Solving Constraint Systems from Traffic Scenarios for the Validation of Autonomous Driving

Karsten Scheibler Andreas Eggers Tino Teige Marius Walz Tom Bienmüller
Udo Brockmeyer

BTC Embedded Systems AG, Gerhard-Stalling-Straße 19, 26135 Oldenburg, Germany
<firstname.lastname>@btc-es.de
http://www.btc-es.de/

Abstract

The degree of automation in our daily life will grow rapidly. This leads to big challenges regarding the *safety validation* of autonomous robots which take over more and more tasks being – as of yet – predestinated for humans. This is in particular true for the emerging area of *autonomous driving* which aims at making road traffic safer, more efficient, more economic, and more comfortable. One promising approach for the safety validation of autonomous driving is the *virtual* simulation of traffic scenarios, i.e. conducting the majority of tests in virtual reality instead of the real world. In addition to quantity, the *quality* of such tests with a focus on critical traffic scenarios will be an essential ingredient for safety validation.

In this paper, we investigate the concretization of traffic scenarios – in particular, scenarios which are specified as a set of constraints with interval parameters. We rely on the iSAT algorithm to perform the core reasoning, adapt its decision heuristics and complement it with an ICP-aware formula generation for non-linear formulas. The resulting Scenario Concretization Solver (SCS) – although being written in Java – is able to outperform SMT solvers on this problem class.

Keywords: Autonomous Driving, Traffic Scenarios, Constraint Systems, Constraint Solving

1 Introduction

Virtual safety validation is bound to play an increasingly important role in the development of complex systems. This is especially true in the context of autonomous driving – where achieving a convincing degree of coverage purely by testing an autonomous vehicle in the real world may not only be impractical (cf. e.g. [12] for an estimate of required mileages), especially given that each change to the system may trigger a full re-test, but furthermore be outright dangerous to the general public.

The need for virtual validation has thus been widely recognized in both industry and academia [13, 8] and has consequently even found its way into the recently published standard for "road vehicles – safety of the intended functionality" [1].

In: J. Abbott, A. Griggio (eds.): Proceedings of the 4th SC-square Workshop, Bern, Switzerland, 10th July 2019, published at http://ceur-ws.org

An essential element in the virtual validation of autonomous vehicles is subjecting the so-called "ego vehicle" (i.e. the vehicle to be tested) to various traffic scenarios that describe real-world situations. Such traffic scenarios can be written manually by experts (e.g. requirement engineers, traffic experts, even by function developers who are keen on getting quick feedback), can be derived from past experience of critical situations (e.g. accident data bases), or by observing real-world traffic. In either case, a scenario may have to be designed or to be analyzed by human operators, who need to have a quick grasp of the contents of the scenario. At the same time, scenarios need to have a clear semantics to allow unambiguous communication, simulation, and testing.

This important topic of scenario formalization is still the subject of ongoing efforts to develop and standardize scenario formats in academia, e.g. traffic sequence charts [5], as well as industry – especially the ASAM OpenScenario standardization in which a wide range of OEMs, suppliers, and tool vendors are involved[1]. While a standard like OpenSceneario may in the future very well become a household name within the context of any virtual validation effort, for the purposes of this paper we confine ourselves to a core of essential elements that we believe define a sensible description of vehicle behavior and inter-vehicular relationships based on the observations we previously made in [7]. The precise description of this formalism on which we base our work can be found in Section 2.

In our view, scenarios need to accommodate different levels of abstraction to allow for an efficient use of modeling effort. By specifying a scenario on an abstract level and using partially or fully automated variation strategies, one abstract scenario can lead to a huge number of concrete test cases. One essential element by which this degree of abstractness can be controlled is the use of intervals for the description of e.g. velocities, distances, and speed differences. Using relatively large intervals within a scenario leads to a wide range of possible executions and may thus lower the effort needed to write more specific concrete instances of the scenario manually.

Within this paper, we present a solver for the (discrete-time) concretization of abstract scenarios. There are in fact multiple purposes of such a concretization. Firstly, scenario concretization allows the modeler of a scenario to check the plausibility and validity of the scenario by seeing one or even multiple concrete instances of it. Unintended behavior can thus be detected as well as inconsistencies be analyzed and fixed. Secondly, a concretization may help to actually execute the scenario during simulation. While a concretization in the form of trajectories for all involved vehicles cannot be used directly as a test case (since the ego vehicle as system under test must of course be allowed to behave freely during the test), the concretization may be used to find a dynamic or rather reactive test strategy that adapts the surrounding vehicles to the ego vehicle's behavior. Thirdly, scenario concretizations can be influenced by adding side constraints that describe different regions of the solution space. Scenario concretization may thus be used as a form of guided scenario variation – a part of the possible automation that reduces the manual effort needed to specify scenarios.

2 Traffic Scenarios

As motivated above, scenario formalisms that could be accepted industry-wide are still to be developed. For our purposes, however, we assume an essential core of scenario elements for which we design our solver. In this section, we introduce this scenario formalism which mostly follows and refines our previous presentation [7].

2.1 Syntax of Traffic Scenarios

A traffic scenario is defined by the following elements. Let $\mathbb{IR}$ denote the set of closed intervals over the real numbers $\mathbb{R}$ including the (half-)open intervals containing $-\infty$ and $+\infty$. We further denote by ID a unique identifier. For the sake of simplicity, we restrict the scenarios to the two dimensional plane and a straight road (both limitations could be lifted at a later stage). We call x the longitudinal direction parallel to the direction of the road and y the lateral direction orthogonal to the road.

Vehicle Type: A *vehicle type* consists of ranges for speed $[v_{min}, v_{max}] \in \mathbb{IR}$ and acceleration $[a_{min}, a_{max}] \in \mathbb{IR}$.

Vehicle: A *vehicle* is defined by an ID and its vehicle type.

Lane: A *lane* is defined by an ID, a width $w \in \mathbb{R}$, and a left and right neighboring lane, which may or may not exist.

[1] https://www.asam.net/standards/detail/openscenario/

Lane Constraint: A *lane constraint*, denoted by $\mathit{lane}(h, s, t, r)$, is defined by a vehicle h, a source lane s, a target lane t, and a change rate $r \in \mathbb{IR}$.

Distance Constraint: A *distance constraint* is defined by two vehicles h_1 and h_2, an initial distance $i \in \mathbb{IR}$, an invariant distance $d \in \mathbb{IR}$, a final distance $f \in \mathbb{IR}$, and a distance change rate $r \in \mathbb{IR}$. It is denoted by $\mathit{distance}(h_1, h_2, i, d, f, r)$. All distances are given only with respect to the x dimension, such that e.g. two cars driving alongside on different lanes, would have a distance of 0. It should be noted that the invariant distance d is only partially handled by the current SCS version (cf. Section 3.3).

Speed Constraint: A *speed constraint*, denoted by $\mathit{speed}(h, i, s, f, r)$, is defined by a vehicle h, an initial speed $i \in \mathbb{IR}$, an invariant speed $s \in \mathbb{IR}$, a final speed $f \in \mathbb{IR}$, and a speed change rate $r \in \mathbb{IR}$. This speed is given only with respect to the x dimension, i.e. the lateral component during lane changes is ignored.

Speed Difference Constraint: A *speed difference constraint* is defined by two vehicles h_1 and h_2, an initial speed difference $i \in \mathbb{IR}$, an invariant speed difference $s \in \mathbb{IR}$, a final speed difference $f \in \mathbb{IR}$, and a speed difference change rate $r \in \mathbb{IR}$. It is denoted by $\mathit{speed\_diff}(h_1, h_2, i, s, f, r)$. Also in this case, only the velocities in the x dimension are considered.

Traffic Phase: A *traffic phase* $\mathcal{P} = (d, L, H, e, C)$ is given by an admissible duration $d \in \mathbb{IR}$, a set L of lanes, a set H of vehicles with one prominently labeled as "ego vehicle" $e \in H$, and a set C of lane, distance, speed, and speed difference constraints defining the initial, invariant, and final states admissible within that phase.

Traffic Scenario: A *traffic scenario* $\mathcal{S} = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$ is given by a sequence of traffic phases $\mathcal{P}_1$, ..., $\mathcal{P}_n$ with the same ego vehicle e in each phase, i.e. $e = e_i$ for each ego vehicle e_i of $\mathcal{P}_i$.

2.2 Semantics of Traffic Scenarios

The semantics of a traffic scenario $\mathcal{S}$ is given by runs. Intuitively, a run comprises the behavior of all vehicles of the traffic scenario that match all the constraints of the traffic phases. The model of the physical behavior of a vehicle for a step of (arbitrary) duration $\Delta t \in \mathbb{R}$ is given by the following equations, with x, y, v_x, v_y, a_x, a_y being the x and y position as well as the speed and acceleration in x and y directions, respectively. The primed version w' of a variable w denotes the value of w after the step.

$$v'_x = v_x + \Delta t \cdot a_x \tag{1}$$
$$v'_y = v_y + \Delta t \cdot a_y \tag{2}$$
$$x' = x + \Delta t \cdot 1/2 \cdot (v'_x + v_x) \tag{3}$$
$$y' = y + \Delta t \cdot 1/2 \cdot (v'_y + v_y) \tag{4}$$

For a vehicle h let $s(h)$ be some valuation for all state components x, y, v_x, v_y, a_x, a_y of v. For a vehicle h and a lane, distance, speed, or speed difference constraint c, we define $\mathit{init}(s(h), c)$, $\mathit{inv}(s(h), c)$, and $\mathit{final}(s(h), c)$ to be true if and only if the initial, invariant, and final constraints of c, respectively, are satisfied by valuation $s(h)$.

Run of a Traffic Scenario: Let $\mathcal{S} = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$ be a traffic scenario with $\mathcal{P}_i = (d_i, L_i, H_i, e_i, C_i)$ being a traffic phase. A *run*

$$r = ((t_1^1, s_1^1), \ldots, (t_{m_1}^1, s_{m_1}^1), \ldots, (t_1^n, s_1^n), \ldots, (t_{m_n}^n, s_{m_n}^n))$$

of $\mathcal{S}$ satisfies the following conditions with $t_j^i \in \mathbb{R}$ being increasing time points and $s_j^i(h)$ for all $h \in H_i$ being a valuation for all state components of h.

- All vehicles follow the model of the physical behavior described by equations 1, 2, 3, and 4, i.e. for $r = (\ldots, (t, s), (t', s'), \ldots)$, we have $\Delta t = t' - t$ and s' and s are connected by equations 1, 2, 3, and 4.
- The admissible duration of each phase is met:
  $\forall 1 \leq i \leq n : t_{m_i}^i - t_1^i \in d_i$.
- The initial constraints of the first phase $\mathcal{P}_1$ are satisfied:
  $\forall h \in H_1 \ \forall c \in C_1 : \mathit{init}(s_1^1(h), c)$.
- The initial constraints of each phase $\mathcal{P}_{i>1}$ are satisfied:
  $\forall 1 < i \leq n \ \forall h \in H_i \ \forall c \in C_i : \mathit{init}(s_{m_{i-1}}^{i-1}(h), c)$.

- The invariant constraints of each phase are satisfied:
  $\forall 1 \leq i \leq n\ \forall 1 \leq j \leq m_i\ \forall h \in H_i\ \forall c \in C_i : \mathit{inv}(s^i_j(h), c)$.
- The final constraints of each phase satisfied:
  $\forall 1 \leq i \leq n\ \forall h \in H_i\ \forall c \in C_i : \mathit{final}(s^i_{m_i}(h), c)$.

2.3 Example

In our formalism, a simple overtaking scenario with two vehicles h_1 and h_2 could be modeled in three phases as follows. While h_2 stays on the right lane in all phases, h_1 changes from the right to the left lane in the first phase, stays on the left lane in the second phase and changes back to the right lane in the third phase. Additionally, the distance between both vehicles changes. While being behind h_2 in the first phase, h_1 overtakes h_2 in the second phase and stays in front of h_2 in the third phase. In all phases h_1 has at least the same speed as h_2, i.e. the speed difference between both vehicles is less than or equal to zero. Or more formally:

- Let T be a vehicle type with $v \in [-5.5, 69]$ and $a \in [-10, 5.5]$.
- Let h_1 and h_2 be two vehicles of type T and let $H = \{h_1, h_2\}$.
- Let *left* and *right* be two neighboring lanes and let $L = \{\mathit{left}, \mathit{right}\}$.
- Let $c_1, c_2, \ldots, c_{12}$ be constraints and let $\sim$ denote the interval $(-\infty, +\infty)$:

$$
\begin{array}{rcl}
c_1 & = & \mathit{distance}(h_1, h_2, \sim, [4, 10], [2, 5], \sim) \\
c_2 & = & \mathit{speed\_diff}(h_1, h_2, \sim, (-\infty, 0], \sim, \sim) \\
c_3 & = & \mathit{lane}(h_1, \mathit{right}, \mathit{left}, [0, +\infty)) \\
c_4 & = & \mathit{lane}(h_2, \mathit{right}, \mathit{right}, [0, 0]) \\
c_5 & = & \mathit{distance}(h_1, h_2, [2, 5], \sim, [-5, -2], \sim) \\
c_6 & = & \mathit{speed\_diff}(h_1, h_2, \sim, (-\infty, 0], \sim, \sim) \\
c_7 & = & \mathit{lane}(h_1, \mathit{left}, \mathit{left}, [0, 0)) \\
c_8 & = & \mathit{lane}(h_2, \mathit{right}, \mathit{right}, [0, 0]) \\
c_9 & = & \mathit{distance}(h_1, h_2, [-5, -2], (-\infty, -2], \sim, \sim) \\
c_{10} & = & \mathit{speed\_diff}(h_1, h_2, \sim, (-\infty, 0], \sim, \sim) \\
c_{11} & = & \mathit{lane}(h_1, \mathit{left}, \mathit{right}, (-\infty, 0]) \\
c_{12} & = & \mathit{lane}(h_2, \mathit{right}, \mathit{right}, [0, 0])
\end{array}
$$

- Let $C_1 = \{c_1, \ldots, c_4\}$, $C_2 = \{c_5, \ldots, c_8\}$ and $C_3 = \{c_9, \ldots, c_{12}\}$.
- Let $\mathcal{P}_i([1, 5], L, H, h_1, C_i)$ with $i \in \{1, 2, 3\}$ be three traffic phases.
- Then $\mathcal{S}_A = (\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3)$ is a simple overtaking scenario.

In Section 4 this scenario is parameterized by the number of vehicles and phases. Similar to h_2, every additional vehicle stays on the right lane in all phases. Furthermore, distance constraints are added to ensure that h_{i+1} drives in front of h_i for $i \geq 2$. Another variant of $\mathcal{S}_A$ (denoted $\mathcal{S}_B$) is obtained by introducing additional speed difference constraints. If the number of phases is increased, extra overtaking maneuvers are performed until there is no further vehicle to overtake – in this case h_1 stays in front of the last vehicle h_n for the remaining phases.

3 Scenario Concretization Solver (SCS)

As explained in the previous section, our scenario formalism uses intervals to constrain the behavior of the vehicles. For example, a vehicle h might be required to have a speed between 20 m/s and 25 m/s at the beginning of a phase – i.e. $c_1 = \mathit{speed}(h, i, s, f, r)$ with $i \in [20, 25]$. Thus, it is only consequent to check whether the intervals within a constraint are consistent. For example, if c_1 additionally constrains the invariant speed s to $[27, 30]$, then this contradicts with the initial speed i – as the interval of i does not intersect with the interval of s. On the other hand, in case i and s have an intersection (e.g. with $i \in [20, 25]$ and $s \in [23, 30]$), then it is possible to confine i to $[23, 25]$.

Unfortunately, such a simple interval intersection check only covers some of the dependencies within a constraint. Furthermore, such a check is unable to handle dependencies between different constraints. For example,

let vehicle h_1 have an invariant speed interval of $[20, 25]$ while vehicle h_2 has an invariant speed interval of $[27, 30]$. Additionally, let h_1 and h_2 be at different lanes but at the same longitudinal position at the beginning of a phase $\mathcal{P}$. Obviously, the distance between h_1 and h_2 is zero at the beginning of $\mathcal{P}$ but will increase over time – as h_2 is strictly faster than h_1. Hence, there could be a conflict with a distance constraint depending on the allowed duration of $\mathcal{P}$. Similar to the equations for the physical behavior of a vehicle (cf. Section 2.2), the dependencies between the speeds of h_1 and h_2, their distance and the phase duration can be expressed symbolically in a formula (cf. Section 3.3). As the involved variables have interval valuations, *Interval Constraint Propagation* (ICP) [3] is a natural choice to check whether such a set of equations is inconsistent regarding the given intervals.

Our approach builds on ICP – but embeds it into the *Conflict-Driven Clause Learning* (CDCL) [18] framework to obtain a search process which allows backtracking to be performed in a non-chronological manner. This combination of ICP and CDCL is also known as the *iSAT algorithm* [10, 11]. Hence, this paper describes another application for it. Originally, the iSAT algorithm was developed for hybrid systems verification. For such systems one could expect some kind of *robustness* [9, 15], i.e. small perturbations do not change the behavior of the system – otherwise the system would have problems with noisy input signals anyway. A similar kind of robustness is characteristic to the traffic scenarios considered in this paper. This means, we accept that the given constraints might be violated by a small margin. For example, if a constraint requires a vehicle to have a speed of exactly 25 m/s, then 25.001 m/s or 24.999 m/s are also considered as valid solutions. In fact, this violation margin is orders of magnitude smaller and was below 10^{-12} in our experiments. Additionally, we apply reasonable bounds to accelerations, speeds, durations, positions and distances, i.e. every $-\infty$ or $+\infty$ in an interval will be replaced with a number which is considered to be a valid lower or upper bound regarding a physically reasonable behavior.

In order to keep the paper self-contained we start with a short and informal introduction to ICP, before giving an overview of the iSAT algorithm. The final two subsections contain the main contributions of this paper: the ICP-aware conversion from a given constraint set to a formula as well as the heuristics which selects a variable for the next interval split to be performed.

3.1 ICP in a Nutshell

When considering an equation like $x = y + z$, the variables x, y and z are usually understood as placeholders which stand for single values. This is generalized when performing ICP [3] – i.e. instead of single values, each variable is considered to represent a set of values bounded by an interval. The interval calculation is performed regarding the worst-case by asking the question: what is the smallest and largest value when considering all value-pairs within the intervals of y and z while calculating the sum of each value-pair. This result (denoted by x') can be calculated by adding the lower and upper bounds (denoted by lb and ub) of y and z. The new interval of x is obtained by intersecting the intervals of x and x'. In case of the subtraction operation $x = y - z$, the smallest value is obtained by subtracting the largest possible value of z from the smallest possible value of y. In a similar fashion the largest possible value is calculated:

$$\begin{array}{rcl} x'_{lb} &=& y_{lb} + z_{lb} \\ x'_{ub} &=& y_{ub} + z_{ub} \end{array} \qquad \left| \qquad \begin{array}{rcl} x'_{lb} &=& y_{lb} - z_{ub} \\ x'_{ub} &=& y_{ub} - z_{lb} \end{array} \right.$$

The multiplication operation $x = y \cdot z$ requires a case distinction regarding the sign of its factors, e.g. $y_{ub} \cdot z_{ub}$ might yield x'_{lb} or x'_{ub} depending on whether the intervals of y and z contain positive or negative values. Furthermore, as the calculations are performed with floating-point numbers, outward rounding is applied.

If an equation contains more than one arithmetic operation, it is possible to build a dedicated interval propagator for it. Alternatively and similar to the Tseitin-transformation [19], each equation with n arithmetic operations can be decomposed into a conjunction of $n - 1$ equations containing one arithmetic operation by introducing additional auxilary variables [11] – such a decomposition is also used in our approach. Thus, interval addition, subtraction and multiplication are sufficient for the formulas considered in this paper. Additionally, for each equation $x = y \odot z$ with $\odot \in \{-, +, \cdot\}$ its two redirected forms are examined as well – e.g. in case of $x = y + z$, the redirected forms $y = x - z$ and $z = x - y$ are also considered by ICP. Thus, besides $-$, $+$ and $\cdot$ a limited form of interval division is used as well (denoted by $/$) in order to confine the intervals of y and z regarding the equation $x = y \cdot z$.

As described, each equation and its redirected forms are handled individually by ICP. An equation $x = y \odot z$ with $\odot \in \{-, +, \cdot, /\}$ is consistent regarding the interval valuation of its three variables, if the intervals of x and $y \odot z$ have a non-empty intersection. If an equation is consistent, then it is guaranteed to have a solution within

the intervals of the variables contained in the equation. But there is no such guarantee if ICP is applied to a conjunction of multiple equations, i.e. the consistency of all equations does not imply the existence of a solution. For example, in the case of $x = y + z$ with $x \in [0, 0]$ and $y, z \in [-10, 10]$ there is a dependency between y and z. On the one hand, for each $y \in [-10, 10]$ there is a $z \in [-10, 10]$ such that $y + z \in [0, 0]$. Thus, ICP cannot confine the intervals of y and z. On the other hand, not every pair of values in the interval of y and z is a solution for $x = y + z$ when $x \in [0, 0]$. Hence, as the intervals of y and z stay unchanged, the dependency between both variables induced by the interval of x is not visible for ICP when examining $w = y - z$ with $w \in [0, 0]$. The conjunction of both equations implies $y = z = 0$, but ICP is unable to detect this[2]. In a similar setting a set of further equations might imply other values for y and z, but as ICP keeps $y, z \in [-10, 10]$ this inconsistency will stay undetected. Thus, when multiple equations are involved, the intervals determined by ICP overapproximate the set of solutions. In other words, if ICP detects an inconsistency there is indeed no solution – in case all equations are consistent, the current interval valuation of all variables might or might not contain a solution. Therefore, a further mechanism is needed to obtain a method which is able to find a solution or prove the absence thereof.

The basic idea to get such a method is to reduce the width of the intervals. If, in the extreme case, all variables have point intervals (i.e. an interval width of zero and thus single values), then the consistency of all equations proves that the current valuation is a valid solution. Hence, intuitively, if all equations are consistent and the interval widths of all variables are very small, one could call this an almost-solution. This directly relates to the robustness mentioned earlier. If small perturbations are allowed (i.e. by adding a small individual perturbation constant to each equation), then this almost-solution can be used to obtain a valid solution for such a perturbed conjunction of equations – e.g. by (a) taking the midpoint of each interval, and (b) calculating the needed perturbation constant for each equation. Hence, the question remains how to reduce the width of the intervals.

This can be achieved by performing heuristic decisions in the form of interval splits. For example, the midpoint of an interval could be chosen as the new upper or lower bound. This is repeated until all intervals have a width below a given *minimum splitting width*. After performing such an interval split, ICP is applied again to deduce the consequences of the split. Of course, such a heuristic decision might lead to an inconsistency. In such a case, the last decision has to be reverted in order to try the interval part which was excluded by the split – resulting in a search process with backtracking. Instead of just reverting the last decision, it is also possible to perform a more thorough analysis to isolate the cause of the current inconsistency – which is not necessarily the last decision. Such an analysis is part of the iSAT algorithm. Its details are explained in the next subsection.

3.2 The iSAT Algorithm

In order to perform a more thorough analysis in case of an inconsistency, an implication graph is used to keep track of the deductions performed by ICP. Besides the deduced lower or upper bounds, it also contains the reasons for each deduction – i.e. the relevant bounds of the other variables in the equation examined by ICP. For example, if ICP examines the equation $x = y + z$ together with the current interval valuation $x \in [0, 10]$, $y \in [2, 5]$ and $z \in [1, 3]$, then the newly deduced lower bound of x is recorded as $((y \geq 2) \wedge (z \geq 1)) \Rightarrow (x \geq 3)$ in the implication graph. Thus, $(y \geq 2)$ and $(z \geq 1)$ can be understood as the reasons for this deduction.

The implication graph is partitioned into decision levels – one level for each decision (i.e. a heuristic interval split). If ICP detects an inconsistency, then an interval of a variable x became empty, i.e. $x_{lb} > x_{ub}$. If this inconsistency happens on decision level 0 (i.e. without performing a decision), the formula is proved to have no solution – i.e. it is unsatisfiable and no further analysis is required. Otherwise, x_{lb} and x_{ub} are analyzed – as at least one of both bounds was deduced by ICP and thus has reasons. Basically, those reasons which were themselves deduced on the current decision level are analyzed in order to obtain one reason which is the *first unique implication point* (1UIP) [20]. The result is a set of reasons R being assigned (i.e. deduced or decided) on earlier decision levels together with the 1UIP which was assigned on the current decision level. Hence, the result is just a new implication: the set of reasons R assigned on earlier decision levels implies the 1UIP. This deduction can be applied on the highest decision level among the decision levels in R – allowing non-chronological backtracking. Usually, the set of reasons in R is small compared to the number of variables in the overall problem, i.e. R is a rather local explanation for the observed inconsistency. Therefore, it is advantageous to learn such implications. If a similar situation occurs again during the search process, such a learned implication can be reused in order to prevent an already known inconsistency to be visited again.

[2] In this concrete case the effect can be mitigated by adding: $(y_2 = 2 \cdot y) \wedge (z_2 = 2 \cdot z) \wedge (w = x - y_2) \wedge (w = z_2 - x)$

Here, we motivated the iSAT algorithm as ICP-based search process with non-chronological backtracking and learning. In the context of satisfiability checking, the motivation is somewhat different. *Satisfiability Modulo Theories* (SMT) considers Boolean combinations of theory atoms – e.g. Boolean combinations of equations and inequalities containing non-linear real arithmetic. When performing lazy SMT solving, the Boolean skeleton is processed by a (usually CDCL-based) SAT solver which determines a valuation for the truth values of the theory atoms. A separate theory solver is used to check whether there is a solution in the conjunction of theory atoms under the given truth values. If this is not the case, a (hopefully) small subset of the theory atoms is returned as conflict explanation to the SAT solver. In the iSAT algorithm there is no separation between the SAT solver and the theory part – instead ICP is tightly integrated into CDCL. In fact, each interval bound can be considered as a literal – depending on its polarity it represents a lower or an upper bound. Thus, the iSAT algorithm performs CDCL [18] and the 1UIP based conflict analysis [20] on top of interval bounds. Due to ICP, the iSAT algorithm can be adapted to handle a broad range of arithmetic operations. Besides non-linear real arithmetic and transcendental functions, it is also possible to perform accurate reasoning for floating-point arithmetic [17].

Our approach uses a solver core being similar to iSAT3 [16] – which is the third implementation of the iSAT algorithm. Currently, our scenario formalism results in a conjunction of equations. As the iSAT algorithm considers Boolean combinations of theory atoms, our approach can be easily adapted if the scenario formalism is extended with Boolean structure – e.g. by having multiple versions of one phase and requiring that exactly one of these versions has to be selected.

3.3 Converting a Scenario to a Formula

In the following we give an overview of the basic building blocks the complete formula is composed of. The current version of SCS divides each traffic phase into two sub-phases with equal duration[3] – i.e. if d_i is the duration of $\mathcal{P}_i$, then each sub-phase has duration $d'_i = 1/2 \cdot d_i$. But in order to simplify the presentation of the formulas, we neglect this division. In fact, the formulas presented below are also used within a sub-phase – but depending on whether the sub-phase is the first or last, the intervals specified by the constraints are applied to different variables, i.e. initial conditions are applied to the first sub-phase, while final conditions are applied to the last sub-phase.

As the duration d_i of a phase $\mathcal{P}_i$ might be an interval, it is a variable within the equations. Thus, the resulting set of equations is non-linear. For each vehicle h and each phase the following set of equations is generated for the x and y dimension (indices for h, x, y and the phase are omitted in favor of better readability):

$$\begin{aligned}
\mathit{sum_of_speed} &= \mathit{initial_speed} + \mathit{final_speed} \\
\mathit{average_speed} &= 1/2 \cdot \mathit{sum_of_speed} \\
\mathit{change_of_speed} &= \mathit{final_speed} - \mathit{initial_speed} \\
\mathit{change_of_speed} &= \mathit{rate_of_speed} \cdot d_i \\
\mathit{change_of_position} &= \mathit{final_position} - \mathit{initial_position} \\
\mathit{change_of_position} &= \mathit{average_speed} \cdot d_i
\end{aligned}$$

In case of the y dimension *initial_position* and *final_position* are point intervals representing the center of the source lane s and target lane t specified in the $\mathit{lane}(h, s, t, r)$ constraint. The change rate r from this constraint corresponds to the invariant speed in the y dimension. As shown in Section 2.2, the speed evolves linearly from its initial to its final value within each time step. Thus, it suffices to use the interval of the invariant speed as a further restriction for the intervals of *initial_speed* and *final_speed*. For the x dimension, the position stays unconstrained while *initial_speed*, *final_speed* and *rate_of_speed* are constrained by i, f and r from the $\mathit{speed}(h, i, s, f, r)$ constraint. Here again, the invariant speed s is an additional restriction for *initial_speed* and *final_speed*.

In case a *distance*() or *speed_diff*() constraint exists for a vehicle pair (h_1, h_2) in at least one phase, then the following set of equations will be added to *each* phase (the indices 1 and 2 correspond to h_1 and h_2 respectively):

$$\begin{aligned}
\mathit{initial_speed_diff} &= \mathit{initial_speed}_2 - \mathit{initial_speed}_1 \\
\mathit{final_speed_diff} &= \mathit{final_speed}_2 - \mathit{final_speed}_1 \\
\mathit{change_of_speed_diff} &= \mathit{final_speed_diff} - \mathit{initial_speed_diff} \\
\mathit{change_of_speed_diff} &= \mathit{change_of_speed}_2 - \mathit{change_of_speed}_1 \\
\mathit{change_of_speed_diff} &= \mathit{rate_of_speed_diff} \cdot d_i
\end{aligned}$$

[3] In the future we plan to determine the number of the sub-phases dynamically – e.g. depending on the phase duration

$$
\begin{array}{lcl}
\mathit{sum\_of\_speed\_diff} & = & \mathit{initial\_speed\_diff} + \mathit{final\_speed\_diff} \\
\mathit{average\_speed\_diff} & = & 1/2 \cdot \mathit{sum\_of\_speed\_diff} \\
\mathit{average\_speed\_diff} & = & \mathit{average\_speed}_2 - \mathit{average\_speed}_1 \\
\mathit{initial\_distance} & = & \mathit{initial\_position}_2 - \mathit{initial\_position}_1 \\
\mathit{final\_distance} & = & \mathit{final\_position}_2 - \mathit{final\_position}_1 \\
\mathit{change\_of\_distance} & = & \mathit{final\_distance} - \mathit{initial\_distance} \\
\mathit{change\_of\_distance} & = & \mathit{change\_of\_position}_2 - \mathit{change\_of\_position}_1 \\
\mathit{change\_of\_distance} & = & \mathit{average\_speed\_diff} \cdot d_i
\end{array}
$$

It should be noted that these equations are only added for the x dimension – as the speed difference and distance are not considered in the y dimension. Each $\mathit{speed\_diff}(h_1, h_2, i, s, f, r)$ constraint restricts *initial_speed_diff*, *final_speed_diff* and *rate_of_speed_diff* with i, f and r. The invariant speed difference s is an additional restriction for *initial_speed_diff* and *final_speed_diff* – similar to the *speed*() constraint. Furthermore, each $\mathit{distance}(h_1, h_2, i, d, f, r)$ constraint restricts *initial_distance* and *final_distance* with i and f as well as d. The change rate r restricts the invariant speed difference and thus *initial_speed_diff* and *final_speed_diff*. In contrast to the speed, the distance does not necessarily evolve linearly within a phase. In particular, it could happen that the distance reaches its peak somewhere within a phase – depending on how the speed difference evolves. Hence, the invariant distance is only partially handled at the moment.

The formula creation is ICP-aware, i.e. it includes some redundant equations, which support the deductions performed by ICP. For example, it is advantageous to generate the above shown set of equations for a vehicle pair (h_1, h_2) in *each* phase – as this allows ICP to propagate the distances and speed differences to the surrounding phases which might be subject to further constraints for this vehicle pair. Additionally, cyclic dependencies of vehicle pair constraints result in further equations to be added. For example, if the vehicle pairs (h_1, h_2), (h_2, h_3) and (h_1, h_3) are subject of distance or speed difference constraints, then the following equations are generated:

$$
\begin{array}{lcl}
\mathit{initial\_distance}_{1,3} & = & \mathit{initial\_distance}_{1,2} + \mathit{initial\_distance}_{2,3} \\
\mathit{final\_distance}_{1,3} & = & \mathit{final\_distance}_{1,2} + \mathit{final\_distance}_{2,3} \\
\mathit{initial\_speed\_diff}_{1,3} & = & \mathit{initial\_speed\_diff}_{1,2} + \mathit{initial\_speed\_diff}_{2,3} \\
\mathit{final\_speed\_diff}_{1,3} & = & \mathit{final\_speed\_diff}_{1,2} + \mathit{final\_speed\_diff}_{2,3}
\end{array}
$$

3.4 Decision Heuristics to Guide the Interval Splits

Due to the formula structure and the deductions performed by ICP, reductions in the interval width of the variables representing speeds or durations imply that the intervals of other variables will shrink as well, e.g. *change_of_speed*, *average_speed*, *change_of_position*, *change_of_speed_diff*, *average_speed_diff* as well as *change_of_distance*. Thus, intuitively, it seems to be advantageous to split speed and duration variables first – this also conforms with our empirical observation. Therefore, we use three buckets for variables representing (B1) speeds or durations, (B2) distances, and (B3) positions. Variables in (B2) are only considered if (B1) is empty – similarly, all variables in (B2) are split before any variable from (B3) is processed. The variables in each bucket are selected pseudo-randomly. Furthermore, each interval split of a variable alternates between cutting off the lower or upper interval part – the size of this part varies pseudo-randomly between 1/3 and 1/8. Additionally, we use different minimum splitting widths in each bucket: (B1) 2^{-38}, (B2) 2^{-37}, and (B3) 2^{-36}.

4 Experimental Results

For technical reasons SCS is embedded into a Java-based framework and is itself written in Java[4]. As mentioned, the core of SCS is similar to the core of iSAT3 [16]. Hence, iSAT3 is a natural candidate for a comparison. We use it with two different settings regarding the minimum splitting width[5]: iSAT3a uses 2^{-36} (similar to SCS) while iSAT3b applies its default value of 0.1. Furthermore, we compare SCS with MathSAT [4], Yices [6], Z3 [14] and CVC4 [2] – as these solvers are also able to solve formulas with non-linear arithmetic. On the other hand, these four solvers search for a solution which satisfies all constraints without a violation (no matter how small it is), while SCS as well as iSAT3 exploit the knowledge that constraints might be violated by a small margin [6].

[4] As Java does not allow to change the floating-point rounding mode, outward rounding is emulated with `Math.nextafter()`.

[5] iSAT3a: `--msw 0x1.0p-36 --mpr 0x1.0p-40`, iSAT3b: `--msw 0.1 --mpr 0.01`

[6] This margin correlates with the minimum splitting width – but is not equivalent to it, as the formula structure has an influence as well.

	R / #V / #P	MathSAT	Yices	Z3	CVC4	iSAT3a	iSAT3b	SCS
$\mathcal{S}_A$	UNS / 4 / 1	0.06s	0.01s	0.04s	0.21s	0.01s	0.01s	1.14s
$\mathcal{S}_A$	UNS / 4 / 2	0.11s	0.02s	0.09s	0.41s	0.02s	0.02s	1.34s
$\mathcal{S}_A$	UNS / 4 / 5	0.31s	0.08s	0.83s	1.16s	0.08s	0.09s	1.69s
$\mathcal{S}_A$	UNS / 4 / 10	0.84s	0.15s	11.08s	2.87s	0.24s	0.28s	2.29s
$\mathcal{S}_A$	UNS / 4 / 20	1.87s	0.29s	778.54s	6.24s	0.63s	0.54s	2.75s
$\mathcal{S}_B$	UNS / 5 / 1	0.07s	0.02s	0.16s	0.27s	0.03s	0.02s	1.37s
$\mathcal{S}_B$	UNS / 5 / 2	0.16s	0.04s	0.22s	0.58s	0.07s	0.09s	1.59s
$\mathcal{S}_B$	UNS / 5 / 5	TO	10.94s	2.14s	TO	0.30s	0.34s	1.76s
$\mathcal{S}_B$	UNS / 5 / 10	1.23s	0.45s	3.37s	4.02s	0.81s	0.95s	2.47s
$\mathcal{S}_B$	UNS / 5 / 20	3.01s	0.58s	TO	8.66s	2.01s	1.95s	3.56s
$\mathcal{S}_A$	SAT / 2 / 1	0.17s	0.07s	0.02s	1.08s	1.11s	0.66s	1.37s
$\mathcal{S}_A$	SAT / 2 / 2	TO	0.27s	0.04s	TO	10.22s	0.97s	1.28s
$\mathcal{S}_A$	SAT / 2 / 5	TO	TO	0.34s	TO	14.06s	3.66s	1.50s
$\mathcal{S}_A$	SAT / 2 / 10	TO	TO	2.63s	TO	TO	20.25s	1.66s
$\mathcal{S}_A$	SAT / 2 / 20	TO	TO	493.81s	TO	TO	54.82s	2.59s
$\mathcal{S}_A$	SAT / 3 / 1	0.50s	0.07s	0.03s	TO	0.28s	0.31s	1.69s
$\mathcal{S}_A$	SAT / 3 / 2	TO	0.49s	0.10s	TO	0.43s	1.42s	1.31s
$\mathcal{S}_A$	SAT / 3 / 5	TO	34.10s	1.33s	TO	12.48s	7.29s	1.98s
$\mathcal{S}_A$	SAT / 3 / 10	TO	TO	TO	TO	TO	16.23s	4.74s
$\mathcal{S}_A$	SAT / 3 / 20	TO	TO	TO	TO	TO	128.76s	54.92s
$\mathcal{S}_A$	SAT / 4 / 1	0.90s	0.17s	0.05s	TO	94.96s	0.27s	1.26s
$\mathcal{S}_A$	SAT / 4 / 2	TO	TO	0.14s	TO	0.53s	1.42s	1.66s
$\mathcal{S}_A$	SAT / 4 / 5	TO	46.21s	0.97s	TO	MO	29.37s	1.89s
$\mathcal{S}_A$	SAT / 4 / 10	TO	TO	TO	TO	TO	127.25s	13.40s
$\mathcal{S}_A$	SAT / 4 / 20	TO	TO	TO	TO	MO	218.18s	86.52s
$\mathcal{S}_B$	SAT / 3 / 1	0.34s	0.04s	0.03s	0.13s	0.03s	0.02s	1.30s
$\mathcal{S}_B$	SAT / 3 / 2	1.64s	0.28s	0.06s	0.26s	0.14s	0.06s	1.27s
$\mathcal{S}_B$	SAT / 3 / 5	TO	4.48s	0.36s	TO	9.74s	3.07s	1.53s
$\mathcal{S}_B$	SAT / 3 / 10	TO	116.30s	4.51s	TO	13.47s	26.01s	2.80s
$\mathcal{S}_B$	SAT / 3 / 20	TO	TO	TO	TO	MO	270.86s	5.15s
$\mathcal{S}_B$	SAT / 4 / 1	0.51s	0.22s	0.07s	0.21s	0.06s	0.09s	1.17s
$\mathcal{S}_B$	SAT / 4 / 2	5.56s	1.74s	0.24s	0.44s	0.54s	0.28s	1.51s
$\mathcal{S}_B$	SAT / 4 / 5	TO	11.26s	0.56s	TO	52.42s	2.19s	1.69s
$\mathcal{S}_B$	SAT / 4 / 10	TO	TO	9.72s	TO	TO	60.40s	5.09s
$\mathcal{S}_B$	SAT / 4 / 20	TO	TO	TO	TO	TO	596.45s	21.20s
$\mathcal{S}_B$	SAT / 5 / 1	1.50s	0.11s	0.11s	0.31s	1.67s	0.25s	1.35s
$\mathcal{S}_B$	SAT / 5 / 2	17.27s	5.48s	0.19s	0.68s	2.55s	0.45s	1.67s
$\mathcal{S}_B$	SAT / 5 / 5	TO	31.10s	0.81s	TO	TO	2.63s	2.39s
$\mathcal{S}_B$	SAT / 5 / 10	TO	TO	13.52s	TO	MO	78.92s	18.32s
$\mathcal{S}_B$	SAT / 5 / 20	TO	TO	TO	TO	TO	TO	40.49s
$\sum$ UNS solved		9	**10**	9	9	**10**	**10**	**10**
$\sum$ SAT solved		9	17	23	7	17	29	**30**

Table 1: Comparing MathSAT 5.5.4, Yices 2.6.1, Z3 4.8.4, CVC4 1.6 and iSAT3 0.04-20170301 (with two different settings) with SCS on the formulas generated for the scenario families $\mathcal{S}_A$ and $\mathcal{S}_B$.

The comparison is performed on the basis of the formulas generated by SCS, i.e. the formulas are written out and converted to the HYS format (for iSAT3) and to the SMT2 format (for MathSAT, Yices, Z3 and CVC4)[7]. Thus, all solvers operate on the same formulas. In order to obtain small and large instances, the scenarios $\mathcal{S}_A$ and $\mathcal{S}_B$ (cf. Section 2.3) are parameterized regarding the number of vehicles and phases. The formula sizes range from 150 variables and 180 equations (containing arithmetic operations) up to 11 600 variables and 16 100 equations. We generated satisfiable and unsatisfiable instances – for an equal number of vehicles and phases the satisfiable and unsatisfiable instances only differ in the intervals constraining the variables.

The results are listed in Table 1. While the first column shows the scenario family, the second column lists the expected result – i.e. whether an instance is satisfiable (SAT) or unsatisfiable (UNS) – as well as the number of vehicles and phases. The remaining columns either list the runtime in seconds or contain TO (for time-out) or MO (for memory-out). While the measurements for MathSAT, Yices, Z3, CVC4 and iSAT3 were performed under Debian 9 on an Intel Xeon E5-2690 with 2.6 GHz and a time limit of 900 seconds as well as a memory limit of 15 GB, SCS was executed under Windows 7 on an Intel Xeon E5-2695 with 2.3 GHz and a memory limit of 2 GB. Besides being written in Java and executed on a slightly slower CPU, the runtime of SCS contains an additional overhead of several hundred milliseconds: the import of the scenario from the surrounding framework, the creation of a trajectory for each vehicle to be visualized graphically as well as the generated formula being written to a file.

The results show that the unsatisfiable instances are solved within seconds in most of the cases. While MathSAT, Z3 as well as CVC4 have one unsolved instance, Yices, iSAT3 and SCS are able to solve all unsatisfiable instances. In case of iSAT3 the result is not surprising, because it applies the same ICP-based reasoning as SCS. With ICP the unsatisfiability of the instances can be determined with a low number of non-chronological backtracking operations – the decision heuristics plays a minor role in these cases.

The picture changes when considering the satisfiable instances. Here, CVC4 and MathSAT each solve less than a third of the 30 instances. In particular, the instances of $\mathcal{S}_A$ are hard to solve for both solvers. Yices and Z3 show better performance, but still have time-outs for 13 and 7 instances respectively. The numbers of iSAT3a, iSAT3b and SCS can be seen as an indicator for the effectiveness of the decision heuristics used by SCS. As the the core of SCS is similar to the core of iSAT3, the numbers of iSAT3a give an impression how SCS would perform without its tailored decision heuristics. Although iSAT3b has only one time-out, the quality of its result (in terms of the margin a constraint might be violated) is worse compared to iSAT3a and SCS – in this set of instances, the violation margin of SCS is below 10^{-12}, while it is above 10^{-3} for iSAT3b. This comes due to the coarser minimum splitting width used by iSAT3b. Additionally, it can be observed that iSAT3a exceeded its memory limit of 15 GB in four cases, while SCS stays within its limit of 2 GB for all instances. Hence, the decision heuristics of SCS prevents that ICP deduces millions of new bounds even with a very small minimum splitting width. Overall, SCS is the only solver which was able to solve all instances. Furthermore, its decision heuristics allows SCS to outperform the other solvers in particular on larger satisfiable instances.

5 Conclusion

In the context of safety validation for autonomous vehicles, virtual validation is one promising approach to reduce the amount of testing in the real world. In this context, formally specified scenarios are used to describe the real-world traffic situations which have to be tested. In order to derive a test case from such a scenario, a concretization is needed. In this paper we investigated the concretization regarding a constraint-based scenario formalism containing interval parameters.

We identified this problem class as another application of the iSAT algorithm. Exploiting the domain specific knowledge, we adapted the solving strategy by an ICP-aware formula generation and a tailored decision heuristics. The resulting Scenario Concretization Solver (SCS) outperforms other solvers in particular on larger satisfiable instances.

References

[1] ISO/PAS 21448:2019: Road vehicles – Safety of the intended functionality (2019)

[2] Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanovi'c, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer (Jul 2011), snowbird, Utah

[7] http://www.btc-es.de/media/downloads/btc-es_sc2_traffic_scenarios.tar.gz

[3] Benhamou, F., Granvilliers, L.: Continuous and Interval Constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, pp. 571–603. Elsevier (2006)

[4] Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (eds.) Proceedings of TACAS. LNCS, vol. 7795. Springer (2013)

[5] Damm, W., Möhlmann, E., Peikenkamp, T., Rakow, A.: A Formal Semantics for Traffic Sequence Charts, LNCS, vol. 10760, pp. 182–205. Springer (2018)

[6] Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer-Aided Verification (CAV'2014). LNCS, vol. 8559, pp. 737–744. Springer (July 2014)

[7] Eggers, A., Stasch, M., Teige, T., Bienmüller, T., Brockmeyer, U.: Constraint Systems from Traffic Scenarios for the Validation of Autonomous Driving. In: Bigatti, A., Brain, M. (eds.) SC-square 2018, Oxford, UK, July 11, 2018

[8] Fraade-Blanar, L., Blumenthal, M.S., Anderson, J.M., Kalra, N.: Measuring Automated Vehicle Safety: Forging a Framework. Tech. rep., RAND Corporation (2018)

[9] Fränzle, M.: Analysis of Hybrid Systems: An Ounce of Realism Can Save an Infinity of States. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 126–140. Springer (1999)

[10] Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient Solving of Large Non-Linear Arithmetic Constraint Systems with Complex Boolean Structure. JSAT **1**(3-4), 209–236 (2007)

[11] Herde, C.: Efficient Solving of Large Arithmetic Constraint Systems with Complex Boolean Structure: Proof Engines for the Analysis of Hybrid Discrete-Continuous Systems. Ph.D. thesis, Carl von Ossietzky University of Oldenburg (2011)

[12] Kalra, N., Paddock, S.M.: Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability? Tech. rep., Rand Corporation (2016)

[13] Koopman, P., Wagner, M.: Toward a Framework for Highly Automated Vehicle Safety Validation. In: WCX World Congress Experience. SAE International (2018)

[14] de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)

[15] Ratschan, S.: Efficient Solving of Quantified Inequality Constraints over the Real Numbers. ACM Trans. Comput. Log. **7**(4), 723–748 (2006)

[16] Scheibler, K.: Applying CDCL to Verification and Test: When Laziness Pays Off. Ph.D. thesis, University of Freiburg, Freiburg im Breisgau, Germany (2017)

[17] Scheibler, K., Neubauer, F., Mahdi, A., Fränzle, M., Teige, T., Bienmüller, T., Fehrer, D., Becker, B.: Accurate ICP-Based Floating-Point Reasoning. In: Piskac, R., Talupur, M. (eds.) FMCAD 2016. pp. 177–184. IEEE (2016)

[18] Silva, J.P.M., Sakallah, K.A.: GRASP: A Search Algorithm for Propositional Satisfiability. IEEE Trans. Computers **48**(5), 506–521 (1999)

[19] Tseitin, G.S.: On the Complexity of Derivations in Propositional Calculus. In: Studies in Constructive Mathematics and Mathematical Logics (1968)

[20] Zhang, L., Madigan, C.F., Moskewicz, M.H., Malik, S.: Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In: ICCAD 2001. pp. 279–285. IEEE Press, Piscataway, NJ, USA