The Symbolic Machine Learning Prover

Franz Brauße Zurab Khasidashvili Konstantin Korovin

February 14, 2024

Abstract

Symbolic Machine Learning Prover (SMLP) is a collection of tools for reasoning about machine learning models. SMLP is based on SMT solver(s). In this document we describe functionality for computing safe and stable regions of neural network models satisfying optput specifications. This corresponds to solving ϵ-guarded $\exists^*\forall^*$ formulas over NN representations [?].

SMLP has been developed by Franz Brauße, Zurab Khasidashvili and Konstantin Korovin and is available under the terms of the Apache License v2.0.[1]

Contents

[1] `https://www.apache.org/licenses/LICENSE-2.0`

1 Changes

- 07/07/2020: documentation of SMLP v0.1

2 Introduction

Symbolic Machine Learning Prover (SMLP) offers multiple capabilities for system's *design space exploration*. These capabilities include methods for selecting which parameters to use in modeling design for configuration optimization and verification; ensuring that the design is robust against environmental effects and manufacturing variations that are impossible to control, as well as ensuring robustness against malicious attacks from an adversary aiming at altering the intended configuration or mode of operation. Environmental affects like temperature fluctuation, electromagnetic interference, manufacturing variation, and product aging effects are especially more critical for correct and optimal operation of devices with analog components, which is currently the main focus area for applying SMLP.

To address these challenges, SMLP offers multiple modes of design space exploration; they will be discussed in detail in Section 6. The definition of these modes refers to the concept of *stability* of an assignment to system's parameters that satisfies all model constraints (which include the constraints defining the model itself and any constraint on model's interface). We will refer to such a value assignment as a *stable witness*, or *(stable) solution* satisfying the model constraints. Informally, stability of a solution means that any eligible assignment in the specified region around the solution also satisfy the required constraints. This notion is sometimes referred to as robustness. We work with parameterized systems, where parameters (also called *knobs*) can be tuned to optimize the system's performance under all legitimate inputs. For example, in the circuit board design setting, topological layout of circuits, distances, wire thickness, properties of dielectric layers, etc. can be such parameters, and the exploration goal would be to optimize the system performance under the system's requirements [MSK21]. The difference between knobs and inputs is that knob values are selected during design phase, before the system goes into operation; on the other hand, inputs remain free and get values from the environment during the operation of the system. Knobs and inputs correspond to existentially quantified and universally quantified variables in the formal definition of model exploration tasks. Thus in the usual meaning of verification, optimization and synthesis, respectively, all variables are inputs, all variables are knobs, and some of the variables are knobs and the rest are inputs.

Below by a *model* we refer to an ML model that models the system under exploration.

Could not used column sep = 8.5em, between origins option in the cube.

The *model exploration cube* in Figure 1 provides a high level and intuitive idea on how the model exploration modes supported in SMLP are related. The three dimensions in this cube represent synthesis ($\searrow$-axis), optimization ($\rightarrow$-axis) and stability ($\uparrow$-axis). On the bottom plane of the cube, the edges represent the synthesis and optimization problems in the following sense: synthesis with constraints configures the knob values in a way that guarantees that assertions are valid, but unlike optimization, does not guarantee optimally with respect to optimization objectives. On the other hand, optimization by itself is not aware of assertions on inputs of the system and only guarantees optimality with respect to knobs, and not the validity of assertions in the configured system. We refer to the process that combines synthesis with optimization and results in an optimal design that satisfies assertions as *optimized synthesis*. The upper plane of

the cube represents introducing stability requirements into synthesis (and as a special case, into verification), optimization, and optimized synthesis. The formulas that make definition of stable verification, optimization, synthesis and optimized synthesis precise are discussed in Sections ?? to ??.

3 Concepts and Preliminaries

In this document, syntactical highlights are placed on `literal strings`, which are ASCII sequences used to communicate input to and output from programs; on `commands` to be executed; and on `API references` which correspond to either concrete symbols or abstract concepts defined in the corresponding API.

We assume familiarity with JSON, `make` and CSV files. Since the CSV format is not unambiguously defined, we give the concrete requirements in appendix A.1.

The glossary in this section defines concepts which are used throughout this document when describing in detail the problems solved by SMLP and is meant to be referred to on an per-instance basis instead of reading it as a whole.

Center Threshold A rational value in $[0, 1]$ larger or equal to Threshold.

Codomain A real vector space.

Data set A list of points in $D \times C$ where D is the domain and C is the codomain.

Domain A domain D is the cartesian product $\bigtimes_{i=1}^{n} D_i$ where D_i is a subset of either $\mathbb{Z}$, $\mathbb{R}$ or a discrete finite subset of $\mathbb{Q}$.

Feature Any column of a data set $\mathcal{D}$ is called a feature.

Instance A tuple $(N, \mathcal{N}_D, o, \mathcal{N}_o)$ is called an instance if N is an NN over domain D, $\mathcal{N}_D$ is a data normalization, o is an objective function and $\mathcal{N}_O$ is an objective normalization. If – in addition to an instance I – a data set $\mathcal{D}$ over D is given, $(I, \mathcal{D})$ is called a data instance.

NN Neural network as understood by the Keras API of Tensorflow in HDF5 format. In particular, the Symbolic Machine Learning Prover only handles `Dense` layers with either `relu` or `linear` activation function. The input layer must be defined on domain D and the output layer must map to the codomain C.

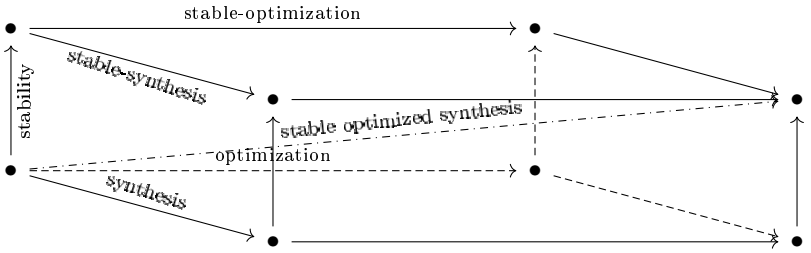

Figure 1: Exploration Cube

Region Given a domain $D = \bigtimes_{i=1}^{d} D_i$, a region (in D) is a product $\bigtimes_{i=1}^{d} C_i$ where each $C_i \subseteq D_i$ is a finite union of compact intervals for $i = 1, \ldots, n$. For discrete D_i, e.g., a subset of $\mathbb{Z}$, C_i is the finite union of point intervals. Otherwise, D_i is a bounded subset of $\mathbb{R}$ and C_i corresponds to just one interval $[c_i \pm r_i]$ with center $c_i \in D_i$ and rational radius $r_i > 0$.

Safe A region R is considered safe (wrt. a given target function f) for a constant $t \in \mathbb{Q}$ if f satisfies $f(x) \geq t$ for all $x \in R$.

.spec Specification file describing the domain, codomain and regions in the domain. Its format is a JSON list where the i-th entry corresponds to the i-th column in the CSV describing the data set.

Given a data set and a .spec file, the components D_i of the domain are defined as $D_i = E_i(F_i)$ where E_i is called the embedding of F_i into D_i where F_i is the i-th input feature.

Target function The target function f_I of an instance $I = (N, \mathcal{N}_D, o, \mathcal{N}_o)$ is defined as the composition $\mathcal{N}_o \circ o \circ \mathcal{N}_{D,\mathrm{o}} \circ N \circ \mathcal{N}_{D,\mathrm{i}}$. The target function $f_{I,\mathcal{D}}$ of a data instance $(I, \mathcal{D})$ is $x \mapsto \min(\{f_I(x)\} \cup \{\mathcal{N}_o(o(y)) : (x, y) \in \mathcal{D}\})$.

Threshold The threshold for a region R and target function f is defined as the maximal $t \in \{t_1, \ldots, t_n\} \subset [0, 1] \cap \mathbb{Q}$ for which R is safe wrt. f for t if it exists, and $-\infty$ otherwise.

The threshold for a domain D and target function f is defined as the maximal $t \in \{t_1, \ldots, t_n\} \subset [0, 1] \cap \mathbb{Q}$ for which there is a region that is safe wrt. f for t if it exists, and $-\infty$ otherwise.

4 Prerequisites

This section describes software environments SMLP is known to work in. All packages corresponding to software listed in in Section 4.2 should be installed "system-wide", meaning that they are accessible to the corresponding interpreter or compiler listed in Section 4.1 without additional options.

Please note, that SMLP may run with versions of the packages not listed here. These lists are subject to change and are gradually extended as soon as more versions have been tested or other package requirements arise.

4.1 Compilers / Interpreters / Utils

These tools should be available in one of the paths mentioned in the `PATH` environment variable.

- `bash`: GNU Bourne Again Shell version 5.0_p17 `http://tiswww.case.edu/php/chet/bash/bashtop.html`
- `gmake`: GNU make version 4.1, 4.2 or 4.3 `https://www.gnu.org/software/make/make.html`
- `awk`: GNU awk version 5.0.1 or 5.1 `https://www.gnu.org/software/gawk/gawk.html`
- `sed`: GNU sed version 4.8 `http://sed.sourceforge.net/`
- GNU coreutils version 8.32 `https://www.gnu.org/software/coreutils/`
- GNU time version 1.7, 1.7.2 or 1.9 `https://www.gnu.org/directory/time.html`

- cc: either GNU C Compiler version 4.7.4, 5.4.0, 9.3.0 or 10.1.0 https://gcc.gnu.org/ or Clang version 10.0.0 https://clang.llvm.org/
- python3: CPython version 3.6 or 3.7 https://www.python.org/

4.2 Libraries

The paths to the installed python directory should be set in the PYTHONPATH environment variable. See python3 -h for details on this variable.

- Tensorflow version 2.1 or 2.2 https://www.tensorflow.org/
- Z3 including its Python API version 4.8.6 or 4.8.8 https://github.com/Z3Prover/z3
- Pandas version 0.24.2 https://pandas.pydata.org/
- scikit-learn version 0.20.4 or 0.22.2_p1 https://scikit-learn.org/
- matplotlib version 2.2.4 or 3.1.2 https://matplotlib.org/
- seaborn version 0.9.x or 0.10.x https://seaborn.pydata.org/
- HDF5 for Python version 2.10.0 https://www.h5py.org/
- kjson version 0.1.3 (bundled in release) https://github.com/fbrausse/kjson

5 Training ML models with SMLP

SMLP supports training tree-based and polynomial models using the scikit-learn[2] and pycaret[3] packages, and training neural networks using the Keras package with TensorFlow[4]. For systems with multiple outputs, SMLP supports training one model with multiple outputs as well as training separate models per response (this is controlled by command-line option *model_per_response*). Supporting these two options allows a trade-off between the accuracy of the models (models trained per response are likely to be more accurate) and with the size of the formulas that represent the model for symbolic analysis (one multi-output model formula will be at least smaller when the same training hyper-parameters are used). Conversion of models to formulas into SMLP language is done internally in SMLP (no encoding options are exposed to user in current implementation, which will change once alternative encodings will be developed).

6 ML model exploration with SMLP

SMLP supports the following model exploration modes (we assume that an ML model M has already been trained).

certify Given an ML model M, a value assignment a to knobs, and a query query, check that a is a stable witness for query on model M. Multiple pairs (a, query) of candidate witness a and query query can be checked in a single SMLP run.

query Given an ML model M and a query query, find a stable witness a for query on M.

[2] https://scikit-learn.org/stable/
[3] https://pycaret.org
[4] https://keras.io

verify Given an ML model M, a value assignment a to knobs, and an assertion assert, verify whether assert is valid on model M for any legal inputs (for the given value assignment a to knobs). SMLP supports verifying multiple assertions in a single run.

synthesize Given an ML model M, find a stable configuration of knobs (a value assignment a to knobs) such that required constraints (possibly including assertions), are valid for all legal value assignments to the inputs.

optimize Given an ML model M, find a stable configuration of knobs that yields a pareto-optimal values of the optimization objectives (pareto-optimal with respect to the *max-min* optimization problem defined in [?]).

optsyn Given an ML model M, find a stable configuration of knobs that yields a pareto-optimal values of the optimization objectives and all constraints and assertions are valid for all legal values of inputs. This mode is a union of the *optimize* and *synthesize* modes, its full name is *optimized synthesis*. All the previous modes can be seen as a special case of optimized synthesis mode.

A formal definition of the tasks accomplished with these model exploration modes can be found in [?]. Formal descriptions combined with informal clarifications will be provided in this section as well. Running SMLP in these modes reduces to solving formulas of the following structure:

$$\exists p \ \left[\eta(p) \wedge \forall p' \ \forall xy \ [\theta(p,p') \implies (\varphi_M(p',x,y) \implies \varphi_{cond}(p',x,y))]\right] \tag{1}$$

where p denotes model knobs, x denotes inputs, y denotes the outputs, $\eta(p)$ defines constraints on the knobs, $\varphi_M(p',x,y)$ defines the ML model constraints, and the condition $\varphi_{cond}(p',x,y)$ depends on the SMLP mode and will discussed in subsections below. In eq. (1), $\theta(p,p')$ is in general a reflexive predicate, and in SMLP it is used as $\theta_r(p,p') = \|p-p'\| \leq r$, where $\|p-p'\|$ is a distance between two configurations p and p', and r is a relative or absolute *radius*.

Definition 1 • *Given a value assignment p^* to knobs p, an assignment x^* to inputs x is called a* θ-stable witness *to* query(p,x,y) *for configuration p^* if the following formula is valid:*

$$\varphi^{px}_{witn}(p^*,x^*) = \eta(p^*) \wedge \ \left(\forall p' \ \forall y \ [\theta(p^*,p') \implies (\varphi_M(p',x^*,y) \implies \varphi_{cond}(p',x^*,y))]\right) \tag{2}$$

where

$$\varphi_{cond}(p,x,y) = \alpha(p,x) \implies \mathsf{query}(p,x,y).$$

In this case the value assignment p^*, x^* to inputs and knobs p, x is called a θ-stable witness for query(p,x,y)*, and x^* is called a* θ-stable witness to query(p^*,x,y).

- *A value assignment p^* to knobs p is called a* θ-stable witness for query(p,x,y) *if any legal value assignment x^* to inputs x is a θ-stable witness to* query(p,x,y) *for p^*; that is, when the following formula is valid:*

$$\varphi^{p}_{witn}(p^*) = \eta(p^*) \wedge \ \left(\forall p' \ \forall xy \ [\theta(p^*,p') \implies (\varphi_M(p',x,y) \implies \varphi_{cond}(p',x,y))]\right) \tag{3}$$

- *An assertion* assert(p,x,y) *is called* θ-valid with respect to knob configuration p^* *if its negation* query$(p,x,y) = \neg$ assert(p,x,y) *does not have a θ-stable witness x^* for p^*. In the latter case assertion* assert(p^*,x,y) *is called* θ-valid*. Otherwise, if x^* is a θ-stable witness for* query(p^*,x,y)*, then we call x^* a* θ-stable counter-example to assert(p^*,x,y)*, and* assert(p^*,x,y) *is* θ-falsifiable *(that is, it is not θ-valid).*

The idea behind the concept of θ-validity of an assertion $\mathsf{assert}(p, x, y)$ with respect to a knob configuration p^* is as follows. Under the assumption that the system is in one of the points in the θ-stability region of p^*, while the probability for the system of being in one of these points is 1 (due to the assumption), the probability for the system of being in a particular point x^* in the θ-stability region is 0. [5] Now, consider an input assignment x^* such that $\mathsf{query}(p^*, x^*, y) = \neg\, \mathsf{assert}(p^*, x^*, y)$ holds and x^* is not a θ-stable witness to $\mathsf{query}(p^*, x, y)$. Then x^* is a counter-example to $\mathsf{assert}(p^*, x, y)$, but x^* can occur with probability 0; as a consequence, $\mathsf{assert}(p^*, x, y)$ does not have a counter-example with probability greater than 0 and is therefore defined as θ-valid in the above definition.

6.1 Certification of candidate stable witness

Certification is defined both for a candidate stable configuration p^* as well as for an assignment p^*, x^* to both inputs and knobs. Still for now perturbation is defined for knobs only, not for inputs. Extension of perturbation to inputs requires more discussion (is this really needed?) Let's first consider the former case. We are given a candidate configuration of knobs p^* (which is a value assignment to knobs p), and a query query on an ML model M, and we want to check whether p^* is a stable witness for query. Intuitively, the latter means that p^* satisfies query on model M for any legal values of inputs x, even if the knob values in configuration p^* are perturbed within a predefined radius. Formally, SMLP checks the validity of formula $\varphi_{cert}(p^*)$:

$$\varphi_{cert}(p^*) = \eta(p^*) \wedge \left(\forall p'\ \forall xy\ [\theta(p^*, p') \implies (\varphi_M(p', x, y) \implies \varphi_{cond}(p', x, y))]\right) \tag{4}$$

where

$$\varphi_{cond}(p, x, y) = \alpha(p, x) \implies \mathsf{query}(p, x, y).$$

and $\alpha(p, x)$ defines user-given constraints on knobs and inputs.

For the above formula to be valid and not be valid vacuously, we need to check that $\eta(p^*)$ evaluates to constant true and $\alpha(p^*, x)$ is satisfiable. This means that p^* witnesses that $\alpha(p, x) \wedge \eta(p)$ is *consistent*, that is, there exist values of inputs that satisfy $\alpha(p^*, x) \wedge \eta(p^*)$. Formally, consistency check for stable witness certification requires the following formula to be valid:

$$\exists x\ [\alpha(p^*, x) \wedge \eta(p^*)]) \tag{5}$$

Syntactic checks in SMLP imposed on the specification file ensure that all knobs are assigned fixed values in p^*. If the consistency check fails, SMLP reports the status of p^* as *not a witness*; otherwise SMLP checks satisfiability of the following formula, which we refer to as *feasibility* part of stable witness certification:

$$\varphi_M(p^*, x, y) \wedge \alpha(p^*, x) \wedge \mathsf{query}(p^*, x, y)) \tag{6}$$

If the above formula is not satisfiable, then p^* cannot be a stable witness to query, and SMLP reports its status as *not a witness*. Otherwise stability of the candidate witness p^* for query is checked by proving validity of formula eq. (4), and this is done by checking satisfiability of formula eq. (7), which is the negation of the right conjunct of eq. (4):

$$\theta(p^*, p') \wedge \varphi_M(p^*, x, y) \wedge \alpha(p^*, x) \wedge \neg\, \mathsf{query}(p^*, x, y)) \tag{7}$$

[5] Here we have the situation that an infinite sun of 0s is 1. The number 1 is selected because we are talking about probability and 1 is the maximum probability. Here we want to make connection between stability and probability, and we need an arithmetic / a calculus where an infinite sum of 0s is not 0. More precisely, likely we want $0 \times \aleph_0 = 0;\ 0 \times \aleph_1 = 1;\ \ldots;\ 0 \times \aleph_i = \aleph_{i-1}$ (0 times the cardinality of rationals; 0 times the cardinality of reals, etc.). Need to figure out what this calculus is – maybe something related to the *Uncertainty Principle* in quantum mechanics.

```
{
        "query1": "not a witness",
        "query2": "stable witness"
}
```


Figure 2: SMLP result in mode "certify".

SMLP supports checking multiple witness-query pairs in a single run. The status of each witness is reported in file $prefix\_dataname\_certify\_results.json$, and each witness has one of these three status options:

- *not a witness* –formula eq. (5) is not valid or formula eq. (6) is not satisfiable.
- *witness, not stable* – formula eq. (4) is not valid (formula eq. (5) might still be valid and formula eq. (6) satisfiable).
- *stable witness* – formula eq. (4) is valid (formula eq. (7) is not satisfiable).

A results file might look like one on figure fig. 2:

Now let's consider that we are given an assignment p^*, x^*, a query query on an ML model M, and we want to check whether x^* is a stable witness to query for p^*. This requires checking validity of formula eq. (2). The consistency check for this task is simply checking validity of $\alpha(p^*, x^*) \wedge \eta(p^*)$, and the feasibility part for this task is checking satisfiability of

$$\varphi_M(p^*, x,^* y) \wedge \alpha(p^*, x^*) \wedge \mathsf{query}(p^*, x^*, y)) \tag{8}$$

If the above formula is not satisfiable, then x^* cannot be a stable witness to query for p^*, and SMLP reports its status as *not a witness*. Otherwise stability of the candidate witness p^*, x^* to query is checked by proving validity of formula eq. (2), and this is done by checking satisfiability of formula eq. (9)

$$\theta(p^*, p') \wedge \varphi_M(p^*, x^*, y) \wedge \alpha(p^*, x^*) \wedge \neg\, \mathsf{query}(p^*, x^*, y)) \tag{9}$$

The result file format is the same as for checking stability of configuration p^*.

6.2 Querying ML model for a stable witness

Given an ML model M and a query $\mathsf{query}(p, x, y)$, the task is to find a stable witness p^* for $\mathsf{query}(p, x, y)$ on M. That is, the task is to find p^* such that $\varphi_{cert}(p)$ (defined in eq. (4)) holds; and hence p^* will be a solution for

$$\varphi_{\mathsf{query}} = \exists p\ \varphi_{cert}(p) = \exists p\ \left[\eta(p) \wedge \left(\forall p'\ \forall xy\ \left[\theta(p, p')\ \implies\ (\varphi_M(p', x, y)\ \implies\ \varphi_{cond}(p', x, y))\right]\right)\right] \tag{10}$$

where

$$\varphi_{cond}(p, x, y) = \alpha(p, x)\ \implies\ \mathsf{query}(p, x, y).$$

For a solution p^* to exist, and for it not to be vacuous, it is necessary $\alpha(p, x) \wedge \eta(p)$ to be consistent, meaning that the following formula must be valid:

$$\exists p, x\ [\alpha(p, x) \wedge \eta(p)]) \tag{11}$$

```
{
        "query1": {
                "status": "UNSAT",
                "witness": null
        },
        "query2": {
                "status": "STABLE_SAT",
                "witness": {
                        "p1": 2.0,
                        "y1": 9.0,
                        "y2": 5.0,
                        "x1": 0.0,
                        "p2": 7.0
                }
        }
}
```


Figure 3: SMLP result in mode "query".

Furthermore, for a solution to exist, the following formula must be satisfiable, and we refer to this as a *feasibility* check for the task of searching for a stable witness for $\mathsf{query}(p, x, y)$.

$$\eta(p) \wedge \varphi_M(p, x, y) \wedge \alpha(p, x) \wedge \mathsf{query}(p, x, y)) \tag{12}$$

When both the consistency check and the feasibility check succeed, SMLP enters an iterative algorithm for searching for a stable witness p^*, and can terminate also by concluding that such a witness does not exist (under some additional assumptions).

SMLP supports querying multiple conditions in one SMLP run. The status of each query is reported in file *prefix_dataname_query_results.json*, and the result might look like the one displayed in Figure fig. 3, where the "witness" field specifies the values of knobs, as well as values of inputs and values of outputs found in the satisfying assignment to eq. (6) that identified the stable witness.

6.3 Synthesis for a query

The task of stable synthesis of knob values for a query query consists of solving the following formula:

$$\exists p, x \ \left[\eta(p) \wedge \forall p' \ \forall y \ [\theta(p, p') \implies (\varphi_M(p', x, y) \implies \varphi_{cond}(p', x, y))]\right] \tag{13}$$

Let p^*, x^* be a solution to eq. (15). Then according to Definition 1, x^* is a stable witness for $\mathsf{query}(p^*, x, y)$.

First, we find a candidate p^*, x^* by solving

$$\exists p, x \ \left[\eta(p) \wedge \forall y \ [(\varphi_M(p, x, y) \implies \varphi_{cond}(p, x, y))]\right] \tag{14}$$

Then check whether the following formula is valid (by checking its negation for satisfiability):

$$\eta(p^*) \wedge \forall p' \ \forall y \ [\theta(p^*, p') \implies (\varphi_M(p', x^*, y) \implies \varphi_{cond}(p', x^*, y))] \tag{15}$$

If the above formula is valid, then we have shown that x^* is a stable witness for $\mathsf{query}(p^*, x, y)$, and the task is accomplished – SMLP reports p^*, x^* is solution to the synthesis task. Otherwise,

search should continue by searching for a solution different from p^*, x^* (and their neighborhood). looks like the full algorithm is same as the synthesis algorithm (interleaving od finding a candidate p^* and searching to its counter-example, except now in search for candidate counter-example the inputs should be bound to values in x^* so that only knobs will be free to assign values in candidate counter-example.

6.4 Assertion verification

In assertion verification usually one assumes that the knobs have already been fixed to legal values, and their impact has been propagated through the constraints, therefore usually in the context of assertion verification knobs are not considered explicitly. However, in order to formalize stability also in the context of verification, SMLP assumes that the values of knobs p are assigned constant values p^*, but can be perturbed (by environmental effects or by an adversary), therefore treatment of p^* is explicit. Then the problem of verifying an assertion $\mathsf{assert}(p^*, x, y)$ under allowed perturbations p' of knob values p^* controlled by $\theta_r(p^*, p')$, as defined in Definition 1, is exactly the problem of certification of stability of witness p^* for query $\mathsf{assert}(p^*, x, y)$, as formalized in eq. (4), where now

$$\varphi_{cond}(p, x, y) = \alpha(p, x) \implies \mathsf{assert}(p, x, y).$$

Therefore, in current implementation, in mode "verify", SMLP supports verification without stability considerations, that is, assumes the stability predicate $\theta(p, p')$ on knobs p, p' is the identity relation (which means, all stability radii are assumed to be 0 and therefore $\theta(p.p') = p == p'$).

Note that the consistency check eq. (5) and feasibility check eq. (6) for mode "certify" in section 6.1 do not refer to stability predicate θ, and are applicable in mode "verify" in a similar way: If the consistency check fails, this is reported to user, with an expectation that the specification file needs to be fixed (this might have been caused by a type).not implemented. Otherwise, if the feasibility check fails (that is, eq. (6) with assert in place of query, is unsatisfiable), then either assert is not valid or it is still possible that the specification is not what was intended. not implemented. Otherwise, formula eq. (16) is checked for satisfiability (recall that $\theta_r(p^*, p'$ is identity in mode "verify" and therefore it does not occur in eq. (16)):

$$\varphi_M(p^*, x, y) \wedge \alpha(p^*, x) \wedge \neg\, \mathsf{assert}(p^*, x, y)) \tag{16}$$

If eq. (16) is satsfiable, SMLP reports the assertion as "FAIL", and otherwise SMLP reports it as "PASS". An example verification results file might look like the one in Figure fig. 4, where the field "asrt" in the result reports the value of the assertion in the SAT assignment to the negated assertion in formula eq. (16), and is displayed as part of the results as a sanity check for the correctness of the counter-example to $\mathsf{assert}(p^*, x, y)$. Assertion verification result is reported in file $prefix\_dataname\_assertions\_results.json$.rename the report file assertions –> verify, to be uniform

6.5 Stable synthesis mode

The task of θ-stable synthesis consists of finding a solution to formula eq. (1), where

$$\varphi_{cond}(p, x, y) = \alpha(p, x) \implies (\beta(p, x, y) \wedge \mathsf{assert}(p, x, y)).$$

. That is, θ-stable synthesis reduces to the task of searching for a θ-stable witness p^* for $\mathsf{query}(p, x, y) = \beta(p, x, y) \wedge \mathsf{assert}(p, x, y))$ (here $\mathsf{assert}(p, x, y)$ might represent a conjunction of multiple assertions).

```
{
        "asrt1": {
                "status": "FAIL",
                "asrt": false,
                "model": {
                        "y1": 9.0,
                        "y2": 5.0,
                        "x3": 7.0,
                        "x1": 0.0,
                        "x2": 2.5
                }
        },
        "asrt2": {
                "status": "PASS",
                "asrt": null,
                "model": null
        }
}
```


Figure 4: SMLP result in mode "verify".

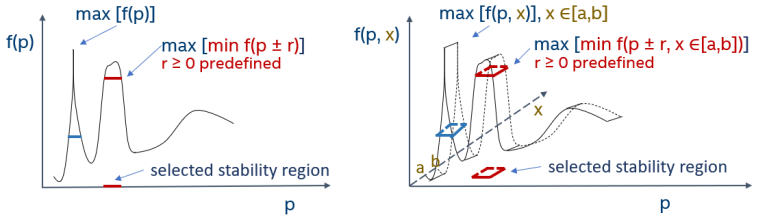

Figure 5: SMLP max-min optimization. On both plots, p denote the knobs. On the right plot we also consider inputs x (which are universally quantified) as part of f.

6.6 Optimization mode

abc

6.7 Optimized synthesis mode

In this subsection we consider the optimization problem for a real-valued function f (in our case, an ML model), extended in two ways: (1) we consider a θ-stable maximum to ensure that the objective function does not drop drastically in a close neighborhood of the configuration where its maximum is achieved, and (2) we assume that the objective function besides knobs depends also on inputs, and the function is maximized in the stability θ-region of knobs, for any values of inputs in their respective legal ranges. We explain these extensions using two plots in Figure 5.

The left plot represents optimization problem for $f(p, x)$ when f depends on knobs only (thus x is an empty vector), while the right plot represents the general setting where x is not empty (which is usually not considered in optimization research). In each plot, the blue threshold (in the form of a horizontal bar or a rectangle) denotes the stable maximum around the point where f reaches its (regular) maximum, and the red threshold denotes the stable maximum, which is

approximated by our optimization algorithms. In both plots, the regular maximum of f is not stable due to a sharp drop of f's value in the stability region.

remove 'near' everywhereLet us first consider optimization without stability or inputs, i.e., far low corner in the exploration cube Figure 1. Given a formula φ_M encoding the model, and an objective function $o : \mathcal{D}_{par} \times \mathcal{D}_{out} \to \mathbb{R}$, the standard optimization problem solved by SMLP is stated by Formula (17).

$$[[\varphi_M]]_o = \max_p\{z \mid \forall y\ (\varphi_M(p,y) \implies o(p,y) \geq z)\} \tag{17}$$

A solution to this optimization problem is the pair $(p^*, [[\varphi_M]]_o)$, where $p^* \in \mathcal{D}_{par}$ is a value of parameters p on which the maximum $[[\varphi_M]]_o \in \mathbb{R}$ of the objective function o is achieved for the output y of the model on p^*. In most cases it is not feasible to exactly compute the maximum. To deal with this, SMLP computes maximum with a specified accuracy. Consider $\varepsilon > 0$. We refer to values $(\tilde{p}, \tilde{z})$ as a solution to the optimization problem with *accuracy* ε, or *ε-solution*, if $\tilde{z} \leq [[\varphi_M]]_o < \tilde{z} + \varepsilon$ holds and $\tilde{z}$ is a lower bound on the objective, i.e., $\forall y[\varphi_M(p,y) \implies o(p,y) \geq \tilde{z}]$ holds.

Now, we consider *stable optimized synthesis*, i.e., the top right corner of the exploration cube. The problem can be formulated as the following Formula (18), expressing maximization of a lower bound on the objective function o over parameter values under stable synthesis constraints.

$$[[\varphi_M]]_{o,\theta} = \max_p\{z \mid \eta(p) \wedge \forall p'\ \forall xy\ [\theta(p,p') \implies (\varphi_M(p',x,y) \implies \varphi^{\geq}_{cond}(p',x,y,z))]\} \tag{18}$$

where

$$\varphi^{\geq}_{cond}(p',x,y,z) = \alpha(p',x) \implies (\beta(p',x,y) \wedge o(p',x,y) \geq z).$$

The stable synthesis constraints are part of a GEAR formula and include usual η, α, β constraints together with the stability constraints θ. Equivalently, stable optimized synthesis can be stated as the *max-min* optimization problem, Formula (19)

$$[[\varphi_M]]_{o,\theta} = \max_p \min_{x,p'}\{z \mid \eta(p) \wedge \forall y\ [\theta(p,p') \implies (\varphi_M(p',x,y) \implies \varphi^{\leq}_{cond}(p',x,y,z))]\} \tag{19}$$

where

$$\varphi^{\leq}_{cond}(p',x,y,z) = \alpha(p',x) \implies (\beta(p',x,y) \wedge o(p',x,y) \leq z).$$

In Formula (19) the minimization predicate in the stability region corresponds to the universally quantified p' ranging over this region in (18). An advantage of this formulation is that this formula can be adapted to define other aggregation functions over the objective's values on stability region. For example, that way one can represent the *max-mean* optimization problem, where one wants to maximize the mean value of the function in the stability region rather one the min value (which is maximizing the worst-case value of f in stability region). Likewise, Formula (19) can be adapted to other interesting statistical properties of distribution of values of f in the stability region.

We can explicitly incorporate assertions in stable optimized synthesis by defining $\beta(p',x,y) = \beta'(p',x,y) \wedge \mathsf{assert}(p',x,y)$, where $\mathsf{assert}(p',x,y)$ are assertions required to be valid in the entire stability region around the selected configuration of knobs p. The notion of ε-solutions for these problems carries over from the one given above for Formula (17).

SMLP implements stable optimized synthesis based on the GearOPT$_\delta$ and GearOPT$_\delta$-BO algorithms [BKK20, BKK22], which are shown to be complete and terminating for this problem under mild conditions. These algorithms were further extended in SMLP to Pareto point computations to handle multiple objectives simultaneously.

	x1	x2	p1	p2	y1	y2
0	2.9800	-1	0.1	4	5.0233	8.0000
1	8.5530	-1	3.9	3	0.6936	12.0200
2	0.5580	1	2.0	4	0.6882	8.1400
3	3.8670	0	1.1	3	0.2400	8.0000
4	-0.8218	0	4.0	3	0.3240	8.0000
5	5.2520	0	4.0	5	6.0300	8.0000
6	0.2998	1	7.1	6	0.9100	10.1250
7	7.1750	1	7.0	7	0.9600	1.1200
8	9.5460	0	7.0	6	10.7007	9.5661
9	-0.4540	1	10.0	7	8.7932	6.4015

Table 1: Toy dataset *smlp_toy_basic.csv* with two inputs x_1, x_2, two knobs i_1, i_2, and two outputs y_1, y_2.

```
 ../src/run_smlp.py -data smlp_toy_basic -out_dir out -pref abc  -mode optimize -pareto t  \
-sat_thresh f -resp y1,y2 -feat x1,x2,p1,p2 -model dt_sklearn -dt_sklearn_max_depth 15 \
-data_scaler min_max -epsilon 0.05 -delta 0.01 -save_model_config f -mrmr_pred 0 -plots f \
-seed 10 -log_time f -spec smlp_toy_basic -pref try
```

Figure 6: Example of SMLP's command to build a decision tree model and find .

7 How to run SMLP

Consider a toy dataset in Table table 1 with two inputs x_1, x_2, two knobs i_1, i_2, and two outputs y_1, y_2. Command to run SMLP in *optimize* mode is given in Figure fig. 6. The optimization problem is specified in the spec file in Figure fig. 7.

Optimization progress is reported in file *try_smlp_toy_basic_optimization_progress.csv*, where *try* is the run ID specified using option '-pref try'; *smlp_toy_basic* is the name of the data file, and *optimization_progress.csv* is suffix for that report. More details on input, knob, output and objective values used to prove the upper and lower bounds of the objectives during search for pareto optimum are reported in file *try_smlp_toy_basic_optimization_progress.json*.

7.1 Design of experiments

Most DOE methods are based on understanding multivariate distribution of legal value combinations of inputs and knobs in order to sample the system. When the number of system inputs and/or knobs is large (say hundreds or more), the DOE may not generate a high-quality coverage of the system's behavior to enable training models with high accuracy. Model training process itself becomes less manageable when number of input variables grows, and models are not explainable and thus cannot be trusted. One way to curb this problem is to select a subset of input features for DOE and for model training. The problem of combining feature selection with DOE generation and model training is an important research topic of practical interest, and SMLP supports multiple practically proven ways to select subsets of features and feature combinations as inputs to DOE and training, including the *MRMR* feature selection algorithm [DP05], and a *Subgroup Discovery (SD)* algorithm [Klö96, Wro97, Atz15]. The MRMR algorithm selects a subset of features according to the principle of *maximum relevance and minimum redundancy*. It is widely used for the purpose of selecting a subset of features for building accurate models, and is therefore useful for selecting a subset of features to be used in DOE; it is a default choice

```
{
  "version": "1.2",
  "variables": [
    {"label":"y1", "interface":"output", "type":"real"},
    {"label":"y2", "interface":"output", "type":"real"},
    {"label":"x1", "interface":"input", "type":"real", "range":[0,10]},
    {"label":"x2", "interface":"input", "type":"int", "range":[-1,1]},
    {"label":"p1", "interface":"knob", "type":"real", "range":[0,10], "rad-rel":0.1, "grid":[2,4,7]},
    {"label":"p2", "interface":"knob", "type":"int", "range":[3,7], "rad-abs":0.2}
  ],
  "alpha": "p2<5 and x1==10 and x2<12",
  "beta": "y1>=4 and y2>=8",
  "eta": "p1==4 or (p1==8 and p2 > 3)",
  "assertions": {
    "assert1": "(y2**3+p2)/2>6",
    "assert2": "y1>=0",
    "assert3": "y2>0"
  },
  "objectives": {
    "objective1": "(y1+y2)/2",
    "objective2": "y1"
  }
}
```

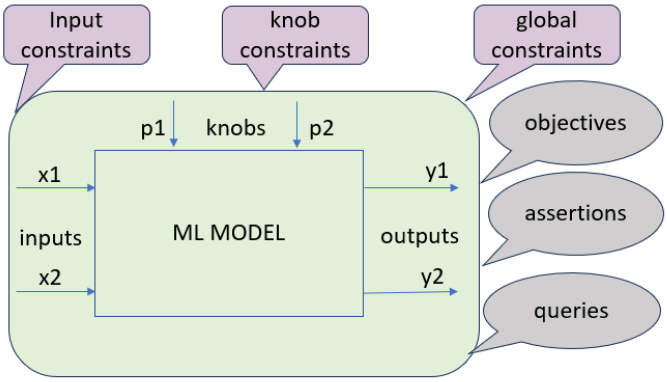

Figure 7: Specification *smlp_toy_basic* used by SMLP command in Figure fig. 6.

in SMLP for that usage. The SD algorithm selects regions in the input space relevant to the response, using heuristic statistical methods, and such regions can be prioritized for sampling in DOE algorithms.

7.2 Root cause analysis

We view the problem of root cause analysis as dual to the stable optimized synthesis problem: while during optimization with stability we are searching for regions in the input space (or in other words, characterizing those regions) where the system response is good or excellent, the task of root-causing can be seen as searching for regions in the input space where the system response is not good (is unacceptable). Thus simply by swapping the definition of excellent vs unacceptable, we can apply SMLP to explore weaknesses and failing behaviors of the system.

Even if a number of witnesses (counter-examples to an assertion) are available, they represent discrete points in the input space and it is not immediately clear which value assignments to which variables in these witnesses are critical to explain the failures. Root causing capability in SMLP is currently supported through two independent approaches: a *Subgroup Discovery (SD)* algorithm that searches through the data for the input regions where there is a higher ratio (thus, higher probability) of failure; to be precise, SD algorithms support a variety of *quality functions* which play the role of optimization objectives in the context of optimization. To find input regions with high probability of failure, SMLP searches for stable witnesses to failures. These capabilities, together with feature selection algorithms supported in SMLP, enable researchers to develop new root causing capabilities that combine formal methods with statistical methods for root cause analysis.

7.3 Model refinement loop

Support in SMLP for selecting DOE vectors to sample the system and generate a training set was discussed in Subsection 7.1. Initially, when selecting sampling points for the system, it is unknown which regions in the input space are really relevant for the exploration task at hand. Therefore some DOE algorithms also incorporate random sampling and sampling based on previous experience and familiarity with the design, such as sampling nominal cases and corner cases, when these are known. For model exploration tasks supported by SMLP, it is not required to train a model that will be an accurate match to the system everywhere in the legal search space of inputs and knobs. We require to train a model that is an *adequate* representation of the system for the task at hand, meaning that the exploration task solved on the model solves this task for the system as well. Therefore SMLP supports a *targeted model refinement* loop to enable solving the system exploration tasks by solving these tasks on the model instead. The idea is as follows: when a stable solution to model exploration task is found, it is usually the case that there are not many training data points close to the stability region of that solution. This implies that there is a high likelihood that the model does not accurately represent the system in the stability region of the solution. Therefore the system is sampled in the stability region of the solution, and these data samples are added to the initial training data to retrain the model and make it more adequate in the stability region of interest. Samples in the stability region of interest can also be assigned higher weights compared to other samples to help training to achieve higher accuracy in that region. More generally, higher adequacy of the model can be achieved by sampling distributions biased towards prioritizing the stability region during model refinement.

Note that model refinement is required only to be able to learn properties on the system by exploring these properties on the model. As a simple scenario, let us consider that we want to check an assertion $\mathsf{assert}(p, x, y)$ on the system. If a θ-stable counter example x^* to $\mathsf{assert}(p, x, y)$ for a configuration p^* of knobs exists on the model according to Definition 1 (which means that x^* is a θ-stable witness for query $\mathsf{query}(p^*, x, y) = \neg\, \mathsf{assert}(p^*, x, y)$), then the system is sampled in the θ–stability region of p^* (possibly with x^* toggled as well in a small region around x^*). If the failure of assertion $\mathsf{assert}(p, x, y)$ is reproduced on the system using the stable witness x^* and a configuration in the θ-stability region of p^*, then the model exploration goal has been accomplished (we found a counter-example to $\mathsf{assert}(p, x, y)$ on the system) and model refinement can stop. Otherwise the system samples can be used to refine the model in this region. For that reason, the wider the θ-stability region, the higher the chances to reproduce failure of assertion $\mathsf{assert}(p, x, y)$ on the system.

On the other hand, if the $\mathsf{assert}(p, x, y)$ does not have a θ-stable counter-example, then it is still possible that assert is not valid on the system but it cannot be falsified on the model due to discrepancy between the system and model responses in some (unknown to us) input space. In this case one can strengthen $\mathsf{assert}(p, x, y)$ to $\mathsf{assert}'(p, x, y)$ (for example, assertion $\mathsf{assert}(p, x, y) = y \geq 3$ can be strengthened to $\mathsf{assert}'(p, x, y) = y \geq 3.01$), or one can weaken the stability condition θ_r to $\theta_{r'}$ by shrinking the stability radii r to r'; or do both. If the strengthened assertion $\mathsf{assert}'(p, x, y)$ has a $\theta_{r'}$-stable counter-example x' for a configuration p' on the model, then p', x' can be used to find a counter-example to the original assertion $\mathsf{assert}(p, x, y)$ on the system in the same way as with p^*, x^* before. If failure of the original assertion assert is reproduced on the system, the model refinement loop stops, otherwise it can continue using other strengthened versions of the original assertion and or shrinking the stability radii even further. Similar reasoning is applied to other modes of design space exploration. In particular, in the optimization modes, one can confirm or reject on the system an optimization threshold proved on the model, and in the latter case the model refinement loop will be triggered by or providing

new data-points that can be used for model refinement in this region, through re-training or incremental training of a new model.

References

[Atz15] Martin Atzmueller. Subgroup discovery. *WIREs Data Mining Knowl. Discov.*, 5(1):35–49, 2015.

[BKK20] Franz Brauße, Zurab Khasidashvili, and Konstantin Korovin. Selecting stable safe configurations for systems modelled by neural networks with ReLU activation. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 119–127. IEEE, 2020.

[BKK22] Franz Brauße, Zurab Khasidashvili, and Konstantin Korovin. Combining constraint solving and bayesian techniques for system optimization. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 1788–1794. ijcai.org, 2022.

[DP05] Chris H. Q. Ding and Hanchuan Peng. Minimum redundancy feature selection from microarray gene expression data. *J. Bioinform. Comput. Biol.*, 3(2):185–206, 2005.

[Klö96] Willi Klösgen. Explora: A multipattern and multistrategy discovery assistant. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 249–271. AAAI/MIT Press, 1996.

[MSK21] Alex Manukovsky, Yuriy Shlepnev, and Zurab Khasidashvili. Machine learning based design space exploration and applications to signal integrity analysis of 112Gb SerDes systems. In *2021 IEEE 71st Electronic Components and Technology Conference (ECTC)*, pages 1234–1245, 2021.

[Wro97] Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In Henryk Jan Komorowski and Jan M. Zytkow, editors, *Principles of Data Mining and Knowledge Discovery, First European Symposium, PKDD '97, Trondheim, Norway, June 24-27, 1997, Proceedings*, volume 1263 of *Lecture Notes in Computer Science*, pages 78–87. Springer, 1997.

A Concrete Formats

A.1 CSV

A structured line-based text format where the first line describes a list of column headers and each line after it describes a list the values corresponding to the respective column in the data set. A line describes a list of elements $(e_i)_{i=1}^n$, if it corresponds to the concatenation $s(e_1),s(e_2),\ldots,s(e_n)$ where $s(v)$ is the ASCII representation of v.

The list of column headers shall not contain duplicates. The ASCII representation of ASCII strings made up of printable characters excluding "," is the identity. Floating point approximations are represented as the decimal encoding where "." is the separator between integer and fractional part. Rational numbers $\frac{p}{q}$ are represented by $s(p)/s(q)$ where $s(z)$ is the decimal encoding of $|z|$ prefixed by "-" if $z \in \mathbb{Z}$ is negative.