ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/236943114

Challenging SMT solvers to verify neural networks

Article *in* Ai Communications · January 2012

DOI: 10.3233/AIC-2012-0525

CITATIONS
55

READS
531

2 authors:

Luca Pulina
Università degli Studi di Sassari
102 PUBLICATIONS 804 CITATIONS

SEE PROFILE

Armando Tacchella
Università degli Studi di Genova
141 PUBLICATIONS 4,067 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Using convolutional neural network to calculate effective properties of porous electrode View project

Project CARVE - ComposAble Robot behaViors with vErification View project

All content following this page was uploaded by Armando Tacchella on 03 January 2014.

The user has requested enhancement of the downloaded file.

Challenging SMT solvers to verify neural networks ⋆

Luca Pulina[1] and Armando Tacchella[2]

[1] DEIS, Università di Sassari, Piazza Università 11, Sassari, Italy.
lpulina@uniss.it
[2] DIST, Università di Genova, Viale Causa 13, Genova, Italy.
Armando.Tacchella@unige.it

Abstract. In this paper we evaluate state-of-the-art SMT solvers on encodings of verification problems involving Multi-Layer Perceptrons (MLPs), a widely used type of neural network. Verification is a key technology to foster adoption of MLPs in safety-related applications, where stringent requirements about performance and robustness must be ensured and demonstrated. In previous contributions, we have shown that safety problems for MLPs can be attacked by solving Boolean combinations of linear arithmetic constraints. However, the generated encodings are hard for current state-of-the-art SMT solvers, limiting our ability to verify MLPs in practice. The experimental results herewith presented are meant to provide the community with a precise picture of current achievements and standing open challenges in this intriguing application domain.

本文我们评估了先进的SMT求解器在编码MLP验证方面的问题(MLP是一种广泛使用的神经网络)。验证是在与安全相关的应用程序中促进MLP采用率的关键技术，在这些应用程序中必须确保有关性能和健壮性的严格要求。在以前的贡献中，我们已经证明了MLP的安全问题可以通过解决线性算术约束的布尔组合来解决。然而，生成的编码对于目前最先进的SMT求解器依然是困难的，这限制了我们在实践中验证MLPs的能力。在此提出的实验结果是为了向社区提供一个关于这个有趣的应用领域的当前成就和开放挑战的精确图片。

1 Introduction

SMT solvers [3] have enjoyed a recent widespread adoption to provide reasoning services in various applications, including interactive theorem provers like Isabelle [11], static checkers like Boogie [8], verification systems, e.g., ACL2 [22], Caduceus [10], software model checkers like SMT-CBMC [1], and unit test generators like CREST [15]. Research and development of SMT algorithms and tools is a very active research area, as witnessed by the annual competition, see e.g. [2]. It is fair to say that SMT solvers are the tool of choice in automated reasoning tasks involving Boolean combinations of constraints expressed in decidable background theories.

交互的

SMT求解器[3]最近被广泛采用，在各种应用中提供推理服务，包括交互式定理证明器如Isabelle[11]，静态检验器如Boogie[8]，验证系统如ACL2 [22]，Caduceus[10]，软件模型检验器如SMT-cbmc[1]，单元测试生成器如CREST[15]。SMT算法和工具的研究和开发是一个非常活跃的研究领域，正如每年的比赛所见证的，如[2]。公平地说，SMT求解器是自动推理任务中选择的工具，这些任务涉及到在可决定背景理论中约束的布尔组合的表达。

This paper is motivated by the fact that SMT solvers can also be used to solve formal verification problems involving neural networks [5]. Verification is indeed a standing challenge for neural networks which, in spite of some exceptions (see e.g., [23]), are confined to non-safety related equipment. The main reason is the lack of general automated, yet effective safety assurance methods for neural-based systems, whereas existing mathematical methods require manual effort and ad-hoc arguments to justify safety claims [23]. In our previous contributions [20, 21] we considered the problem of

本文的动机是SMT求解器也可以用于解决神经网络[5]的形式验证问题。尽管有一些例外(如[23])，但对神经网络来说，验证确实是一个长期的挑战，它仅限于与安全无关的设备。主要原因是缺乏通用的、自动化的、有效的安全保证方法的神经系统，而现有的数学方法需要人工努力和特别的论点来证明安全主张[23]。

⋆ This research has received funding from the European Community's Information and Communication Technologies Seventh Framework Programme [FP7/2007-2013] under grant agreement N. 215805, the CHRIS project.
Proceedings of the 18^{th} RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2011).
In conjunction with IJCAI 2011, Barcelona, Spain, July 17-18, 2011.

verifying a specific kind of neural network known as Multi-Layer Perceptron (MLP). The main feature of MLPs is that, even with a fairly simple topology, they can in principle approximate any non-linear mapping $f : \mathbb{R}^n \to \mathbb{R}^m$with $n, m \geq 1$ – see [16]. In [20] we proposed a first step towards MLP verification, checking that the output of an MLP always ranges within stated safety thresholds – a "global" safety bound. Our technique, that we defined Counter-Example Triggered Abstraction-Refinement (CETAR), computes an overapproximation of the concrete MLP (the abstract MLP) and checks it by solving a satisfiability problem in Quantifier Free Linear Arithmetic over Reals (QF_LRA in [7]) using the solver HySAT [12]. Conservative abstraction guarantees that safety of the abstract MLP implies safety of the concrete one. On the other hand, realizable counterexamples demonstrate that the concrete MLP is unsafe, whereas spurious counterexamples *trigger* a refinement of the abstraction. This approach is closely related to recent works in inductive synthesis of concurrent programs – see, e.g., [24, 25]. More recently, in [21] we extended the CETAR approach to verify that an MLP is guaranteed to satisfy also "local" safety bounds. Informally speaking, this is about guaranteeing that the MLP output is close to some known value – or range of values – modulo the expected error variance. In [21] we showed that also local safety can be checked with abstraction-refinement techniques based on QF_LRA encodings.

在我们之前的文章[20,21]中，我们考虑了验证一种被称为多层感知器(MLP)的特定神经网络的问题。MLPs的主要特点是，即使拓扑相当简单，原则上也可以近似任何非线性映射f: Rn→Rmwith n, m≥1 -参见[16]。在[20]中，我们提出了MLP验证的第一步，即检查MLP的输出是否始终在规定的安全阈值范围内——一个“全局”安全界限。我们的技术,我们定义反例触发Abstraction-Refinement (CETAR),计算具体的overapproximation MLP抽象(MLP)并检查它通过求解可满足性问题在量词自由线性运算在实数([7]QF LRA)使用解决HYSAT[12]。保守的抽象保证了抽象MLP的安全性意味着具体MLP的安全性。另一方面，可实现的反例表明具体的MLP是不安全的，而虚假的反例则会触发对抽象的细化。这种方法与最近在并发程序的归纳综合方面的工作密切相关，例如，参见[24,25]。最近，在[21]中，我们扩展了CEGAR方法来验证MLP也保证满足“局部”安全边界。非正式地说，这是为了保证MLP输出接近某些已知值——或值的范围——以期望误差方差为模。在[21]中，我们还证明了可以使用基于QF LRA编码的抽象细化技术来检查局部安全性。

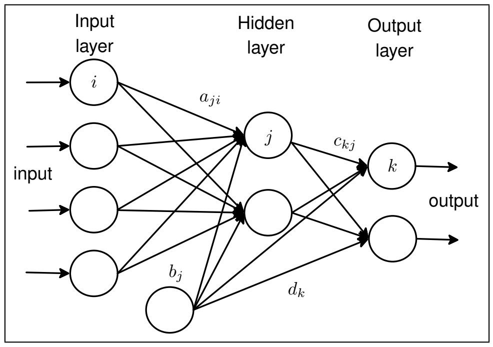
The goal of this paper is to compare state-of-the-art SMT solvers on challenging test cases derived from verification problems involving MLPs – see Section 3. In particular, we consider HySAT [12], our solver of choice in [20, 21]; MATHSAT [6], the winner of SMTCOMP 2010 in the QF_LRA category; and YICES [9] the winner of SMTCOMP 2009 in the same category. We also consider CVC [4] as a baseline in the comparison. All the solvers above are tested extensively on different families of instances related to satisfiability checks in QF_LRA. These instances are obtained considering neural-based estimation of internal forces in the arm of the humanoid James [18] – see Section 2. Such estimation is used in [13] to compute external forces and to detect, e.g., contact of the arm with obstacles or humans. In order to ensure safe actions in dynamic unstructured environments such estimates must be guaranteed against misbehaviors in all possible working configurations. In particular, in Section 4 we describe three groups of experiments on the selected solvers. The first is a competition-style evaluation considering different safety flavours (local and global), different types and sizes of MLPs, and different degrees of abstraction "grain". The second is an analysis of scalability considering satisfiable and unsatisfiable encodings and varying the parameters which mostly influence performances in this regard. The last one is verification of the MLPs proposed in [13] using our system NeVer [20] with different solvers as back-ends. This experimental analysis shows that current state-of-the-art SMT solvers have the capability of attacking several non-trivial (sub)problems in the MLP verification arena. However, the overall verification process, particularly for networks of realistic size and fine grained abstractions, remains a standing open challenge.

本文的目的是将最先进的SMT求解器与涉及MLP的验证问题衍生的具有挑战性的测试用例进行比较 – 参见第3节。特别是，我们考虑了HYSAT [12]，这是我们在[20, 21]中选择的求解器;MATHSAT [6]，2010年SMTCOMP QF LRA类获奖者；和YICES [9]曾获得2009年SMTCOMP冠军。在比较中，我们还将CVC [4]作为基准。以上所有求解器都在与QF LRA中的可满足性检查相关的不同实例系列中进行了广泛的测试。考虑到基于神经的机器人詹姆斯[18]手臂内力的估计，获得了这些实例 – 参见第2节。
在[13]中使用这种估计来计算外力并检测例如手臂与障碍物或人的接触。为了确保在动态的非结构化环境中采取安全措施，必须保证此类估计不会在所有可能的工作配置中出现不良行为。特别是，在第4节中，我们描述了所选求解器上的三组实验。第一种是竞争方式的评估，其中考虑了不同的安全口味（本地和全局），MLP的类型和大小以及抽象度“颗粒”的不同。第二个是对可伸缩性的分析，它考虑了令人满意的编码和不满意的编码，并改变了在这方面主要影响性能的参数。最后一个是使用我们的系统NEVER [20]，以不同的求解器作为后端，对[13]中提出的MLP进行验证。该实验分析表明，当前最先进的SMT求解器具有攻击MLP验证领域中几个不重要的（子）问题的能力。但是，整个验证过程，尤其是对于具有实际大小和细粒度抽象的网络而言，仍然是一个长期存在的挑战。

2 Verification of MLPs: case study and basic concepts

All the encodings used in our analysis are obtained considering verification problems related to a control subsystem in the humanoid James [18]. James is a torso composed

考虑到与机器人James[18]中控制子系统有关的验证问题，获得了我们在分析中使用的所有编码。 詹姆斯是一个躯干组成的



$a[1, \ldots, h][1, \ldots, n]$ ▷ Input weights
$c[1, \ldots, m][1, \ldots, h]$ ▷ Output weights
$b[1, \ldots, h]$ ▷ Hidden nodes' bias
$d[1, \ldots, m]$ ▷ Output nodes' bias

FUNCTION $\nu(\underline{x})$
  for $j \leftarrow 1$ **to** h **do**
    $r_j \leftarrow 0$;
    for $i \leftarrow 1$ **to** n **do**
      $r_j \leftarrow r_j + a_{ji} \cdot x_i + b_j$
  for $k \leftarrow 1$ **to** m **do**
    $s_k \leftarrow 0$;
    for $j \leftarrow 1$ **to** h **do**
      $s_k \leftarrow s_k + c_{kj} \cdot \sigma_h(r_j) + d_k$
    $y_k \leftarrow \sigma_o(s_k)$
  return $\underline{y}$

Fig. 1. Single hidden-layer MLP representations: graphical (left) and pseudo-code (right). In the graphical representation neurons and connections are represented by circles and arrows, respectively. In the pseudo-code, n, h and m are the numbers of input, hidden and output nodes, respectively; a,b,c and d are inductively synthesized numerical parameters; $\sigma_h : \mathbb{R} \rightarrow \mathbb{R}$ and $\sigma_o : \mathbb{R} \rightarrow \mathbb{R}$ are the functions computed in the hidden and output nodes, respectively; $\nu : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is the function computed by the MLP.

of a seven degrees-of-freedom (DOF) head, a seven DOF left arm with an eight DOF hand. In [13] James was extended to detect potentially unsafe situations, e.g., contact with obstacles or humans, by measuring external forces using a single force/torque sensor placed along the kinematic chain of the arm. The problem is that measuring external forces requires *compensation* of the manipulator dynamic, i.e., the contribution of internal forces must be subtracted from sensor readings. In [13] neural networks are suggested as a possible solution to estimate internal forces. The actual network used to provide such estimation has to be considered safety-related equipment: An "incorrect" approximation of internal forces, may lead to either undercompensation – the robot is lured to believe that an obstacle exists – or overcompensation – the robot keeps moving even when an obstacle is hit. Both conditions represent potential hazards, e.g., in contexts characterized by close physical human-robot interaction. Internal forces estimation in [13] relies on a Multi-Layer Perceptron [14] (MLP). In the following, we introduce the main technical aspects of MLPs and their usage to estimate internal forces in James' arm.

考虑到与类人动物James [18]中的控制子系统有关的验证问题，获得了我们在分析中使用的所有编码。詹姆斯（James）是一个躯干，由七个自由度（DOF）头，一个七个自由度的左臂和一个八个自由度的手组成。在[13]中，James扩展到通过使用沿着手臂运动链放置的单个力/扭矩传感器测量外力来检测潜在的不安全情况，例如与障碍物或人的接触。问题在于，测量外力需要补偿机械手的动力，即必须从传感器读数中减去内力的作用。在[13]中，建议将神经网络作为估计内力的可能解决方案。用于提供这种估计的实际网络必须被视为与安全相关的设备：内力的“不正确”近似值可能导致补偿不足-机器人被诱使相信存在障碍-或补偿过度-机器人保持运动即使碰到障碍物也可以。两种情况都代表潜在的危险，例如，在以人与机器人密切互动为特征的环境中。 [13]中的内力估计依赖于多层感知器[14]（MLP）。在下文中，我们介绍了MLP的主要技术方面及其在估算James手臂内力方面的用法。

MLPs are usually represented as a system of interconnected computing units (neurons), which are organized in layers. Figure 1 (left) shows an MLP consisting of three layers. The *input layer* serves to pass the input vector to the network, the *hidden layer* provides a first stage of computation, the *output layer* provides the final output. Operationally, MLPs implement the pseudo-code shown in Figure 1 (right). Given the network $\nu : \mathbb{R}^n \rightarrow \mathbb{R}^m$. The total input received by a neuron is called *induced local field* (ILF). With n neurons in the input layer and h neurons in the hidden layer the ILF of the j-th

MLP通常表示为互连计算单元（神经元）的系统，这些系统按层组织。 图1（左）显示了一个由三层组成的MLP。 输入层用于将输入向量传递到网络，隐藏层提供第一级计算，输出层提供最终输出。 在操作上，MLP实现图1（右）所示的伪代码。 给定网络ν：Rn→Rm。 神经元接收到的总输入称为感应局部场（ILF）。 输入层有n个神经元，隐藏层有h个神经元，第j个隐藏神经元的ILF定义为

hidden neuron is defined as

$$r_j = \sum_{i=1}^{n} a_{ji} x_i + b_j \qquad j = \{1, \ldots, h\} \tag{1}$$

where a_{ji} is the *weight* of the connection from the i-th neuron in the input layer to the j-th neuron in the hidden layer, and the constant b_j is the *bias* of the j-th neuron. The output of a neuron j in the hidden layer is a monotonic non-linear function of its ILF, the *activation function* σ_h. As long as the activation function is differentiable everywhere, MLPs with *only one* hidden layer can, in principle, approximate any real-valued function with n real-valued inputs [16]. Commonly used activation functions (see, e.g., [5]) are sigmoidal non-linearities such as the *hyperbolic tangent* (tanh) and the *logistic* (logi) functions defined as follows:

其中aji是从输入层的第i个神经元到隐藏层的第j个神经元的连接权重，常数bj是第j个神经元的偏差。 隐藏层中神经元j的输出是其ILF的单调非线性函数，即激活函数σh。 只要激活函数在任何地方都是可微的，原则上只有一个隐藏层的MLP都可以近似于具有n个实值输入的任何实值函数[16]。 常用的激活函数（例如，参见[5]）是S型非线性，例如双曲正切（tanh）和逻辑（logi）函数，其定义如下

$$\tanh(r) = \frac{e^r - e^{-r}}{e^r + e^{-r}} \qquad \text{logi}(r) = \frac{1}{1+e^{-r}} \tag{2}$$

where $\tanh : \mathbb{R} \to (-1, 1)$ and $\text{logi} : \mathbb{R} \to (0, 1)$. The MLP suggested in [13] uses hyperbolic tangents, but our encodings are obtained using the logistic function instead.[3] With m neurons in the output layer the ILF of an output neuron is

$$s_k = \sum_{j=1}^{h} c_{kj} \sigma_h(r_j) + d_k \qquad k = \{1, \ldots, m\} \tag{3}$$

where c_{kj} denotes the weight of the connection from the j-th neuron in the hidden layer to the k-th neuron in the output layer, while d_k represents the bias of the k-th output neuron. Therefore, the output of the MLP is a vector $\nu(\underline{x}) = \{\sigma_o(s_1), \ldots, \sigma_o(s_m)\}$. The activation function σ_o can be either the identity function, i.e., $\nu(\underline{x}) = \{s_1, \ldots, s_m\}$, or a sigmoidal non-linearity as in (2). Notice that in the latter case, each output of ν is constrained to range within a known interval *by construction*. In applications where this is feasible, e.g., by rescaling input domains, this choice of σ effectively mitigates the risk of exceedingly low or high output values. In [13] tanh is used as activation function in the output neurons for this reason, and all inputs are rescaled in the range $[-1, 1]$.

注意，在后一种情况下，通过构造将ν的每个输出限制在已知间隔内的范围内。 在可行的应用中（例如，通过重新调整输入域的大小），这种σ的选择可有效降低输出值过低或过高的风险。 因此，在[13]中，tanh用作输出神经元的激活函数，并且所有输入都在[-1，1]范围内重新缩放。

Using MLPs for estimation amounts to choose appropriate weights a, b, c and d. Technically, this is a *regression problem*, i.e., we are given a set of *patterns* – input vectors $X = \{\underline{x}_1, \ldots, \underline{x}_t\}$ with $\underline{x}_i \in \mathbb{R}^n$ – and a corresponding set of *labels* – output vectors $Y = \{\underline{y}_1, \ldots, \underline{y}_t\}$ with $\underline{y}_i \in \mathbb{R}^m$. We think of the labels as generated by some unknown function $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ applied to the patterns, i.e., $\varphi(\underline{x}_i) = \underline{y}_i$ for $i \in \{1, \ldots, t\}$. The task of ν is thus to *extrapolate* φ, i.e., construct ν from X and Y so that given some $\underline{x}^* \notin X$ we ensure that $\nu(\underline{x}^*)$ is "as close as possible" to $\varphi(\underline{x}^*)$. In [13] internal forces in James' arm are estimated considering angular positions and velocities of two shoulder and two elbow joints, i.e., for each $\underline{x} \in X$, we have $\underline{x} = \langle q_1, \ldots, q_4, \dot{q}_1, \ldots, \dot{q}_4 \rangle$.[4] Labels $\underline{y} \in Y$ are corresponding values of internal forces and torques – denoted by f and τ, respectively – in a Cartesian space, i.e.,

[3] From a practical standpoint, the impact of our choice is negligible, since the logistic function has the same "shape" of the hyperbolic tangent, and they are often used interchangeably.

[4] This is standard control-theory notation, where q represents the angular position of the joint, and $\dot{q}$ the angular velocity, i.e., the derivative of q with respect to time.

$\underline{y} = \langle f_1, f_2, f_3, \tau_1, \tau_2, \tau_3 \rangle$. The unknown relation $\varphi : \mathbb{R}^8 \to \mathbb{R}^6$ is the one tying joint positions and velocities to internal forces, and takes into account components from gravity, Coriolis forces, and manipulator inertia.

未知关系?：R8→R6是将关节位置和速度与内力联系在一起的一种，它考虑了重力，科里奥利力和机械手惯性的分量。

Given a set of patterns X and a corresponding set of labels Y the process of tuning weights of an MLP ν in order to extrapolate φ – manipulator dynamic in our case – is called *training*, and the pair (X, Y) is called the *training set* – see [14] for details about the process. Even assuming that a training set is sufficient to learn φ, it is still the case that different sets may yield different MLP weights. The problem is that the resulting MLP may *underfit* the unknown target φ, i.e., consistently deviate from φ, or *overfit* φ, i.e., be very close to φ only when the input pattern is in the training set. These phenomena lead to poor *generalization* performances, i.e., the MLP largely fails to predict $\varphi(\underline{x}^*)$ on inputs $\underline{x}^* \notin X$. In our experiments, we use a *leave-one-out estimate* of the generalization error of an MLP as follows. Given the set (X, Y), the MLP $\nu_{(i)}$ is obtained by training on the patterns $X_{(i)} = \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_t\}$ and corresponding set labels $Y_{(i)}$. If we repeat the process t times, then we have t different MLPs and we obtain the estimate

$$\hat{\epsilon}_k = \frac{1}{t} \sum_{i=1}^{t} (\underline{y}_{ki} - \nu_{(i)}(\underline{x}_i))^2 \qquad k \in \{1, \ldots, m\} \tag{4}$$

where m is the number of outputs of each $\nu_{(i)}$. Notice that $\hat{\epsilon}_k$ is the mean squared error (MSE) among all the predictions made by each $\nu_{(i)}$ when tested on input $\underline{x}_i$. Clearly, $\underline{\hat{\epsilon}}$ should be kept at a minimum to ensure good generalization properties, but, as we show in the next Section, there is also an interplay between certain kinds of safety conditions and the standard deviation of $\underline{\hat{\epsilon}}$.

3 SMT encodings to verify MLPs

We consider two verification problems involving MLPs, and an approach to solve them using SMT encodings – more precisely, encodings in Quantifier Free Linear Arithmetic over Reals (QF_LRA) as defined in [7]. Both problems have been originally introduced in two previous contributions of ours [20, 21], and can be viewed as a way to ensure a safe behaviour of MLPs. In [20] we introduced *global safety*, i.e., checking that the response of the MLP ranges within a stated interval for all acceptable inputs, whereas in [21] we introduced *local safety*, i.e., checking that the response of the MLP is "close to" some training label as long as the input is "close to" the corresponding training input pattern. Both notions of safety can be useful to ensure that the response of an MLP does not behave unpredictably to the point of disrupting the functionality of control systems relying on it. In the following we will always consider MLPs with $n \geq 1$ inputs and $m \geq 1$ outputs in which the *input domain* is a Cartesian product $\mathcal{I} = D_1 \times \ldots \times D_n$ where $D_i = [a_i, b_i]$ is a closed interval bounded by $a_i, b_i \in \mathbb{R}$ for all $1 \leq i \leq n$; analogously, the *output domain* is a Cartesian product $\mathcal{O} = E_1 \times \ldots \times E_m$ where $E_i = [c_i, d_i]$ is a closed interval bounded by $c_i, d_i \in \mathbb{R}$ for all $i \leq i \leq m$.[5]

我们考虑了两个涉及MLP的验证问题，以及一种使用SMT编码解决它们的方法-更准确地说，是[7]中定义的基于实数的无量化线性算术（QF LRA）中的编码。 这两个问题最初是在我们的前两篇文章中提出的[20，21]，可以看作是确保MLP安全行为的一种方式。 在[20]中，我们引入了全局安全性，即检查了所有可接受输入的MLP响应是否在规定的间隔内；而在[21]中，我们引入了局部安全性，即检查了MLP的响应是否“ 接近” 只要输入“ 接近 ” 相应的训练输入模式，就可以使用“ 一些训练标签 ” 。 两种安全概念对于确保MLP的响应不会破坏其所依赖的控制系统功能的行为都非常有用。

[5] In the definitions above, and throughout the rest of the paper, a closed interval $[a, b]$ bounded by $a, b \in \mathbb{R}$ is the set of real numbers comprised between a and b, i.e, $[a, b] = \{x \mid a \leq x \leq b\}$ with $a \leq b$.

3.1 Global safety

Checking for global safety of an MLP $\nu : \mathcal{I} \to \mathcal{O}$ amounts to prove that

$$\forall \underline{x} \in \mathcal{I}, \forall k \in \{1, \ldots, m\} : \nu_k(\underline{x}) \in [l_k, h_k] \tag{5}$$

where $\nu_k(\underline{x})$ denotes the k-th output of ν, and $l_k, h_k \in E_k$ are *safety thresholds*, i.e., constants defining an interval wherein the k-th component of the MLP output is to range, given all acceptable input values. In [20] we proposed to verify a *consistent abstraction* of ν, i.e., a function $\tilde{\nu}$ such that if the property corresponding to (5) is satisfied by $\tilde{\nu}$ in a suitable abstract domain, then it must hold also for ν. The key point is that verifying condition (5) in the abstract domain can be encoded to a satisfiability check in QF_LRA. Clearly, abstraction is not sufficient per se, because of *spurious counterexamples*, i.e., abstract counterexamples that do not correspond to concrete ones. A spurious counterexample calls for a refinement of the abstraction which, in turn, generates a new satisfiability check in QF_LRA. Therefore, a global safety check for a single output of ν may generate several logical queries to the underlying SMT solver. In practice, we hope to be able to either verify ν or exhibit a counterexample within a reasonable number of refinements.

In the following, we briefly sketch how to construct consistent abstractions and related refinements to check for property (5). The proof that the proposed abstraction is consistent and further details about the construction can be found in [20]. Given a concrete domain $D = [a, b]$, the corresponding abstract domain is $[D] = \{[x, y] \mid a \leq x \leq y \leq b\}$, i.e., the set of intervals inside D, where $[x]$ is a generic element. We can naturally extend the abstraction to Cartesian products of domains, i.e., given $\mathcal{I} = D_1 \times \ldots \times D_n$, we define $[\mathcal{I}] = [D_1] \times \ldots \times [D_n]$, and we denote with $[\underline{x}] = \langle [x_1], \ldots, [x_n] \rangle$ the elements of $[\mathcal{I}]$ that we call *interval vectors*. Given a generic MLP ν – the *concrete* MLP – we construct the corresponding *abstract* MLP by assuming that σ_h is the logistic function $\mathrm{logi} : \mathbb{R} \to (0, 1)$ as defined in (2), and that σ_o is the identity function. Since no ambiguity can arise between σ_h and σ_o, we denote σ_h with σ for the sake of simplicity. Given an abstraction parameter $p \in \mathbb{R}^+$, the *abstract activation function* $\tilde{\sigma}^p$ can be obtained by considering the maximum increment of σ over intervals of length p. Since σ is a monotonically increasing function, and the tangent to σ reaches a maximum slope of $1/4$ we have that

$$\forall x \in \mathbb{R} : 0 \leq \sigma(x + p) - \sigma(x) \leq \frac{p}{4} \tag{6}$$

for any choice of the parameter $p \in \mathbb{R}^+$. Now let x_0 and x_1 be the values that satisfy $\sigma(x_0) = p/4$ and $\sigma(x_1) = 1 - p/4$, respectively, and let $p \in (0, 1)$. We define $\tilde{\sigma}^p : [\mathbb{R}] \to [[0, 1]]$ as follows

$$\tilde{\sigma}^p([x_a, x_b]) = \begin{cases} [0, p/4] & \text{if } x_b \leq x_0 \\ [0, \sigma(p\lfloor \frac{x_b}{p} \rfloor) + \frac{p}{4}] & \text{if } x_a \leq x_0 \text{ and } x_b < x_1 \\ [\sigma(p\lfloor \frac{x_a}{p} \rfloor), \sigma(\lfloor \frac{x_b}{p} \rfloor) + \frac{p}{4}] & \text{if } x_0 < x_a \text{ and } x_b < x_1 \\ [\sigma(p\lfloor \frac{x_a}{p} \rfloor), 1] & \text{if } x_0 < x_a \text{ and } x_1 \leq x_b \\ [1 - p/4, 1] & \text{if } x_a \geq x_1 \\ [0, 1] & \text{if } x_a \leq x_0 \text{ and } x_1 \leq x_b \end{cases} \tag{7}$$

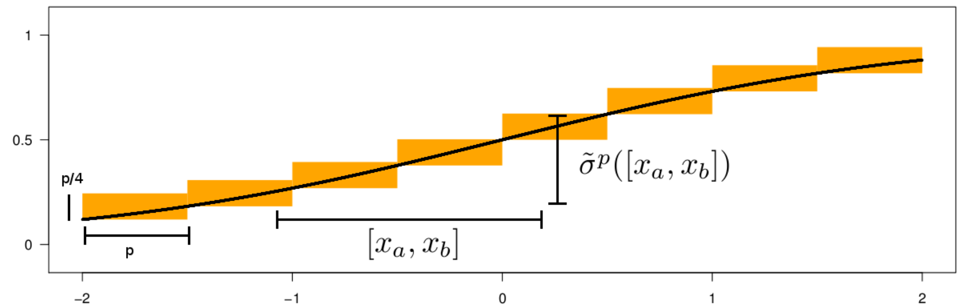


Fig. 2. Activation function $\sigma(x)$ and its abstraction $\tilde{\sigma}^p(x)$ in the range $x \in [-2, 2]$. The solid line denotes σ, while the boxes denote $\tilde{\sigma}^p$ with $p = 0.5$.

Figure 2 gives a pictorial representation of the above definition.

According to (7) we can control how much $\tilde{\sigma}^p$ over-approximates σ, since large values of p correspond to coarse-grained abstractions, whereas small values of p correspond to fine-grained ones. We can now define $\tilde{\nu}^p : [\mathcal{I}] \rightarrow [\mathcal{O}]$ as

$$\tilde{\nu}^p_k([\underline{x}]) = \sum_{j=1}^{h} c_{kj} \tilde{\sigma}^p(\tilde{r}_j([\underline{x}])) + d_k \qquad k = \{1, \ldots, m\} \tag{8}$$

where $\tilde{r}_j([\underline{x}]) = \sum_{i=1}^{n} a_{ji}[x_i] \tilde{+} b_j$ for all $j = \{1, \ldots, h\}$, and we overload the standard symbols to denote products and sums among interval vectors, e.g., we write $x + y$ to mean $x \tilde{+} y$ when $x, y \in [\mathbb{R}]$. Since $\tilde{\sigma}^p$ is a consistent abstraction of σ, and products and sums on intervals are consistent abstractions of the corresponding operations on real numbers, defining $\tilde{\nu}^p$ as in (8) provides a consistent abstraction of ν – see [20] for details. This means that our original goal of proving the safety of ν according to (5) can be now recast, modulo refinements, to the goal of proving its abstract counterpart

$$\forall [\underline{x}] \in [\mathcal{I}], \forall k \in \{1, \ldots, m\} : \tilde{\nu}^p_k([\underline{x}]) \sqsubseteq [l_k, h_k] \tag{9}$$

where "$\sqsubseteq$" stands for the usual containment relation between intervals, i.e., given two intervals $[a, b] \in [\mathbb{R}]$ and $[c, d] \in [\mathbb{R}]$ we have that $[a, b] \sqsubseteq [c, d]$ exactly when $a \geq c$ and $b \leq d$, i.e., $[a, b]$ is a subinterval of – or it coincides with – $[c, d]$.

3.2 Local safety

As we mentioned in Section 2, in James' test case the need to check for global safety is somewhat mitigated using sigmoidal non-linearities in the output neurons to "squash" the response of the MLP within an acceptable range, modulo rescaling. A more stringent, yet necessary, requirement is represented by local safety. Informally speaking, we can say that an MLP ν trained on a dataset (X, Y) of t patterns is "locally safe" whenever given an input pattern $\underline{x}^*$ it turns out that $\nu(\underline{x}^*)$ is "close" to $\underline{y}_j \in Y$ as long as $\underline{x}^*$

is "close" to $\underline{x}_j \in X$ for some $j \in \{1, \ldots, t\}$. As we argue in [21], local safety cannot be guaranteed by design, because the range of acceptable values varies from point to point. Moreover, it ensures that the error of an MLP never exceeds a given bound on yet-to-be-seen inputs, and it ensures that the response of an MLP is relatively stable with respect to small perturbations in its input.

To formalize local safety, given an MLP $\nu : \mathcal{I} \to \mathcal{O}$, and a training set (X, Y) consisting of t elements, we introduced the following concepts.

- Given two patterns $\underline{x}, \underline{x}' \in X$ their *distance along the i-th dimension* is defined as $\delta_i(\underline{x}, \underline{x}') = |x'_i - x_i|$.
- Given some $\underline{x} \in X$, the function $N_i^q : X \to 2^X$ maps $\underline{x}$ to the set of *q-nearest-neighbours along the i-th dimension*, i.e., the first q elements of the list $\{\underline{x}' \in X \mid \underline{x}' \neq \underline{x}\}$ sorted in ascending order according to $\delta_i(\underline{x}, \underline{x}')$.
- Given some $\underline{x} \in X$, the function $\delta_i^q : X \to \mathbb{R}$ maps $\underline{x}$ to the *q-nearest-distance along the i-th dimension*, i.e.,

$$\delta_i^q(\underline{x}) = \max_{\underline{x}' \in N_i^q(\underline{x})} \delta_i(\underline{x}, \underline{x}')$$

- The *q-n-polytope* $\mathcal{X}_j^q$ corresponding to $\underline{x}_j \in X$ for some $j \in \{1, \ldots, t\}$ is the region of space comprised within all the $2n$ hyper-planes obtained by considering, for each dimension i, the two hyper-planes perpendicular to the i-th axis and intersecting it in $(x_i - \delta_i^q(\underline{x}))$ and $(x_i + \delta_i^q(\underline{x}))$, respectively.

The above definitions can be repeated for labels, and thus $\mathcal{Y}_j^q$ denotes a q-m-polytope associated to $\underline{y}_j \in Y$ for some $j \in \{1, \ldots, t\}$. In the following, when the dimensionality is understood from the context, we use $\mathcal{X}$ to denote 1-n-polytopes, and $\mathcal{Y}$ to denote 1-m-polytopes. In the following, we refer to q as the *neighborhood size*.

Given an MLP $\nu : \mathcal{I} \to \mathcal{O}$ with training set (X, Y) consisting of t patterns we consider, for all $j \in \{1, \ldots, t\}$, the set of *input polytopes* $\{\mathcal{X}_1, \ldots, \mathcal{X}_t\}$ associated with each pattern $\underline{x}_j \in X$, and the set of *output polytopes* $\{\mathcal{Y}_1^q, \ldots, \mathcal{Y}_t^q\}$ associated with each label $\underline{y}_j \in Y$, for a fixed value of q. We say that ν is locally safe if the following condition is satisfied

$$\forall \underline{x}^* \in \mathcal{I},\ \exists j \in \{1, \ldots, t\} : \underline{x}^* \in \mathcal{X}_j \to \underline{\nu}(\underline{x}^*) \in \mathcal{Y}_j^q \tag{10}$$

Notice that this condition is trivially satisfied by all the input patterns $\underline{x}^*$ such that $\underline{x}^* \in \mathcal{I}$ but $\underline{x}^* \notin \mathcal{X}_j$ for all $j \in \{1, \ldots, t\}$. Indeed, these are patterns which are "too far" from known patterns in the training set, and for which we simply do not have enough information in terms of local (un)safety. Also, we always consider 1-n-polytopes on the input side of ν whereas we can vary the size of neighbourhoods on the output side by increasing q. Clearly, the larger is q, the larger is the neighbourhood considered in the output, and the less stringent condition (10) becomes. As we show in [21], this additional degree of freedom is important in order to "tune" the safety condition according to the expected variance in the network error. Intuitively, assuming that we obtained a network whose expected error mean and variance is satisfactory, if we try to certify such network on safety bounds which imply a smaller error variance, we will invariably generate feasible counterexamples.

Solver	Total		Sat		Unsat	
	#	Time	#	Time	#	Time
YICES	809	27531.90	639	22141.03	170	5390.87
HYSAT	698	17641.61	561	13089.82	137	4551.79
MATHSAT	683	41975.01	544	35043.15	139	6931.86
CVC	–	–	–	–	–	–

Table 1. Evaluation results at a glance. We report the number of encodings solved within the time limit ("#") and the total CPU time ("Time") spend on the solved encodings. Total number of formulas solved ("Total") is also split in satisfiable and unsatisfiable formulas ("Sat") and ("Unsat"), respectively. Solvers are sorted according to the number of encodings solved. A dash means that a solver did not solve any encoding in the related group.

The abstraction-refinement approach to check local safety is similar to the one described for global safety. In particular, the abstract network $\tilde{\nu}^p$ is obtained as shown previously, i.e., by abstracting the activation function and thus the whole network to compute interval vectors. The only difference is that in [21], and according to the usage pattern in James' test case, we consider networks whose output neurons compute logistic functions instead of identity. The abstract local safety condition corresponding to (10), for any fixed value of q, is

$$\forall[\underline{x}^*] \in [\mathcal{I}],\, \exists i \in \{1, \ldots, t\} : [\underline{x}^*] \in \mathcal{X}_i \rightarrow \tilde{\nu}([\underline{x}^*]) \sqsubseteq \mathcal{Y}_i^q \tag{11}$$

It can be shown (see [21]) that the above condition implies local safety of the concrete network ν, and, as in the case of (9), verifying it can be encoded into a QF_LRA satisfiability check.

4 Challenging SMT solvers to verify MLPs

本节中详细介绍的实验是在一系列相同的Linux工作站上进行的，这些工作站由10台具有4GB RAM的Intel Core 2 Duo 2.13 GHz PC组成。 除非另有说明，否则授予求解器的资源为600s CPU时间和2GB内存。 评估中涉及的求解器为CVC [4]（版本3.2.3，默认选项），HYSAT [12]（版本0.8.6， ε = 1005和 δ = 106选项），MATH SAT [6]（版本4），-no随机决策选项）和YICES [9]（版本2，默认选项）。

The experiments detailed in this section are carried out on a family of identical Linux workstations comprised of 10 Intel Core 2 Duo 2.13 GHz PCs with 4GB of RAM. Unless otherwise specified, the resources granted to the solvers are 600s of CPU time and 2GB of memory. The solvers involved in the evaluation are CVC [4] (version 3.2.3, default options), HYSAT [12] (version 0.8.6, $\varepsilon = 10^{-5}$ and $\delta = 10^6$ options), MATHSAT [6] (version 4, `-no_random_decisions` option), and YICES [9] (version 2, default options).

为了比较solver，我们使用考虑全局和局部安全性，不同类型和大小的MLP以及不同程度的抽象程度的编码。我们将编码分为“ 套房 ”和“ 家庭 ”。前者的区别在于全球与局部安全性，我们从中获得两个套件，即GLOBAL和LOCAL。对于每个套件，我们将隐藏的神经元数量和输出神经元数量不同的家族中的编码分组。 HN-XX ON-YY族分别表示具有XX隐藏和YY输出神经元的编码。我们在{5，10，20}范围内改变隐藏神经元的数量，并考虑一个或六个输出神经元。对于每个套件，我们都会结合这些参数的不同值来生成所有可能的编码。最后，对于每个系列，我们从p = 0.5编码公式，并以r = 1.3的比率递减，以模拟[20]中提出的优化程序。当我们为每个家庭获得20种编码时，我们将停止。此外，在LOCAL的情况下，我们计算具有不同邻域大小的编码，即q = {1，10，20，50，100，199}。

To compare the solvers we use encodings considering both global and local safety, different types and sizes of MLPs, and different degrees of abstraction grain. We classify the encodings in "Suites" and "Families". The former distinction is about global vs. local safety, from which we obtain two suites, namely GLOBAL and LOCAL. For each suite, we group encodings in families differing for the number of hidden neurons and the numbers of output neurons. The family HN-XX_ON-YY denotes encodings with XX hidden and YY output neurons, respectively. We vary the number of hidden neurons in the range $\{5, 10, 20\}$ and we consider either one or six output neurons. For each suite, we produce all possible encodings obtained combining different values of these parameters. Finally, for each family, we encode formulas from $p = 0.5$, and decreasing it by a rate $r = 1.3$ in order to simulate the refinement procedure proposed in [20]. We stop when we obtain 20 encodings for each family. Moreover, in the case of LOCAL, we compute encodings with different neighborhood sizes, i.e. $q = \{1, 10, 20, 50, 100, 199\}$.

在表1中，我们报告了评估结果的整体情况。在下文中，当我们说“ 求解器A支配求解器B ”时，是指B所解决的问题集是A所解决的问题的子集。查看结果，我们可以看到除CVC之外的所有求解器都能够解决至少70%的测试集。 CVC在达到时间限制之前会耗尽内存资源，并且它无法解析任何编码，因此我们从分析中删除了它。仍然看表1，我们可以看到YICES在征服96%的测试集方面胜过其他求解器，而HYSAT和MATHSAT分别可以解决83%和81%。尽管HYSAT和MATHSAT的性能非常相似-仅将15种编码分开，但我们可以看到HYSAT花费了MATHSAT所花费的CPU时间的约42%。我们还报告说，相对于YICES（约34s）和MATHSAT（约61s），HYSAT的每次编码平均时间最佳（约25s）。最后，我们报告评估的求解器的可满足性结果没有差异。

In Table 1 we report a global picture of the evaluation results. In the following, when we say that “solver A *dominates* solver B” we mean that the set of problems solved by B is a subset of those solved by A. Looking at the result, we can see that all solvers but CVC, were able to solve at least 70% of the test set. CVC exhausts memory resources before reaching the time limit, and it is able to solve no encodings, thus we drop it from the analysis. Still looking at Table 1, we can see that YICES outperforms the other solvers conquering 96% of the test set, while HYSAT and MATHSAT were able to solve 83% and 81%, respectively. Despite the very similar performance of HYSAT and MATHSAT – only 15 encodings separate them – we can see that HYSAT spends about 42% of the CPU time spent by MATHSAT. We also report that HYSAT, has the best average time per encoding (about 25s) with respect to both YICES (about 34s) and MATHSAT (about 61s). Finally, we report no discrepancies in the satisfiability result of the evaluated solvers.

Table 2 shows the results of the evaluation dividing the encodings by suites and families. As we can see, in terms of number of encodings solved, YICES is the strongest solver. Concerning the suite GLOBAL, it leads the count with 109 solved encoding (91% of the test set), while concerning the LOCAL suite, it solves 700 encodings (97% of the test set). Focusing on the suite GLOBAL, 10 encodings separate the strongest solver from the weakest one – HYSAT, that solves 99 encodings (82% of the test set). If we consider the problems that are uniquely solved, then we see that no solver is dominated by the others. Now focusing on the suite LOCAL, the first thing to observe is that the difference between the strongest and the weakest solver is increased: 101 encodings separate YICES and MATHSAT, the was able to solve 580 encodings (about 80% of the test set). We also report that MATHSAT is dominated by YICES.

In Table 3 we show the classification of encodings included in the test set. In the table, the number of encodings solved and the cumulative time taken for each family is computed considering the “SOTA solver”, i.e., the ideal solver that always fares the best time among all considered solvers. An encoding is thus solved if at least one of the solvers solves it, and the time taken is the best among all times of the solvers that solved the encoding. The encodings are classified according to their hardness with the following criteria: easy encodings are those solved by all the solvers, medium encodings are those non easy encodings that could still be solved by at least two solvers, medium-hard encodings are those solved by one reasoner only, and hard encodings are those that remained unsolved.

According to the data summarized in Table 3, the test set consisted in 840 encodings, 821 of which have been solved, resulting in 636 easy, 97 medium, 88 medium-hard, and 19 hard encodings. Focusing on families comprised in the suite GLOBAL, we report that all 120 encodings were solved, resulting in 80 easy, 31 medium, and 9 medium-hard encodings. Considering the families in LOCAL, we report that 821 encodings (out of 840) were solved, resulting in 636 easy, 97 medium, and 88 medium-hard encodings.

Finally, we report the contribution of each solver to the composition of the SOTA solver. Focusing on the suite GLOBAL, YICES contributed to the SOTA solver 77 times

Suite	Family	Solver	Total		Sat		Unsat		Unique	
			#	Time	#	Time	#	Time	#	Time
GLOBAL (120)	HN-5_ON-1 (20)	YICES	20	23.889	20	23.889	–	–	–	–
		MATHSAT	20	281.56	20	281.56	–	–	–	–
		HYSAT	12	288.91	12	288.91	–	–	–	–
	HN-5_ON-6 (20)	YICES	20	28.90	20	28.90	–	–	–	–
		HYSAT	20	712.30	20	712.30	–	–	–	–
		MATHSAT	19	548.96	19	548.96	–	–	–	–
	HN-10_ON-1 (20)	YICES	19	493.71	19	493.701	–	–	3	115.68
		HYSAT	17	816.59	17	816.59	–	–	1	478.53
		MATHSAT	9	731.21	9	731.21	–	–	–	–
	HN-10_ON-6 (20)	HYSAT	19	902.85	19	902.85	–	–	1	4.68
		YICES	19	1188.10	19	1188.10	–	–	–	–
		MATHSAT	15	1112.26	15	1112.26	–	–	–	–
	HN-20_ON-1 (20)	MATHSAT	20	1178.47	20	1178.47	–	–	2	326.58
		YICES	18	1637.82	18	1637.82	–	–	–	–
		HYSAT	17	1066.20	17	1066.20	–	–	–	–
	HN-20_ON-6 (20)	MATHSAT	20	1034.77	20	1034.77	–	–	2	406.29
		HYSAT	14	961.40	14	961.40	–	–	–	–
		YICES	13	2633.74	13	2633.74	–	–	–	–
LOCAL (720)	HN-5_ON-1 (120)	YICES	120	910.77	82	451.68	38	459.09	13	592.63
		MATHSAT	106	5103.72	74	2860.42	32	2243.30	–	–
		HYSAT	98	1639.34	70	393.22	28	1246.11	–	–
	HN-5_ON-6 (120)	HYSAT	120	20.57	100	19.53	20	1.04	–	–
		YICES	120	1225.49	100	1134.06	20	91.43	–	–
		MATHSAT	113	4714.75	93	4272.69	20	442.06	–	–
	HN-10_ON-1 (120)	YICES	120	2808.87	83	1603.70	37	1205.17	22	2194.79
		HYSAT	94	5382.65	67	3469.57	27	1913.08	–	–
		MATHSAT	94	6802.15	67	4412.35	27	2389.80	–	–
	HN-10_ON-6 (120)	YICES	120	4047.18	100	3760.97	20	286.20	9	1729.51
		HYSAT	106	1346.47	86	1344.97	20	1.49	–	–
		MATHSAT	104	6272.87	84	5728.48	20	544.39	–	–
	HN-20_ON-1 (120)	YICES	115	6014.31	80	3563.78	35	2450.53	23	4274.22
		HYSAT	90	2651.61	67	1263.40	23	1388.22	–	–
		MATHSAT	76	6420.13	56	5796.93	20	623.19	–	–
	HN-20_ON-6 (120)	YICES	105	6519.13	85	5620.70	20	898.43	11	3238.09
		HYSAT	91	1852.71	72	1850.86	19	1.85	1	3.03
		MATHSAT	87	7774.15	67	7085.04	20	689.11	–	–

Table 2. Solver-centric view of the results. Columns “Suite” and “Family” report suite and family name of the encodings, respectively. The remainder of the table is organized similarly to Table 1, with the exception of column (“Unique”), that shows data about uniquely solved encodings.

Family	Overall		Time	Hardness			Family	Overall		Time	Hardness		
	N	#		EA	ME	MH		N	#		EA	ME	MH
GLOBAL_HN-5_ON-1	20	20	23.89	12	8	–	LOCAL_HN-5_ON-1	120	120	823.53	97	10	13
GLOBAL_HN-5_ON-6	20	20	28.91	19	1	–	LOCAL_HN-5_ON-6	120	120	20.57	113	7	–
GLOBAL_HN-10_ON-1	20	20	744.19	9	7	4	LOCAL_HN-10_ON-1	120	120	2563.94	90	8	22
GLOBAL_HN-10_ON-6	20	20	345.42	14	5	1	LOCAL_HN-10_ON-6	120	120	2137.99	99	12	9
GLOBAL_HN-20_ON-1	20	20	906.36	17	1	2	LOCAL_HN-20_ON-1	120	115	4811.98	74	18	23
GLOBAL_HN-20_ON-6	20	20	827.05	9	9	2	LOCAL_HN-20_ON-6	120	106	4484.83	83	11	12

Table 3. Encoding-centric view of the results. The table consists of seven columns where for each family of encodings we report the name of the family in alphabetical order (column “Family”), the number of encodings included in the family, and the number of encodings solved (group “Overall”, columns “N”, “#”, respectively), the CPU time taken to solve the encodings (column “Time”), the number of easy, medium and medium-hard encodings (group “Hardness”, columns “EA”, “ME”, “MH”).

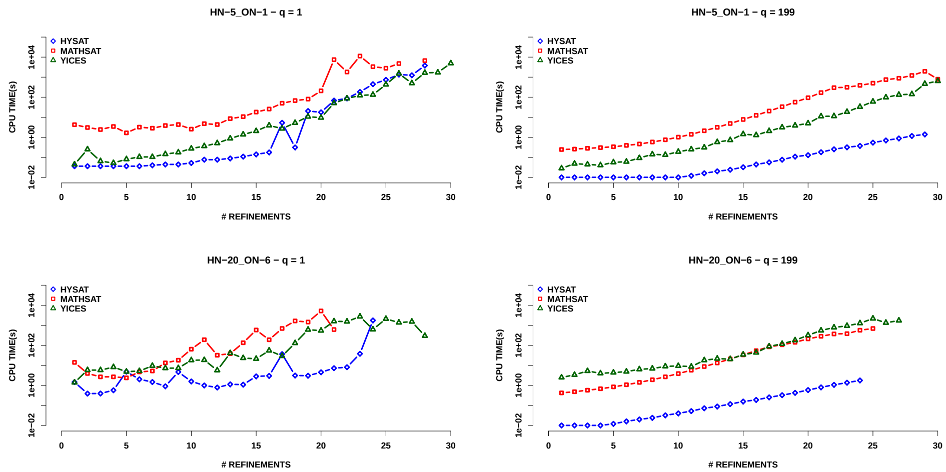


Fig. 3. Scalability test on the evaluated solvers. For each plot, in x axis is shown the refinement step, while in the y axis (in logarithmic scale) the related CPU time (in seconds). HYSAT performance is depicted by blue diamonds, while MATHSAT and YICES results are denoted by red boxes and green triangles, respectively. Plots in the same column are related to encodings having the same satisfiability result, i.e. SAT (left-most column) and UNSAT (right-most column). Plots in the same row are related to encodings having the same MLP architecture, i.e. HN-5_ON-1 (top) and HN-20_ON-6 (bottom).

(out of 120), while MATHSAT and HYSAT 29 and 14 times, respectively. If we consider now the suite LOCAL, the picture is quite different: HYSAT contributed 509 times, while YICES and MATHSAT 188 and 4 times, respectively. Considering all 821 solved encodings, we report that HYSAT was the main contributor (64%), despite the fact that it is not the best solver in terms of total amount of solved encodings.

Our next experiment aims to draw some conclusions about the scalability of the evaluated solvers. In order to do that, we compute a pool of encodings satisfying the following desiderata:

1. Consider values of the abstraction parameter p which correspond to increasingly fine-grained abstractions.
2. Consider different MLP size in terms of hidden neurons.
3. Consider the satisfiability result of the computed encodings.

To cope with (1), we generate encodings related to 30 refinement steps. To take care the potentially increasing difficulty of such encodings, we set the time limit to 4 CPU hours. In order to satisfy desiderata (2), we compute encodings both considering HN-5_ON-1 and HN-20_ON-6 MLP architectures. Finally, in order to cover (3), we focus on the suite LOCAL, selecting the encodings related to $q = 1$, and $q = 199$. The former encodings are almost always satisfiable, i.e., an abstract counterexamples is easily found, and, conversely, the latter encodings are almost always unsatisfiable.

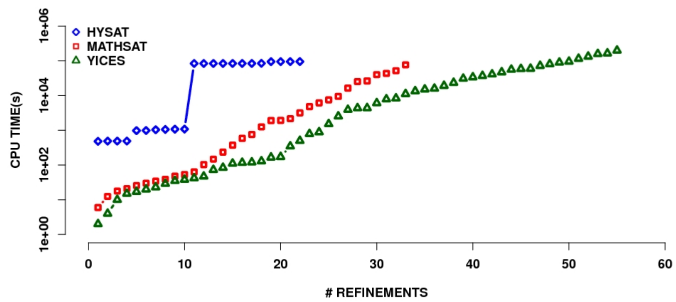


Fig. 4. CPU cumulative time (y axis) vs. number of refinement steps (x axis) of NEVER featuring the evaluated solvers as back-engine. The plot is organized similarly to the plots in Figure 3.

As a result of the selection above, we obtain 4 groups of encodings, and Figure 3 shows the results of experimenting with them. Looking at Figure 3 (top-left), we can see that HYSAT is the best solver along the first 17 refinement steps. After this point, its performance is comparable to YICES, but with the noticeable difference that the latter is able to solve all the encodings, while HYSAT exhausts CPU time resources trying to solve the two encodings having the smallest value of p. The CPU time spent by MATHSATon each of the first 24 refinement steps is at least one order of magnitude higher than HYSAT and YICES. Considering now the same safety problem, but related to a larger MLP architecture, we can see a different picture. From Figure 3 (bottom-left), we can see that no solver is able to solve all the encodings within 4 CPU hours. In particular, MATHSAT stops at the 21st step (out of 30), while YICES is able to solve all encodings but the last two. While the performances of MATHSAT and YICES seems to have a smooth increasing trend, HYSAT is less predictable: In the first 22 refinement steps it is up to one order of magnitude faster than YICES– with the noticeable exception of four "peaks" – and for the following two steps it is two order of magnitude slower than YICES.

Considering now the plots in Figure 3 (right), we can see that the trend in solver's performance is much smoother than the plots in Figure 3 (left). Looking at Figure 3 (top-right), we can see that, excluding the last encoding, HYSAT is one order of magnitude faster than YICES, that in turn is one order of magnitude faster that MATHSAT. Looking now at the last plot, we can see that we have two main differences with respect to the picture resulting from the previous plot. First, the encodings are more challenging, because no solver was able to solve all the pool within the CPU time limit. Second, there is no noticeable difference – excluding the last two solved encodings – between MATHSAT and YICES.

Our last experiment concerns the analysis of the performance of our tool NEVER [20] equipped with the various solvers back-ends. We experiment with a local safety problem with $k = 75$, $p = 0.5$, and $r = 1.1$ about an MLP having a HN-20_ON-6 architecture. In Figure 4 we report the results of such experiment. Also if NEVER was not able to conclude about the safety of the considered MLP because all solvers exhaust their memory resources, looking at the figure we can see that YICES clearly outperforms both MATHSAT and HYSAT. YICES allowed NEVER to refine 54 times, while MATHSAT and HYSAT stopped to 32 and 21 refinements, respectively. Concerning the cumulative CPU time, performances of MATHSAT and YICES are in the same ballpark until the 12th step, and they increase smoothly until the end of the computation. On the other hand, if we look at HYSAT performance, we can see that it is very close to be constant, with the noticeable exception of two peaks: The first one (between step 3 and 4) is small, and the second one (between step 9 and 10) implies a two orders of magnitude jump in the cumulative CPU time. In this problem, HYSAT shows the same behaviour shown in Figure 3 (bottom-left).

Summing up, our experiments show that such encodings are challenging for SMT solvers, in particular from a scalability point of view. Despite this fact, we can conclude that YICES seems the "best candidate" to be used as back-engine of our tool NEVER, mainly because it scales better for encodings related to small values of p. As future work, we plan to investigate two main directions. The first one is a deep investigation about the parameters of the SMT solvers, in particular referring to some recent work about Automated Configuration – see, e.g. [17]. The second one, motivated by the fact that there is no clear "winner" among the solvers, especially for not-so-small values of p, is to characterize SMT instances using quantitative features and try a multi-engine approach in the spirit of [19].

References

1. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *International Journal on Software Tools for Technology Transfer (STTT)*, 11(1):69–83, 2009.
2. C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification*, pages 20–23. Springer, 2005.
3. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. *Handbook of Satisfiability, IOS Press, Amsterdam*, pages 825–885, 2009.
4. C. Barrett and C. Tinelli. CVC3. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, 2007.
5. C. Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 2009.
6. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *Proceedings of the 20th international conference on Computer Aided Verification*, pages 299–303. Springer-Verlag, 2008.
7. D. Cok. The SMT-LIBv2 Language and Tools: A Tutorial, 2011. Available from `http://www.grammatech.com/resources/smt/`.
8. R. DeLine and K. Leino. BoogiePL: A typed procedural language for checking object-oriented programs. *Technical Report MSR-TR-2005-70*, 2005.
9. B. Dutertre and L. De Moura. A fast linear-arithmetic solver for DPLL (T). In *Computer Aided Verification*, pages 81–94. Springer, 2006.

10. J. Filliâtre and C. Marché. The Why/Krakatoa/Caduceus platform for deductive program verification. In *Proceedings of the 19th international conference on Computer aided verification*, pages 173–177. Springer-Verlag, 2007.
11. P. Fontaine, J. Marion, S. Merz, L. Nieto, and A. Tiu. Expressiveness+ automation+ soundness: Towards combining SMT solvers and interactive proof assistants. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–181, 2006.
12. M. Franzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.
13. M. Fumagalli, A. Gijsberts, S. Ivaldi, L. Jamone, G. Metta, L. Natale, F. Nori, and G. Sandini. Learning to Exploit Proximal Force Sensing: a Comparison Approach. *From Motor Learning to Interaction Learning in Robots*, pages 149–167, 2010.
14. S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall, 2008.
15. T. Hoang and N. Binh. Extending CREST with Multiple SMT Solvers and Real Arithmetic. In *Knowledge and Systems Engineering (KSE), Second International Conference on*, pages 183–187. IEEE, 2010.
16. K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
17. F. Hutter, H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 186–202, 2010.
18. L. Jamone, G. Metta, F. Nori, and G. Sandini. James: A humanoid robot acting over an unstructured world. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 143–150. IEEE, 2007.
19. L. Pulina and A. Tacchella. A self-adaptive multi-engine solver for quantified Boolean formulas. *Constraints*, 14(1):80–116, 2009.
20. L. Pulina and A. Tacchella. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *22nd International Conference on Computer Aided Verification (CAV 2010)*, volume 6174 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2010.
21. L. Pulina and A. Tacchella. Abstraction-refinement verification of adaptive parameter estimation in robot control. Technical report, University of Genoa, 2011. Full text available at `www.mind-lab.it/~luca/SMT-ANN`.
22. S. Ray. Connecting External Deduction Tools with ACL2. *Scalable Techniques for Formal Verification*, pages 195–216, 2010.
23. J. Schumann and Y. Liu, editors. *Applications of Neural Networks in High Assurance Systems*, volume 268 of *Studies in Computational Intelligence*. Springer, 2010.
24. A. Solar-Lezama, C. Jones, and R. Bodik. Sketching concurrent data structures. In *2008 ACM SIGPLAN conference on Programming language design and implementation*, pages 136–148. ACM, 2008.
25. M. Vechev, E. Yahav, and G. G. Yorsh. Abstraction-guided synthesis of synchronization. In *37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 327–338. ACM, 2010.

View publication stats