Fossil 2.0

Design, usage and impact of a software tool for verification and control of dynamical models

Edwards, Alec; Peruffo, Andrea; Abate, Alessandro

DOI
10.1016/j.scico.2025.103354

Publication date
2026

Document Version
Final published version

Published in
Science of Computer Programming

Science of Computer Programming 247 (2026) 103354

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Original Software Publication

Fossil 2.0: Design, usage and impact of a software tool for verification and control of dynamical models

Alec Edwards [a], Andrea Peruffo [b], Alessandro Abate [a,*]

[a] University of Oxford, United Kingdom
[b] TU Delft, Netherlands

ARTICLE INFO

Keywords:
Inductive synthesis
Nonlinear analysis
Formal verification

ABSTRACT

This paper introduces Fossil 2.0, an advanced software tool designed for synthesizing certificates such as Lyapunov and barrier functions for dynamical systems represented by ordinary differential equations and difference equations. Fossil 2.0 features a range of significant enhancements, including improved user interfaces, an expanded library of certificates, controller synthesis capabilities, and an extensible architecture. These advancements are detailed as part of this paper. The core of Fossil is a counterexample-guided inductive synthesis (CEGIS) framework that ensures soundness. The tool employs neural networks as templates to generate candidate functions, which are rigorously validated using a satisfiability modulo theories (SMT) solver. Key improvements over the previous release include support for a broader class of certificates, integration of control law synthesis, and compatibility with discrete-time models.

1. Motivation and significance

Neural network based synthesis is an emerging technique in the control field, with applications ranging from, e.g., fault tolerant control [1], robotics and multi-agent systems [2], safety for stochastic systems [3–6]. In the context of certificates synthesis, most of the existing work has focused on Lyapunov and barrier functions, however it is evident that complex applications require richer specifications, e.g., reaching a target region while avoiding an undesirable (or unsafe) portion of the state space – embodied by the reach-while-avoid certificate available in our tool. We present a software tool that aims at the composition of certificates for ever richer requirements.

Early works on sound Lyapunov and barrier function synthesis can be found in [7–11,1,12–14]. More complex properties, such as 'reach while stay', are discussed in [15–17], while a recent survey on neural certificates is presented in [2].

Fossil 1.0 [18] is a tool based upon earlier works on neural template synthesis for Lyapunov [7,9] and barrier functions [12]. Within this narrower focus, these works benchmark Fossil against alternative synthesis techniques, such as SOStools [19] and neural Lyapunov control (NLC) [8], proving that it outperforms them in terms of computational time, robustness, whilst supporting a larger set of characteristics.

Recently, a general verification framework for dynamical models via inductive synthesis of certificate via counter-example guided inductive synthesis (CEGIS) has been introduced: [20] presents a theoretical framework for controller and certificate synthesis, for

* Corresponding author.
E-mail address: alessandro.abate@cs.ox.ac.uk (A. Abate).

https://doi.org/10.1016/j.scico.2025.103354
Received 23 December 2024; Received in revised form 7 May 2025; Accepted 17 June 2025

Available online 23 June 2025

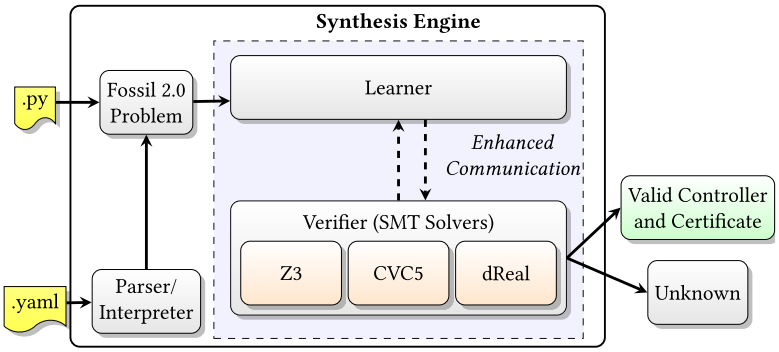

Fig. 1. General architecture of Fossil 2.0. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

a broad range of properties (requirements). Fossil 2.0 is built upon the methodology outlined in [20] in order to construct a new, user-friendly tool with a broad range of uses.

2. Software description

Fossil 2.0's improved codebase enables design of control laws for the satisfaction of a wide range of properties, from the canonical stability, to the rather complex reach while avoid. It handles polynomial and non-polynomial dynamical flows and control laws alike, for convex and non-convex domains. Further, its code architecture allows for an easy extension to tailored certificates (see also Section A.2). Metadata for the software is provided in Table A.2.

2.1. Software architecture

Fossil 2.0 adopts an automated, formal approach for generating certificates by utilizing feed-forward neural networks [21] as potential functions. This approach is grounded in CEGIS, a proven method for addressing second-order logic synthesis problems, comprising two interdependent components.

The first component is the *learner*, which functions within a numerical environment to train a candidate to meet the desired conditions over a finite sample set, D. The second component is the *verifier*. It works in a symbolic environment to determine whether the conditions hold across the entire dense domain, $\mathcal{X}$. If the verifier finds the candidate is incorrect, counterexamples are added to the sample set, prompting the network to retrain. This process repeats until the verifier confirms no counterexamples exist or until a timeout is reached. A high-level overview of Fossil's architecture is shown in Fig. 1.

The success of the CEGIS algorithm depends on the efficient exchange of information between the learner and verifier. In Fossil the CEGIS architecture has been specifically tailored to improve the communication between the numerical and symbolic environments, with the support of specialized subroutines.

2.2. Software functionalities

Here we describe a brief overview of the current functionality of Fossil, and present a high-level overview of its architecture in Fig. 1.

As input, Fossil requires either a Python file (calling the Python module) or a .yaml file (passed via the command line) describing the certificate synthesis problem required. This must define the dynamical model in question, the property to be proved (and the corresponding domains), and other suitable configuration settings (such as the desired certificate template, choice of SMT solver). As output, Fossil returns a yes or no answer based on whether it was able to verify the property. It can also be configured to return the corresponding certificate and controller (if relevant) upon successful termination.

Fossil supports a range of different properties, SMT solvers, domain specifications and other miscellaneous features which are summarised in Table 1. Notably Fossil supports a wide range of different properties for continuous time models, which are depicted graphically in Fig. 2. Detailing these properties is a nontrivial task so we exclude this for brevity, and point the interested reader to [20].

Finally, Fossil handles polynomial and non-polynomial certificates, with linear and non-linear control design, over convex and non-convex sets. Let us emphasise that, in view of the SMT verification engine underpinning it, our tool is *sound*, namely the result of the synthesis is formally valid over (a dense domain within) $\mathbb{R}^n$ - this is unlike many alternatives in the literature.

3. Illustrative examples

Fossil 2.0 is equipped with an *easy-to-use* interface based on command line, which leverages YAML configuration files to define the required parameters for the program. We illustrate the use of Fossil with two test benchmarks: an autonomous and a control model. We delegate advanced user examples of Fossil (using its Python interface and extending its features) to Appendix A.

Table 1
Comparison of features between the two releases of Fossil. Italic red text denotes Fossil 1.0 only, and bold blue text denotes Fossil 2.0 only.

Feature	Details
Interface	*Jupyter Interface*, **Command Line**, **Python Interface**
Properties	Stability, Safety, **SWA**, **RWA**, **RSWA**, **ROA**, **RAR**,
Models	Continuous Time, **Discrete Time**
Verifiers	Z3, dReal, **CVC5**
Domains	Spheres, Boxes, **Open spheres**, **Open boxes**, **Ellipsoids**, Custom sets
Misc.	**Control Synthesis**, **Certificate Extensibility**, **Verifier-only**, **Learner-only**

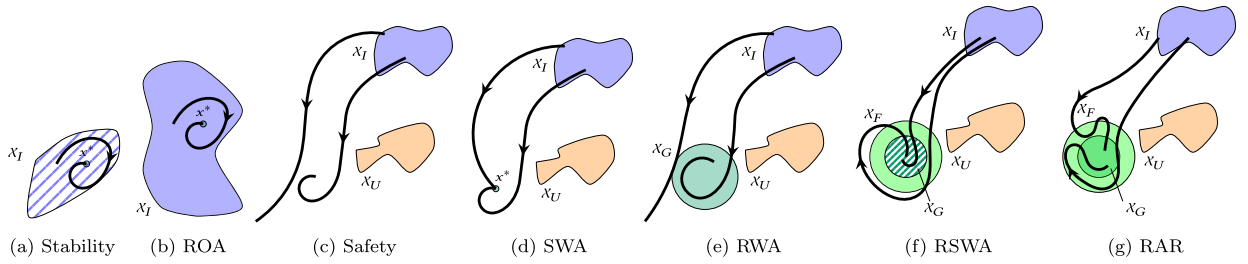

Fig. 2. Pictorial depiction of relevant properties verifiable by Fossil 2.0. Here, $\mathcal{X}_I$ is the initial set, $\mathcal{X}_U$ the unsafe set ($\mathcal{X}_S$ is its safe complement), $\mathcal{X}_G$ the goal/target set, $\mathcal{X}_F$ the final set. (The entire state space is $\mathcal{X}$.) A dashed background denotes that the corresponding set's existence is implied by the corresponding certificate, but that it is not explicitly defined in the property.

3.1. Simple use-case

Let us consider the following continuous-time dynamical model,

$$\begin{cases} \dot{x}_0 = x_1 - x_0^3, \\ \dot{x}_1 = -x_0, \end{cases} \tag{1}$$

which has a single equilibrium located at the origin. We can use Fossil to prove whether this equilibrium is (locally) asymptotically stable by synthesising a Lyapunov function. To this end, it is sufficient to define a YAML file as in Listing 1. We specify the system dynamics and certificate type in the corresponding fields. Since certificates in Fossil are neural networks, we must input their structure as part of the configuration. In this example, we specify a network consisting of a single hidden layer (5 neurons) with quadratic (square) activation functions (resulting in an SOS-like quadratic Lyapunov function). We outline the domain of verification (which implicitly impacts the verified region of attraction) as a hyper-sphere, centred at the origin of radius 1.0. We then specify that 1000 data points should be sampled from this domain to train the Lyapunov function. Finally, we tell Fossil to perform the verification step using Z3. Once Fossil's Python is installed, this can be run using the command `\# fossil config.yaml`. Alternatively, when using a Docker image, a user can run `\# docker run --rm fossil fossil config.yaml`.

```
N_VARS: 2
SYSTEM: [x1 - x0**3, -x0]
CERTIFICATE: Lyapunov
TIME_DOMAIN: CONTINUOUS
DOMAINS:
  XD: Sphere([0,0], 1.0)
N_DATA:
  XD: 1000
N_HIDDEN_NEURONS: [5]
ACTIVATION: [SQUARE]
VERIFIER: Z3
```

Listing 1: Example YAML configuration file to synthesise a Lyapunov function.

3.2. Controller synthesis

Fossil 2.0 is able to synthesise feedback controllers for dynamical models with control input. These controllers are synthesised concurrently with a certificate, and guide the model to satisfy the required conditions. Consider a modified version of the model in (1) as:

$$\begin{cases} \dot{x}_0 = x_1 - x_0^3, \\ \dot{x}_1 = u_0, \end{cases} \tag{2}$$

where u_0 represents a control input. We can modify the configuration file in Listing 1 to synthesise a simple linear controller and Lyapunov function for this model, which we show in Listing 2. In this example, we use dReal as a verifier. In view of the internal mechanics of dReal (ϵ-satisfiability, cf. [22] for a detailed discussion), we should exclude a small region around the origin from the domain, to avoid pathological problems involving the equilibrium point. This issue is limited to Lyapunov certificate synthesis using dReal, and is overcome by employing a spherical domain where a smaller, inner spherical region is removed – e.g., in two dimensions, this results in an annulus. Fossil supports this feature with the domain denoted `Torus(c, r_o, r_i)` which refers to the hyper sphere centred at c of radius r_o (set)-minus the hyper-sphere centred at c of radius r_i. This domain grants the so-called ϵ-stability.

```
N_VARS: 2
SYSTEM: [x1 - x0**3, u0]
CERTIFICATE: Lyapunov
TIME_DOMAIN: CONTINUOUS
DOMAINS:
  XD: Torus([0,0], 1.0, 0.01)
N_DATA:
  XD: 1000
N_HIDDEN_NEURONS: [5, 5]
ACTIVATION: [SIGMOID, SQUARE]
CTRLAYER: [5,1]
CTRLACTIVATION: [LINEAR]
VERIFIER: DREAL
```

Listing 2: Example YAML configuration file to synthesise a Lyapunov function and corresponding feeback controller.

We specify that the Lyapunov certificate should consist of two hidden layers: one of sigmoidal activation functions and one of square activations; the control design instead uses a linear feedback law.

3.3. Reproducible capsule

We do not present additional results in this manuscript as these are present in previous works describing Fossil 2.0 [23] and the underlying research [20]. These results demonstrate Fossil 2.0's ability to outperform its predecessor, which was benchmarked against state-of-the-art SOS-based approaches for Lyapunov and barrier certificate synthesis and found to provide comparable results. The results go further than this, demonstrating Fossil 2.0's ability to successfully find control laws and correctness certificates for a wide range of properties and dynamical models - even in difficult cases involving non-convex (or even disjoint) sets and non-polynomial dynamics. We note that Fossil 2.0 is limited by its dependence on SMT-solving in terms of scalability. This presents most in the dimensionality of the system: the highest dimension in these results is eight. In practice, we are also limited by the size of the neural networks used as candidates, though our results show that the approach can handle sufficiently large networks (consisting of multiple layers) to satisfy the required properties. Advancing this limitation of SMT-solvers is an area of open research, which Fossil can motivate and improve alongside. Instructions for how to use the reproducible capsule to reproduce these results may be found in the Zenodo repository page for Fossil [24], which also contains the source code and a corresponding Docker image. The source code may also be found on GitHub, and the Docker image on DockerHub (detailed in the instructions).

4. Impact

We approach several key problems in the analysis and verification of dynamical systems, modeled either by coupled ordinary differential equations (ODEs) or ordinary difference equations, in terms of three main properties: *reaching* a desired set (either within a finite time or asymptotically), *avoiding* an unsafe or undesirable set, and *remaining* within a specified final set. These properties are fundamental to control theory and have numerous practical applications, ranging from robotic tasks to autonomous vehicles [2]. Many seminal works consider only stability and safety, and use sum-of-squares (SOS) polynomials to convexify the problem and obtain results.

Fossil emerged from a line of research (from its original authors and others) studying alternative synthesis approaches to SOS for synthesis of Lyapunov-like functions. Though effective and a longstanding state-of-the-art approach, concerns regarding the lack of formal guarantees providing by SOS in the context of safety critical systems had emerged. Meanwhile successes in machine learning indicated that using highly expressive neural network templates could result in more successful synthesis.

Since its original release in 2021, Fossil has become a common benchmark, alongside SOS tools, for works which study synthesis of Lyapunov functions and Barrier certificates. Relative to similar approaches for neural-network based synthesis of these certificates, Fossil, to the best of our knowledge, is the only tooled research work which provides a usable interface specify custom certificate templates and dynamics for continuous time models. This is especially true for the second release, and is the only unified tool able to synthesis neural certificates for problems which include non-convex certificate properties and non-polynomial dynamics.

Since the initial release of Fossil we have collaborated with researchers from both academia and industry to explore the best ways to utilise and extend its functionality. We hope this updated release continues this trend.

5. Conclusions

We have presented Fossil 2.0, a software tool designed for the verification of properties in dynamical models through automated formal synthesis of a wide range of certificates, leveraging recent advances in certificate synthesis. This process relies on a CEGIS loop, where neural networks are used to generate candidate functions, which are then verified using SMT solvers.

Fossil 2.0 significantly enhances the capabilities of its previous release, offering a much broader selection of certificate-based verification queries for dynamical models. Additionally, Fossil 2.0 can simultaneously synthesize controllers that guide a model to satisfy a specification while also generating a certificate that verifies the property holds. The new version also features an improved, more user-friendly Python interface, as well as an intuitive command-line interface designed for casual users.

6. Future plans

We hope that Fossil continues to grow as a tool that is useful to both the research and industrial communities interested in certificate-based verification of dynamical models. The second version of Fossil sought to drastically increase the number of properties the tool can reason over, and enable users to more easily define additional properties. Currently, Fossil is limited to deterministic models, but future versions will address expanding the model paradigms Fossil is able to reason over. Furthermore, as the field of neural-network based verification matures, Fossil must keep apace and use the start-of-the-art techniques for verification of these objects. We welcome contributions that improve the features and functionality of Fossil from external collaborators.

CRediT authorship contribution statement

Alec Edwards: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Andrea Peruffo:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Alessandro Abate:** Writing – review & editing, Writing – original draft, Supervision, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Alec Edwards reports financial support was provided by EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (EP/S024050/1). If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank Daniele Ahmed and Mirco Giacobbe for their contributions to the development of the original release of Fossil [18]. We also thank the chairs of HSCC 2024, Manuel Mazo and Erika Ábrahám for organising this special issue.

Appendix A. Advanced usage of Fossil

A.1. Advanced (Python-based) interface

Our command line interface is comprehensive, providing users with the ability to synthesise any of Fossil 2.0's seven certificates alongside control laws. Fossil may also be interfaced as a Python package, allowing for a more feature-rich experience in terms of functionality and extensibility. Let us now describe the definition of the synthesis procedure for the model in (2).

```python
import fossil

class TestModel(fossil.control.ControllableDynamicalModel):
    n_vars = 2  # system variables
    n_u = 1  # control inputs

    def f_torch(self, v, u):  # tensor computations
        x0, x1 = v.T
        u0 = u[:,0]
        return [x1 - x0**3, u0]

    def f_smt(self, v, u):  # smt computations
        x0, x1 = v
        u0 = u
        return [x1 -x0**3, u0]
```

Listing 3: Example model definition.

Within Fossil 2.0, dynamical models may be declared as objects inheriting from either the DynamicalModel class (for simply autonomous models) and ControllableDynamicalModel (for models with control input to be realised as a state-feedback law). The

class presents the number of variables and control inputs as `n_vars` and `n_u`, respectively; autonomous models do not need the instantiation of `n_u`. The two specular methods define the dynamical model, to be manipulated by PyTorch (`f_torch`), whose inputs are tensors of data points, and the SMT solver (`f_smt`), whose inputs are lists of symbolic variables. Both of these must be defined due to differences between PyTorch and the symbolic SMT-solvers used. Following the model definition, we may outline the chosen certificate along with the relevant sets, as follows, where we assume to synthesise a quadratic control Lyapunov function over a spherical domain of radius 10.

```
1  import fossil
2
3  # get the system model
4  open_loop = TestModel
5  system = fossil.control.GeneralClosedLoopModel.
6              .prepare_from_open(open_loop())
7
8  # set the certificate domain
9  XD = fossil.domains.Sphere([0.0, 0.0], 10.)
10 sets = {fossil.XD: XD,}
11 data = {fossil.XD:
12         XD._generate_data(batch_size=500),}
13
14 # certificate and neural architectures parameters
15 opts = fossil.CegisConfig(
16     SYSTEM=system,
17     DOMAINS=sets,
18     DATA=data,
19     N_VARS=open_loop.n_vars,
20     CERTIFICATE=fossil.CertificateType.LYAPUNOV,
21     TIME_DOMAIN=fossil.TimeDomain.CONTINUOUS,
22     VERIFIER=VerifierType.Z3,
23     ACTIVATION=[fossil.ActivationType.SQUARE],
24     N_HIDDEN_NEURONS=[4],
25     CTRLAYER=[15, 1],
26     CTRLACTIVATION=[fossil.ActivationType.LINEAR],
27 )
28
29 # start the synthesis process
30 fossil.synthesise(opts)
```

Listing 4: Example benchmark using Python-package interface.

The procedure first pre-processes the model (line 5) to include the dynamics within a closed-loop model. We then can define the domain set, a sphere centered at the origin (line 9). The domain set is used both in its symbolic formulation, for verification purposes, and as a set to sample datapoints from. These two distinct aspects are specified as `sets` including the symbolic set formulations, whilst `data` denotes the samples generated through the `_generate_data` method.

Following the definition of the Lyapunov certificate and the time domain (lines 20-21), we can set a few additional parameters within the ad-hoc class `CegisConfig`. We choose the Z3 solver as the SMT engine, the candidate certificate is embodied by a neural network with a single hidden layer of 4 neurons with square activation function. Note that by increasing the list of neurons, we increase the layers of the network: e.g. `[4, 5]` creates a network with two hidden layers composed of 4 and 5 neurons, respectively. Finally, we may specify the neural architecture of the control network, a single hidden layer of 15 neurons and 1 outputs (representing the single control input), with a linear activation (denoting a canonical feedback control law) – naturally, the definition of a control architecture is not needed for autonomous models.

The command `synthesise` starts the procedure and its CEGIS loop. The default number of loops is set to 10, but can be easily modified by setting the additional parameter `CEGIS_MAX_ITERS` (not shown). A detailed list of parameters (e.g., certificates, domain sets) supported by Fossil can be found in the parameters guide at the project's repository: https://github.com/oxford-oxcav/fossil [25].

A.2. Extensibility of Fossil

A.2.1. New certificate-based properties

Fossil 2.0 is a tool for verifying properties of dynamical models using certificates. We provide a broad range of certificates for continuous-time models, but we appreciate that users may wish to synthesise certificates that prove properties not covered. With this in mind, Fossil 2.0 presents a codebase that enables extensions to new certificates. Here, we demonstrate how a new certificate can be specified within Fossil.

Let us first explain how Fossil's codebase is structured to enable defining further certificates. At its core, Fossil consists of sub-modules corresponding to the components *learner* and *verifier*. The tasks of the learner and verifier must vary for each certificate: the learner must define a loss function that trains a neural network to satisfy the certificate's conditions while the verifier must falsify these conditions.

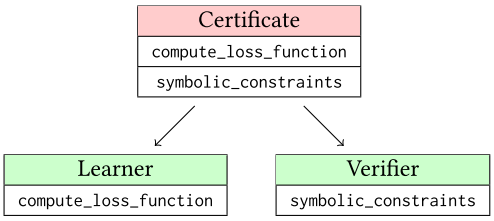

Fig. A.3. Schematic representation of the `Certificate` class providing required functionality to the components of CEGIS.

We delegate the tasks specific to a given certificate to a new module, the *certificate* module. A defined certificate must provide the following functionality: calculation of a loss function to guide learning; and the construction of the symbolic formula consisting of the negation of the conditions for the certificate to be valid. A schematic depiction of the certificate code structure is provided in Fig. A.3.

Let us focus on the classic stability property, and let us consider the Lyapunov certificate that proves it for a continuous time dynamical model. Given a domain $\mathcal{X}$ and a model $f : \mathcal{X} \to \mathcal{X}$ with unique equilibrium point $x^* \in \mathcal{X}$, consider a function $V : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}, V \in C^1$. V is a Lyapunov function if:

$$V(x^*) = 0, \tag{A.1a}$$

$$V(x) > 0 \quad \forall x \in \mathcal{X} \setminus \{x^*\}, \tag{A.1b}$$

$$\dot{V}(x) = \langle \nabla V(x), f(x) \rangle < 0 \quad \forall x \in \mathcal{X} \setminus \{x^*\}. \tag{A.1c}$$

We illustrate in Listing 5 a class which defines the required functionality. The `compute_loss` method calculates a dedicated loss function based on the conditions specified in Eq. (A.1), whilst the `get_constraints` method returns the symbolic constraints relevant for the certificate (specifically the negation of the above conditions).

```
class LyapunovCertificate(Certificate):

def init(self, domain):
    # initialise the domain of verification
    self.domain = domains[XD]

def compute_loss(self, V, grad_V, f):
    """ Calculate loss function based on sample points
    - V: Values of certificate
    - grad_V: Values of gradient of certificate
    - f: Values of vector field
    """
    lyap_loss = relu(-V).mean()
    Vdot = torch.sum(torch.mul(grad_V, f), dim=1)
    lie_loss = (relu(Vdot)).mean()
    loss = lyap_loss + lie_loss
    return loss

def get_constraints(self, verifier, C, Cdot):
    """ SMT-based constraints for Certificate conditions.
    - verifier: Verification object
    - C: Certificate formula
    - Cdot: Certificate lie derivative formula
    """
    lyap_constr = _And(C <= 0, self.domain)
    lie_constr = _And(Cdot >= 0 self.domain)
    return lyap_constr, lie_constr
```

Listing 5: Pseudocode of a Certificate file.

The loss function penalises positive values of $\dot{V}(x)$ and negative values of $V(x)$, hence the choice of the ReLU function. Other choices are possible: accordingly, our tool supports several loss function computations. The `get_constraints` method returns the negation of the symbolic conditions, as the verifier searches for an instance (a counter-example) that satisfies them.

Notice that condition (A.1a) is not included in the certificate file: its satisfaction is automatically guaranteed by considering x^* as the origin (the default setting), by choosing activation functions that evaluate to zero in x^*, and by omitting any network bias, thus ensuring $V(x^*) = 0$.

A.2.2. Bespoke domains

A crucial limitation of the provided command line interface is that all domains specified must be one amongst a hyper-sphere, -torus or -box. Within the package, users may specify domains that are bespoke to their verification problem. This requires defining

two methods: one which returns a symbolic expression representing the domain, and one which provides data points sampled over the domain. Examples of this may be found amongst the large number of benchmarks showcased at [25], and a guide on how to construct them is described in 'set_guide.md'.

A.3. Metadata

Table A.2
Code metadata (mandatory).

Nr.	Code metadata description	
C1	Current code version	v2.1
C2	Permanent link to code/repository used for this code version	https://github.com/oxford-oxcav/fossil
C3	Permanent link to Reproducible Capsule	https://hub.docker.com/r/aleccedwards/fossil, https://doi.org/10.5281/zenodo.14470576
C4	Legal Code License	BSD-3
C5	Code versioning system used	Git/GitHub
C6	Software code languages, tools, and services used	Python 3
C7	Compilation requirements, operating environments and dependencies	x86, Linux (Docker)
C8	If available, link to developer documentation/manual	Documentation in codebase
C9	Support email for questions	aabate@cs.ox.ac.uk

References

[1] D. Grande, D. Fenucci, A. Peruffo, E. Anderlini, A.B. Phillips, T. Giles, G. Salavasidis, Systematic synthesis of passive fault-tolerant augmented neural Lyapunov control laws for nonlinear systems, in: 2023 62nd IEEE Conference on Decision and Control (CDC), 2023.

[2] C. Dawson, S. Gao, C. Fan, Safe control with learned certificates: a survey of neural Lyapunov, barrier, and contraction methods for robotics and control, IEEE Trans. Robot. 39 (3) (2023) 1749–1767, https://doi.org/10.1109/TRO.2022.3232542.

[3] F.B. Mathiesen, S.C. Calvert, L. Laurenti, Safety certification for stochastic systems via neural barrier functions, IEEE Control Syst. Lett. 7 (2023) 973–978.

[4] Đ. Žikelić, M. Lechner, T.A. Henzinger, K. Chatterjee, Learning control policies for stochastic systems with reach-avoid guarantees, Proc. AAAI Conf. Artif. Intell. 37 (10) (2023) 11926–11935.

[5] M. Lechner, Đ. Žikelić, K. Chatterjee, T.A. Henzinger, Stability verification in stochastic control systems via neural network supermartingales, Proc. AAAI Conf. Artif. Intell. 36 (7) (2022) 7326–7336.

[6] K. Chatterjee, T.A. Henzinger, M. Lechner, Đ. Žikelić, A learner-verifier framework for neural network controllers and certificates of stochastic systems, in: S. Sankaranarayanan, N. Sharygina (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Nature Switzerland, Cham, 2023, pp. 3–25.

[7] D. Ahmed, A. Peruffo, A. Abate, Automated and Sound Synthesis of Lyapunov Functions with SMT Solvers, 2018.

[8] Y.-C. Chang, N. Roohi, S. Gao, Neural Lyapunov control, Adv. Neural Inf. Process. Syst. 32 (2019).

[9] A. Abate, D. Ahmed, M. Giacobbe, A. Peruffo, Formal Synthesis of Lyapunov Neural Networks, vol. 5, Institute of Electrical and Electronics Engineers (IEEE), 2021, pp. 773–778.

[10] P. Samanipour, H.A. Poonawala, Stability analysis and controller synthesis using single-hidden-layer relu neural networks, IEEE Trans. Autom. Control (2023) 1–12.

[11] D. Grande, E. Anderlini, A. Peruffo, G. Salavasidis, Augmented neural Lyapunov control, IEEE Access (2023).

[12] A. Peruffo, D. Ahmed, A. Abate, Automated and Formal Synthesis of Neural Barrier Certificates for Dynamical Models, 2021, pp. 370–388.

[13] H. Zhao, X. Zeng, T. Chen, Z. Liu, Synthesizing barrier certificates using neural networks, in: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control, HSCC '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–11.

[14] S. Ratschan, Simulation based computation of certificates for safety of dynamical systems, in: Formal Modeling and Analysis of Timed Systems: 15th International Conference, FORMATS 2017, Berlin, Germany, September 5–7, 2017, Proceedings 15, Springer, 2017, pp. 303–317.

[15] C.F. Verdier, M. Mazo Jr., Formal controller synthesis for hybrid systems using genetic programming, CoRR, arXiv:2003.14322, 2020.

[16] C.F. Verdier, M. Mazo, Formal synthesis of analytic controllers for sampled-data systems via genetic programming, in: 2018 IEEE Conference on Decision and Control (CDC), 2018, pp. 4896–4901.

[17] H. Ravanbakhsh, S. Sankaranarayanan, Counterexample guided synthesis of switched controllers for reach-while-stay properties, CoRR, arXiv:1505.01180, 2015.

[18] A. Abate, D. Ahmed, A. Edwards, M. Giacobbe, A. Peruffo, FOSSIL: a software tool for the formal synthesis of Lyapunov functions and barrier certificates using neural networks, in: Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control, HSCC '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1–11.

[19] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, P.A. Parrilo, SOSTOOLS version 3.00 sum of squares optimization toolbox for MATLAB, CoRR, arXiv:1310.4716, 2013.

[20] A. Edwards, A. Peruffo, A. Abate, A general verification framework for dynamical and control models via certificate synthesis, arXiv:2309.06090, 2023.

[21] D.J.C. MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge University Press, Cambridge, UK; New York, 2003.

[22] S. Gao, J. Kapinski, J. Deshmukh, N. Roohi, A. Solar-Lezama, N. Arechiga, S. Kong, Numerically-robust inductive proof rules for continuous dynamical systems, in: I. Dillig, S. Tasiran (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2019, pp. 137–154.

[23] A. Edwards, A. Peruffo, A. Abate, Fossil 2.0: formal certificate synthesis for the verification and control of dynamical models, in: Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control, HSCC '24, Association for Computing Machinery, New York, NY, USA, 2024.

[24] A. Edwards, A. Peruffo, A. Abate, Fossil, https://doi.org/10.5281/zenodo.14470576, Dec. 2024.

[25] A. Edwards, A. Peruffo, A. Abate, Fossil 2.0 repository, https://github.com/oxford-oxcav/fossil, 2023.