, indicating an inconclusive verification process. In terms of verification times, the mean time taken for verifying all instances was approximately 3.43 seconds, with the maximum time being around 121.67 seconds. Regarding the branch and bound method (BAB), 4 instances were marked as unsafe, whereas in the context of the Projected Gradient Descent (PGD) method, 42 instances were identified as unsafe. Additionally, one instance fell into an unknown category, likely meaning it couldn't be clearly classified as safe or unsafe. All of this information, along with more detailed insights, can be observed in figure 3.3.



```
############# Summary #############
Final verified acc: 74.73118279569893% (total 186 examples)
Problem instances count: 186 , total verified (safe/unsat): 139 , total falsified (unsafe/sat): 46 , timeout: 1
mean time for ALL instances (total 186):3.4258197729442763, max time: 121.66838788986206
mean time for verified SAFE instances(total 139): 3.2035437968137455, max time: 69.02389788627625
mean time for verified (SAFE + UNSAFE) instances (total 185): 2.7866709412755193, max time: [6.593449592590332, 1.4914493560791016,
            1.6982715129852295, 1.699157953262329, 1.367812156677246, 1.544968605041504, 1.211033582687378, 1.115572214126587, 0.960261344909668,
            1.4838402271270752, 1.8557357788085938, 1.5557324886322021, 1.6423921585083008, 2.6727206707000732, 2.121976852416992, 3.18771409984033,
            3.3327901363372803, 3.486150026321411, 1.672057867050171, 1.9643828868865967, 1.616469383239746, 1.7642669677734375, 2.103534698486328,
            2.4126174449920654, 2.725808620452881, 2.75018048286438, 4.746416091918945, 1.9457457065582275, 1.4810757637023926, 1.6634256839752197,
            1.6527905464172363, 1.9220342636108398, 3.1455817222595215, 3.7480132579803467, 3.1933257579803467, 3.929172992706299, 1.5892579555511475,
            1.6597096920013428, 1.6044375896453857, 1.544086217880249, 1.909965991973877, 2.561063289642334, 2.546858072280884, 3.4891741275787354,
            2.563258409500122, 6.86913800239563, 2.9534189701080322, 2.9563703536987305, 3.5174779891967773, 69.02389788627625, 60.4321439266048,
            38.35318660736084, 2.2602035999298096, 3.603236198425293, 0.5781066417694092, 0.562549352645874, 0.3272852897644043, 1.8589386940002441,
            1.518350601196289, 1.1791000366210938, 0.3472404479980469, 0.5987656116485596, 0.34107375144958496, 0.34270453453063965, 0.33918070793151855,
            0.3351600170135498, 0.5638759136199951, 0.9826886653900146, 0.7355170249938965, 0.8460133075714111, 0.5742025375366211, 0.7043938636779785,
            0.3595156669616699, 0.5634491443634033, 0.7017567157745361, 2.0560193061828613, 2.2462828159332275, 2.220916986465454, 0.569467306137085,
            0.32691144943237305, 0.5664246082305908, 0.5524919033050537, 0.3294045925140381, 0.6358237266540527, 2.243938684463501, 1.863510370254516,
            0.5598487854003906, 0.5709457397460938, 0.5748660564422607, 0.5918481349945068, 0.33344483375549316, 0.5774366855621338, 0.574824333190918,
            5.385686874389648, 2.6115918159484863, 2.0005033016204834, 0.6295840740203857, 0.6467714309692383, 0.5508847236633301, 0.5537140369415283,
            0.5487990379333496, 0.5513427257537842, 0.5530869960784912, 0.553532600402832, 0.5486283302307129, 0.5518431663513184, 0.742418289184570,
            0.34212803840637207, 0.5928018093109131, 0.6976168155670166, 0.3322329521179199, 0.5513088703155518, 0.5715970993041992, 0.5597193241119385,
            0.569686412811279, 0.7348320484161377, 0.5645120143890381, 0.3343653678894043, 0.5659401416778564, 0.5552043914794922, 0.5619966983795166,
            0.556092023849487, 0.5617139339447021, 0.5648159980773926, 0.5658106803894043, 0.5753490924835205, 0.5667135715484619, 0.5589241981506348,
            0.5628185272216797, 0.558765172958374, 0.7555243968963623, 0.33486104011535645, 0.3280525207519531, 0.5675747394561768, 0.562096357345581,
            40.89622139930725, 28.771302223205566, 10.994776964187622, 8.399756908416748]
mean time for verified SAFE + TIMEOUT instances (total 140): 4.049721254621233, max time: [121.66838788986206]
mean time for verified UNSAFE instances (total 46): 1.5269899212795754, max time: 55.9679319858551
safe (total 139), index: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
                         34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 51, 52, 53, 65, 73, 90, 91, 92, 93, 94, 95, 99, 100, 101, 102, 103, 104, 105, 106, 107,
                         108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134,
                         135, 136, 137, 138, 139, 140, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
                         165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 184, 185]
unsafe-bab (total 4), index: [46, 47, 49, 182]
unsafe-pgd (total 42), index: [48, 50, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68, 69, 70, 71, 72, 74, 75, 76, 77, 78, 79, 80, 81, 82, 84, 85, 86,
                               87, 88, 89, 96, 97, 98, 141, 142, 143, 183]
unknown (total 1), index: [83]
```

Figure 3.3: Summary

Bibliography

[huanzhang12, 2021] huanzhang12 (2021). alpha-beta-crown. `https://github.com/Verified-Intelligence/alpha-beta-CROWN`.

[Katz et al., 2017] Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. (2017). Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 97–117. Springer.

[Kochenderfer and Chryssanthacopoulos, 2011] Kochenderfer, M. J. and Chryssanthacopoulos, J. (2011). Robust airborne collision avoidance through dynamic programming. *Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371*, 130.

[ONNX Contributors, 2023] ONNX Contributors (Accessed: 2023). Open Neural Network Exchange (ONNX). `https://onnx.ai/`.