Towards Verification of Artificial Neural Networks [*]

Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker
Chair of Computer Architecture
Albert-Ludwigs-Universität Freiburg im Breisgau, Germany
{scheibler,winterel,wimmer,becker}@informatik.uni-freiburg.de

Abstract

We consider the safety verification of controllers obtained via machine learning. This is an important problem as the employed machine learning techniques work well in practice, but cannot guarantee safety of the produced controller, which is typically represented as an artificial neural network. Nevertheless, such methods are used in safety-critical environments.

In this paper we take a typical control problem, namely the Cart Pole System (a. k. a. inverted pendulum), and a model of its physical environment and study safety verification of this system. To do so, we use bounded model checking (BMC). The created formulas are solved with the SMT-solver iSAT3. We examine the problems that occur during solving these formulas and show that extending the solver by special deduction routines can reduce both memory consumption and computation time on such instances significantly. This constitutes a first step towards verification of machine-learned controllers, but a lot of challenges remain.

1. Introduction

Machine learning [MRT12] is a powerful method to derive a controller automatically for systems which are hard to handle otherwise: There may either be no closed-form description of the environment, or the optimal behavior cannot be determined, or the system has to adapt itself to changing environments. The user specifies which variables describe the state space of the system and the possible actions that can be used to control it. For so-called reinforcement learning [SB98], additionally a reward (or, equivalently, cost) function is used which measures how desirable a certain state is. Machine learning methods then explore the state space using the available actions. The obtained information about the system is used to derive a controller which maximizes the reward. Since exploring the whole state space is typically infeasible, the learning algorithms generalize the information obtained by exploration to unexplored parts. When the learning phase is completed, the result is a controller that is typically represented as an artificial neural network.

Such learned controllers are used successfully in many scenarios. Many of them are safety-critical; examples are controlling autonomous driving [For02], deep-brain stimulation against epileptic

[*] This work has been partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (DFG, SFB/TR 14 AVACS, `http://www.avacs.org/`) and by the Cluster of Excellence BrainLinks-BrainTools (DFG, grant number EXC 1086, `http://www.brainlinks-braintools.uni-freiburg.de/`)

seizures [KWB$^+$14, BCP08, PMG$^+$14], and robots [KCC13]. However, the learning algorithms in general cannot provide any guarantees regarding the safety of the controller. Therefore, after learning, a verification step should be included, which checks that the controller actually prevents the system from running into an unsafe state. Nevertheless, only few works have addressed this issue so far [Bor14, Gal14]. In this paper we present our first attempts to improve this situation.

We apply bounded model checking [BCC$^+$03] to a typical control problem, namely the Cart Pole System, also known as inverted pendulum [BSA83]: a cart moves left and right on a rail and has to balance a loosely attached pole on top of it. Since both the neural network representing the controller and the description of the physical environment are non-linear (they contain transcendental functions like exp and cos), an SMT solver is needed which can handle such constraints. The state-of-the-art SMT solver iSAT3 [SKB13] is one of the few solvers which can cope with transcendental functions. It is based on interval deductions to exclude variable assignments which cannot satisfy the formula, and is tightly integrated with a modern SAT-solver to handle the Boolean structure of the formula.

We first present the translation of the closed-loop system into an SMT formula by discretizing the differential equation that describes the behavior of the cart pole using a standard Runge-Kutta method. As experiments confirm, the problem of this approach is that for formula generation the loops typically executed during the discretization have to be unrolled, leading to a large number of auxiliary variables and constraints and consequently to high memory requirements and long computation times. We solve this problem by extending the solver core with problem-specific operations. Experiments confirm that this reduces the memory consumption significantly as well as the computation times. However, this constitutes only the first step towards an efficient verification of such controllers as in spite of our improvements the analysis is restricted to rather short time horizons.

Structure of the paper In the following section, we introduce the Cart Pole System, define artificial neural networks, and sketch how iSAT3 works. In Section 3 we describe the translation of the controlled Cart Pole System into the iSAT3 formalism and analyze the problems with this approach. In Section 4 we propose the introduction of user-defined operations to speed up the solution of the BMC formulas. Section 5 presents our experimental results, and the paper finishes in Section 6 with a conclusion and pointers to future work.

2. Preliminaries

In this section we first describe the system we want to analyze, namely a Cart Pole System, which is a typical example from control theory and machine learning. We then define multi-layer perceptrons; they are the variant of artificial neural networks that we use to represent controllers. Finally, we describe the core mechanisms underlying the SMT-solver iSAT3 used in our studies for bounded model checking.

2.1. Cart Pole System

A standard benchmark for machine learning is the Cart Pole system (or inverted pendulum) – a cart that is attached to a rail moves left and right, trying to balance a loosely attached pole at right angle on top of it. Several versions of the model exist, with more complex ones having the start position with the pole hanging upside down, requiring the algorithm to first perform a ‘swing up’ move

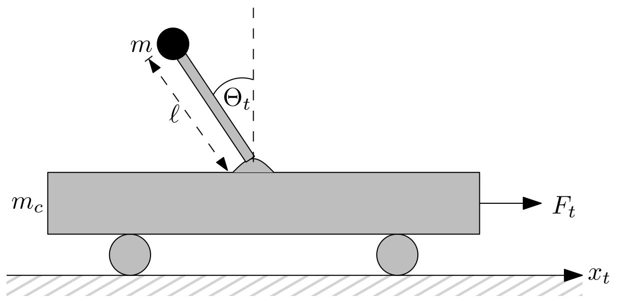

Figure 1: Visualization of the Cart Pole Model

before starting the balancing phase. The model used here has the pole starting in an almost-upright position and requires the cart to move to a certain area of the rail without losing balance of the pole.

The exact movements of the cart and pole are described by the following ordinary differential equations, taken from [BSA83]. They describe the evolution of the current position x_t of the cart as well as the angle Θ_t of the pole, depending on the current time $t \geq 0$ (cf. Figure 1):

$$\Theta_t = \frac{g \cdot \sin\Theta_t + \cos\Theta_t \cdot [\frac{-F_t - m \cdot \frac{\ell}{2} \cdot \dot{\Theta}_t^2 \cdot \sin\Theta_t + \mu_c \cdot \operatorname{sgn}(\dot{x}_t)}{m_c + m}] - \frac{\mu_p \cdot \dot{\Theta}_t}{m \cdot \frac{\ell}{2}}}{\frac{\ell}{2}[\frac{4}{3} - \frac{m \cdot \cos^2(\Theta_t)}{m_c + m}]}, \tag{1a}$$

$$\ddot{x}_t = \frac{F_t + m \cdot \frac{\ell}{2} \cdot [\dot{\Theta}_t^2 \cdot \sin(\Theta_t) - \ddot{\Theta}_t \cdot \cos(\Theta_t) - \mu_c \cdot \operatorname{sgn}(\dot{x}_t)]}{m_c + m}, \tag{1b}$$

The constant $g \approx 9.81 \frac{\mathrm{m}}{\mathrm{s}^2}$ is the gravity acceleration of the earth, m_c and m are the masses of the cart and pole, resp., ℓ is the length of the pole, μ_c and μ_p are the coefficients of friction for the cart on the rails and the pole on the cart, and F_t is the force applied to the cart's center at time t. The value of the external force F_t is the parameter which can be used to control the system. The current state of the system is given by the vector $(\Theta_t, \dot{\Theta}_t, x_t, \dot{x}_t)$.

2.2. Artificial Neural Networks

The external force F_t in our cart pole example is controlled by an artificial neural network. It obtains the current state of the system and returns the value of the force to be applied. The particular type of neural network we use in this paper is a multi-layer perceptron:

Definition 1 (Multi-layer perceptron) *A* multi-layer perceptron *is a weighted directed acyclic graph $G = (V, E, w)$ with V being a finite set of nodes, $E \subseteq V \times V$ the set of edges, and $w : E \to \mathbb{R}^{\geq 0}$ the edge-weight function. Nodes $I \subseteq V$ without incoming edges are called* input nodes*, nodes $O \subseteq V$ without outgoing edges* output nodes.

For a multi-layer perceptron $G = (V, E, w)$ and a node $v \in V$ we denote the set of predecessor nodes of v by $\mathrm{pre}(v)$ and the set of successor nodes by $\mathrm{succ}(v)$. Given an input assignment $\mathrm{val}(v)$ for the input nodes $v \in I$, we can assign a value to each node of the network by traversing it in topological order, using the activation function of the nodes:

$$\mathrm{val}(v) = \frac{1}{1 + \exp\Big(\sum_{u \in \mathrm{pre}(v)} w(u, v) \cdot \mathrm{val}(u)\Big)} \quad \text{for } v \in V \setminus I. \tag{2}$$

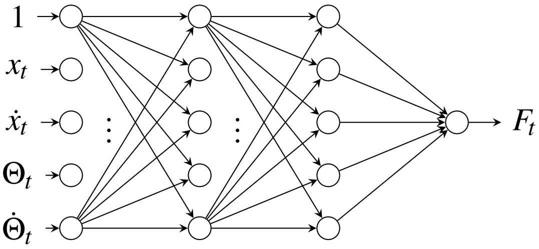

Figure 2: Visualization of a simple neural network with two hidden layers as used in this work (the edge weights have been omitted for the sake of readability).

Other activation functions are possible. The output of the network is then given by the values assigned to the output nodes O. Figure 2 shows an illustration of the neural network used to control the Cart Poly System in our experiments.

Machine learning trains a multi-layer perceptron by adapting the weight function w in order to minimize the difference between the actual output of the perceptron network and its expected output. Standard algorithms to do so are the back-propagation algorithm [RHW86] and reinforcement learning [SB98]. We omit the details about the learning, because in our scenario we are given a readily trained multi-layer perceptron which computes the external force applied to the cart (normalized to the interval $[0,1]$). For more details we refer the reader to the machine learning literature, e. g., [Bis96].

2.3. SAT and SMT

The *propositional satisfiability problem* (SAT) addresses the question whether there exists an assignment for the variables of a propositional formula such that the formula is satisfied (evaluates to true). Programs handling this kind of problems are called SAT-solvers. Usually, SAT-solvers expect formulas to be given in *conjunctive normal form* (CNF). A CNF consists of a conjunction of clauses. Each clause is a disjunction of literals and each literal is a Boolean variable or its negation. The Tseitin-transformation [Tse68] allows the conversion of arbitrary propositional formulas into CNF.

In [DLL62] the DPLL method was proposed. In DPLL the search for a satisfying assignment exploits that a CNF is satisfied if and only if all clauses are satisfied. If during the search process a (partial) assignment of the variables results in an unsatisfied clause (a clause with all literals being false), backtracking is performed in order to enter a different part of the search space. Additionally, *Boolean constraint propagation* (BCP) is used to detect implied assignments. Everytime a clause with n literals contains $n-1$ literals being false, the remaining literal has to be true in order to retain a chance to satisfy the formula. Moreover, today's SAT-solvers do non-chronological backtracking and add conflict clauses to the formula to prune the search space even further – so-called *conflict-driven clause learning* (CDCL) [SS96].

SAT Modulo Theories (SMT) allows Boolean combinations of so-called theory atoms. A theory atom is an arithmetic expression which can be evaluated to true or false, e. g., $(x+y<10)$. Depending on the underlying theory such an expression may contain linear and/or non-linear arithmetic. When doing *eager SMT* a given Boolean combination of theory atoms is completely translated into an equi-satisfiable propositional formula. This formula is then passed to a SAT-solver. In many cases such an approach is either infeasible (the CNF would become to large or the translation itself would take to long) or not possible at all due to properties of the underlying theory. Therefore, *lazy SMT* is mostly used. In lazy SMT, each theory atom is abstracted by a literal, resulting in a Boolean abstraction of the original SMT formula. This abstraction is given to a SAT-solver. If the abstraction is unsatisfiable, then the original SMT formula is also unsatisfiable. If the SAT-solver returns a satisfying assignment, we employ a theory solver to check if the selected truth values for the theory atoms are indeed a solution. If this is not the case, then a conflict clause is added to the Boolean abstraction which contains the literals of the conflicting theory atoms. Thus, the Boolean abstraction is lazily refined with additional knowledge. This scheme is also abbreviated as DPLL(T) or CDCL(T) – with T being the theory used.

2.4. The iSAT algorithm, iSAT3

The iSAT algorithm [FHT$^+$07, Her11] uses *interval constraint propagation* (ICP, see, e. g., [BG06]) to check the consistency of theory atoms. But in contrast to classical lazy SMT there is no clear separation between the theory and SAT-solver part; instead ICP is tightly integrated into the CDCL framework. This deep integration allows to share the common core of the search algorithms between the propositional and the theory-related part of the solver. Therefore the iSAT algorithm goes beyond CDCL(ICP).

The iSAT algorithm allows theory atoms to contain linear and non-linear arithmetic as well as transcendental functions, e. g., $(x+y^2=z^2)$ or $(\sqrt{x}+\cos z<e^y)$. Furthermore, three variable types are supported: Boolean, integer-, and real-valued variables. The latter two types have to be declared with a bounded initial interval. Before solving a given formula, each theory atom is decomposed into *simple bounds* and *primitive constraints* by introducing fresh auxiliary variables similar to the Tseitin-transformation. A simple bound imposes a lower or upper bound to a variable – either strict, e. g., $(x<1)$ or non-strict $(x\leq 1)$. Since we store these bounds as floating-point numbers, outward rounding is used to get a safe interval enclosure. Furthermore, this decomposition enables us to rely on a fixed set of primitive constraints which are used to apply ICP later on. A primitive constraint consists of a unary or binary operator and the associated variables, e. g., $(h=\sin x)$ or $(h=x+y)$. Applying ICP means, a so-called contractor is used to narrow the intervals of the variables occurring in each primitive constraint in order to exclude definitive non-solutions[1].

The CDCL framework consists of three building blocks: (1) decide, (2) propagate and (3) resolve conflicts. The iSAT-algorithm builds on CDCL and extends it for the operation on integer- and real-valued intervals in addition to Boolean variables. Firstly, deciding an integer- or real-valued variable corresponds to splitting the interval of the variable and selecting the lower or upper part. Secondly, in addition to BCP for Boolean variables, ICP is used to narrow the intervals of the variables occurring in primitive constraints. Similar to BCP we use an implication queue for ICP to keep track of all changed variables – primitive constraints containing these variables will be processed by ICP. The propagation phase is finished if the BCP and ICP implication queues are empty. And finally, conflict resolution has additionally to cope with conflicts occurring in

[1] For instance, if we have the primitive constraint $(h=x+y)$ and $h\in[1,9]$, $x\in[1,3]$ and $z\in[4,10]$ then it is possible to contract the interval of h. This is done by calculating $x+y$ with interval arithmetic and intersecting the result with the current interval of h. In this example $x+y$ would yield $[5,13]$ resulting in $[5,9]$ as the new interval for h. In order to contract the intervals for x and y the primitive constraint is redirected accordingly and a similar calculation is done. In this example x stays unchanged and y could be shrunken to $[4,8]$.

inconsistent primitive constraints. A primitive constraint is consistent if the interval on the left-hand side has a non-empty intersection with the resulting interval of the right-hand side – otherwise the primitive constraint is inconsistent. For instance, $(h = x + y)$ with $h \in [1,1]$ and $x, y \in [0,1]$ is consistent while it would be inconsistent with $h \in [3,3]$, $x \in [0,1]$ and $y \in [-1,1]$.

In order to prevent infinite propagation sequences, we check for every newly deduced bound for a variable if it has reasonable progress compared to the current bound of that variable (so-called *minimum progress*) – if the progress is too small the new bound will be discarded. Furthermore, interval splits are only performed if the considered interval is larger than the so-called *minimal splitting width*. Due to the undecidability of the supported logic, the iSAT-algorithm may terminate with an inconclusive answer. That means an interval box (so-called *candidate solution*) will be returned which may contain a solution. To be precise, all primitive constraints have solutions in this interval box (i. e., all primitive constraints are consistent), but it is not guaranteed that all primitive constraints have a common solution point in that interval box.

In this paper we focus on iSAT3 [SKB13] – the third implementation of the iSAT-algorithm. iSAT3 and its predecessors HySAT [HEFT08] and iSAT [EKKT08] share the same major core principles of tightly integrating ICP into the CDCL framework – but they differ on a more technical level in the handling of the simple bounds. While the core solvers of HySAT and iSAT operate on simple bounds, the core of iSAT3 uses literals. This results in a different way of doing decisions: HySAT and iSAT always perform interval splits when doing a decision – in contrast iSAT3 first decides existing literals, before splitting intervals.

3. Translation into the iSAT3 input language

In a first attempt to verify properties of the Cart Pole System and its neural network controller, we translated both components into the input language of iSAT3. The translation of the neural network was directly possible, because all needed operations are natively supported by iSAT3.

For the translation of the Cart Pole System, we had to abstract the differential equations as iSAT3 does not support these. However, we had access to a C++-implementation which was also used to train the neural networks that were provided to us. In this implementation, the differential equations (1a) and (1b) have been iteratively approximated using the Runge-Kutta-methods [But87] – resulting in two nested loops (as seen in Algorithm 1 and Algorithm 2) computing the next-step values of the parameters, given a fixed step size δ_t (20 ms in all examples used).
Since iSAT3 only supports total operations, we had to rearrange some constraints, e. g.,

$$\ddot{x}_{new} = \frac{(h - m \cdot l \cdot \ddot{\Theta} \cdot \cos(\Theta)) - fric_{cart}}{m_c + m} \tag{3}$$

is translated into iSAT3 syntax as follows:

```
x_pp' * (m_c + m) = h - m * l * Theta_pp' * cos(Theta) - fric_cart;
```

Primed variables (e.g. `x_pp'`) indicate the values computed during the current step and unprimed variables refer to the values computed in the last step. Furthermore, since iSAT3 does not support any loops, we had to unroll the ten iterations of the for-loop which lead to a large instance with over 400 constraints.

Algorithm 1 Auxiliary function, approximating the second-order derivatives in a given state.

```
**2nd_deriv**($x$, $\dot{x}$, $\Theta$, $\dot{\Theta}$, $F^t$)
**begin**
    $h = F^t + m \cdot \ell \cdot \dot{\Theta}^2 \cdot \sin(\Theta)$                                  (1)
    $fric_{pole} = \frac{\mu_p \cdot \dot{\Theta}}{m \cdot \ell}$                                      (2)
    $\ddot{\Theta}_{new} = \frac{((m+m_c) \cdot g \cdot \sin(\Theta) - \cos(\Theta) \cdot h + \cos(\Theta) \cdot \mu_c \cdot \mathrm{sgn}(\dot{x})) - fric_{pole}}{\frac{4}{3}(m+m_c) \cdot \ell \cdot \cos^2(\Theta)}$   (3)
    $fric_{cart} = \mu_c \cdot \mathrm{sgn}(\dot{x})$                                          (4)
    $\ddot{x}_{new} = \frac{(h - m \cdot \ell \cdot \ddot{\Theta} \cdot \cos(\Theta)) - fric_{cart}}{m_c + m}$            (5)
    **return** $(\ddot{x}_{new}, \ddot{\Theta}_{new})$                                         (6)
**end**
```

4. Special Deduction Functions

While the translation of the Cart Pole System and the neural network to the input language of iSAT3 was easily doable, we observed a poor performance of the solver on these translated instances. This is mainly due to two problems:

1. As mentioned in Section 2, newly deduced bounds need to have a reasonable progress compared to the current bounds (minimum progress) in order to prevent infinite propagation sequences. Of course, every discarded bound leads to a coarser interval. This could result in additional conflicts occurring later on, forcing the solver to do further decisions and slowing down the overall performance. To mitigate this problem, one could decrease the minimum progress in order to prevent the solver from discarding bounds. But this comes at the cost of millions of new literals (each representing a bound) to be present in the solver and blowing up the implication graph. This could be tackled with implication graph compression as proposed in [SB14] – nonetheless, the following problem would be still present.

2. An implication queue (IQ) is used to keep track of all variables changed. For every variable in the queue we have to apply ICP to all primitive constraints containing this variable. This may result in further changed variables. Depending on the structure of the original constraints, variables could be enqueued in a bad order – i. e., an order where a variable is deduced multiple times whereas a different order would require only one deduction per variable. For example if we have the primitive constraints $C_1 : (h_1 = x + h_2)$, $C_2 : (h_2 = x^2)$. Now assume x is changed and enqueued to the IQ. When x is dequeued from the IQ, we examine the constraint C_1. This results in new bounds for h_1 – hence h_1 is enqueued to the IQ. When examining C_2 we get stronger bounds for h_2 as well – therefore h_2 is enqueued too. Now we would dequeue h_1 from the IQ and proceed with all primitive constraints containing this variable. If we now dequeue h_2 from the IQ we have to examine C_1 once more – yielding again new bounds for h_1. With a different ordering (h_2 before h_1) we would consider h_1 only once. This example illustrates that the order of the IQ impacts how many deductions are made. Since the primitive constraints are deduced in all directions, it is not sufficient to just apply a topological sorting. Unfortunately, the translated instances contain hundreds of constraints with a structure as shown in the example. That means the overall deduction process has to deal with a huge overhead.

Algorithm 2 Given the current state, `NextState` computes the successor state after δ_t time steps.

```
NextState($x^0$, $\dot{x}^0$, $\Theta^0$, $\dot{\Theta}^0$, $F$, $\varepsilon = \frac{\delta_t}{10}$)
begin
    for $i = 1$ to $10$ do
        $(\ddot{x}^i_1, \ddot{\Theta}^i_1) = 2nd\_deriv(x^{i-1}, \dot{x}^{i-1}, \Theta^{i-1}, \dot{\Theta}^{i-1}, F^t)$
        $x^i_1 = x^{i-1} + \frac{\varepsilon}{2} \cdot \dot{x}^{i-1}$
        $\dot{x}^i_1 = \dot{x}^{i-1} + \frac{\varepsilon}{2} \cdot \ddot{x}^i_1$
        $\Theta^i_1 = \Theta^{i-1} + \frac{\varepsilon}{2} \cdot \dot{\Theta}^{i-1}$
        $\dot{\Theta}^i_1 = \dot{\Theta}^{i-1} + \frac{\varepsilon}{2} \cdot \ddot{\Theta}^i_1$

        $(\ddot{x}^i_2, \ddot{\Theta}^i_2) = 2nd\_deriv(x^i_1, \dot{x}^i_1, \Theta^i_1, \dot{\Theta}^i_1, F^t)$
        $x^i_2 = x^{i-1} + \frac{\varepsilon}{2} \cdot \dot{x}^i_1$
        $\dot{x}^i_2 = \dot{x}^{i-1} + \frac{\varepsilon}{2} \cdot \ddot{x}^i_2$
        $\Theta^i_2 = \Theta^{i-1} + \frac{\varepsilon}{2} \cdot \dot{\Theta}^i_1$
        $\dot{\Theta}^i_2 = \dot{\Theta}^{i-1} + \frac{\varepsilon}{2} \cdot \ddot{\Theta}^i_2$

        $(\ddot{x}^i_3, \ddot{\Theta}^i_3) = 2nd\_deriv(x^i_2, \dot{x}^i_2, \Theta^i_2, \dot{\Theta}^i_2, F^t)$
        $x^i_3 = x^{i-1} + \varepsilon \cdot \dot{x}^i_2$
        $\dot{x}^i_3 = \dot{x}^{i-1} + \varepsilon \cdot \ddot{x}^i_3$
        $\Theta^i_3 = \Theta^{i-1} + \varepsilon \cdot \dot{\Theta}^i_2$
        $\dot{\Theta}^i_3 = \dot{\Theta}^{i-1} + \varepsilon \cdot \ddot{\Theta}^i_3$

        $(\ddot{x}^i, \ddot{\Theta}^i) = 2nd\_deriv(x^i_3, \dot{x}^i_3, \Theta^i_3, \dot{\Theta}^i_3, F^t)$
        $x^i = x^{i-1} + \frac{\varepsilon}{6} \cdot (\dot{x}^{i-1} + 2 \cdot (\dot{x}^i_1 + \dot{x}^i_2 + \dot{x}^i_3))$
        $\dot{x}^i = \dot{x}^{i-1} + \frac{\varepsilon}{6} \cdot (\ddot{x}^i_1 + 2 \cdot (\ddot{x}^i_2 + \ddot{x}^i_3 + \ddot{x}^i))$
        $\Theta^i = \Theta^{i-1} + \frac{\varepsilon}{6} \cdot (\dot{\Theta}^{i-1} + 2 \cdot (\dot{\Theta}^i_1 + \dot{\Theta}^i_2 + \dot{\Theta}^i_3))$
        $\dot{\Theta}^i = \dot{\Theta}^{i-1} + \frac{\varepsilon}{6} \cdot (\ddot{\Theta}^i_1 + 2 \cdot (\ddot{\Theta}^i_2 + \ddot{\Theta}^i_3 + \ddot{\Theta}^i))$
    end for
    return $(x^{10}, \dot{x}^{10}, \Theta^{10}, \dot{\Theta}^{10})$
end
```

Furthermore, if we look closer to the problem, then it is not needed to expose the whole internal structure of the calculations of the Cart Pole System and the neural network as constraints to the solver. Instead the solver should concentrate on these five important variables per unrolling depth: the position x_t and velocity $\dot{x}_t$ of the cart, the angle Θ_t and the velocity $\dot{\Theta}_t$ of the pole and the action F_t determined by the neural network. Everything else is an intermediate result – used to calculate one of these five values. Therefore, we decided to implement dedicated deduction functions for the neural network and the Cart Pole System. This also eliminates the problem of discarded bounds and a badly ordered implication queue.

We have added five new functions to the iSAT3 input language: `nncart_pole_angle`, `nncart_pole_v`, `nncart_cart_pos`, `nncart_cart_v`, and `nncart_action`. The first four provide the computations needed for the next state of the Cart Pole System while the last function returns the action computed by the neural network.

We expect that many other applications can benefit from such solver extensions: They can be applied whenever a complicated function has to be expressed as a (large) set of simpler constraints and auxiliary variables which are not referenced outside the function computation.

5. Experimental Results

We have run our experiments on a computer with 4 Quad-Core AMD Opteron$^{\text{TM}}$ 8356 processors, running at 2.3 GHz with 64 GB DDR2 RAM. We have set a time limit of 900 seconds and a memory limit of 8000 MB. Instances of the *simple_property*-series represent verification of a simple property, which asks for the states reachable from a given input range and mainly involves deductions from the input nodes to the output nodes, whereas instances of the *complex_property*-series verify a more advanced property of the system at hand, which checks reachability of a certain set of states and requires many deductions from output values to input values. The number appended represents to which depth the solver has to advance in order to verify the given property. We encoded the instances in two ways: (1) as described in Section 3 as a large number of constraints and (2) exploiting the special deduction routines presented in Section 4.

Table 1 illustrates that the special deduction routines boost the performance of the solver on this kind of instances by several orders of magnitude – while the plain translation can only handle up to 3 unrollings, the special deduction routines perform well even for hundreds of steps. Furthermore, when the special deduction functions are used, the memory footprint is reduced significantly – the number of variables indicates this. Every simple bound is represented as a literal in the solver and therefore increases the number of variables. The more real-valued variables are declared in the instance, the more simple bounds will be created during solving. Since the translated instances use hundreds of constraints and declare just as many additional real-valued variables, it is not surprising that they exceed the memory limit.

As mentioned in Section 2, we do outward rounding for every interval calculation to get a safe enclosure of all possible solutions. Even if intervals were tight at the beginning, they could "blow up" if a large number of interval operations is applied. This is even more the case with large unrolling depths – in particular when the exponential function is used as it is done for the neural network. Thus, the precision of a standard 64 bit double could be not enough to get reasonable results. Therefore, we exploit the ability of iSAT3 to use MPFR arithmetic with a user-specified mantissa width instead. The last column of Table 1 indicates which bitwidth was used to solve the instance (a hyphen stands for standard double precision).

6. Conclusion

We have considered the problem of safety verification for controllers obtained by machine learning techniques. Such controllers are typically represented using an artificial neural network. To ensure its safety, we applied bounded model checking using the solver iSAT3, which supports the required non-linear arithmetic.

We observed that even the smallest BMC instances could hardly be solved. One reason is the strong non-linearity and non-invertability of the neural network: It contains nested exponential functions, which makes in particular deducing input values from given output values very hard.

A further reason is the discretization of the environment and the decomposition of the constraints into triplets which introduces a large number of auxiliary variables and constraints, which prevent

Table 1: Comparing the approach presented in Section 3 with the approach from Section 4

Instance	Approach Section 3		Approach Section 4		Bit-Width
	Time	# Variables	Time	# Variables	
simple_property_01.hys	2.07	46907	0.34	145	–
simple_property_02.hys	95.10	116497	0.36	253	–
simple_property_03.hys	660.12	187804	0.40	361	–
simple_property_04.hys	timeout	258198	0.43	469	–
simple_property_05.hys	timeout	311794	0.47	577	–
simple_property_10.hys	memout (136.18)	16739789	0.70	1417	64
simple_property_50.hys	memout (129.33)	16776270	3.01	6937	128
simple_property_100.hys	memout (130.68)	16776270	5.94	13837	128
simple_property_500.hys	memout (291.31)	16725363	75.32	69037	1024
complex_property_01.hys	6.06	49189	0.41	192	–
complex_property_02.hys	78.04	107691	0.48	330	–
complex_property_03.hys	timeout	193361	0.98	477	–
complex_property_04.hys	timeout	258535	2.88	637	–
complex_property_05.hys	timeout	341588	3.80	775	–
complex_property_10.hys	timeout	725342	1.74	1434	–

an efficient deduction. To overcome this latter problem we have extended the solver with problem-specific deduction operators, allowing the user to directly describe the behavior of the neurons and the environment without encoding them using primitive constraints. Experiments show that this can give a speed-up of several orders of magnitude.

However, this still does not solve the problem of safety verification. In spite of the obtained speed-up only rather small unrolling depths and very basic safety properties can be handled. Therefore, we are investigating further optimizations like improved decision heuristics that are adapted to the verification task as well as alternative methods based on Lyapunov's stability theory.

References

[BCC^{+}03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58, 2003.

[BCP08] Douglas J. Bakkum, Zenas C. Chao, and Steve M. Potter. Spatio-temporal electrical stimuli shape behavior of an embodied cortical network in a goal-directed learning task. *J. Neural Eng.*, 5:310–323, 2008.

[BG06] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 571–603. 2006.

[Bis96] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Advanced Texts in Econometrics. Oxford University Press, 1996.

[Bor14] Luca Bortolussi. Machine learning meets stochastic model checking. In *Proc. of EPEW*, September 2014. (invited talk).

[BSA83] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 13(5):835–846, September/October 1983.

[But87] John C. Butcher. *The Numerical Analysis of Ordinary Differential Equations. Runge-Kutta and General Linear Methods.* (A Wiley-Interscience publication). Wiley, 1987.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[EKKT08] Andreas Eggers, Natalia Kalinnik, Stefan Kupferschmid, and Tino Teige. Challenges in constraint-based analysis of hybrid systems. In *Proc. of CSCLP*, volume 5655 of *LNCS*, pages 51–65. Springer, 2008.

[FHT$^+$07] Martin Fränzle, Christian Herde, Tino Teige, S. Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling, and Computation*, 1(3-4):209–236, 2007.

[For02] Jeffrey Roderick Norman Forbes. *Reinforcement Learning for Autonomous Vehicles*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2002.

[Gal14] Galois wins NASA award for formal methods in machine learning. `http://galois.com/blog/2010/12/galois-wins-nasa-award-for-formal-methods-in-machine-learning/`, November 2014.

[HEFT08] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. Analysis of hybrid systems using HySAT. In *Proc. of ICONS*, pages 196–201. IEEE CS, 2008.

[Her11] Christian Herde. *Efficient solving of large arithmetic constraint systems with complex Boolean structure: proof engines for the analysis of hybrid discrete-continuous systems*. PhD thesis, Carl-von-Ossietzky-Universität Oldenburg, 2011.

[KCC13] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[KWB$^+$14] Sreedhar Saseendran Kumar, Jan Wülfing, Joschka Boedecker, Ralf Wimmer, Martin Riedmiller, Bernd Becker, and Ulrich Egert. Autonomous control of network activity. In *Proc. of the 9th Int'l Meeting on Substrate-Integrated Microelectrode Arrays (MEA)*, July 2014.

[MRT12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012.

[PMG$^+$14] Eric A. Pohlmeyer, Babak Mahmoudi, Shijia Geng, Noeline W. Prins, and Justin C. Sanchez. Using reinforcement learning to provide stable brain-machine interface control despite neural input reorganization. *PLoS ONE*, 9(1), January 2014.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, October 1986.

[SB98] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[SB14] Karsten Scheibler and Bernd Becker. Implication graph compression inside the SMT solver iSAT3. In *Proc. of MBMV*, pages 25–36. Cuvillier, 2014.

[SKB13] Karsten Scheibler, Stefan Kupferschmid, and Bernd Becker. Recent improvements in the SMT solver iSAT. In *Proc. of MBMV*, pages 231–241. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2013.

[SS96] João P. Marques Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In *Proc. of ICCAD*, pages 220–227, 1996.

[Tse68] Grigori S. Tseitin. On the complexity of derivations in propositional calculus. In A. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logics*. 1968.