SMT: Something you Must Try

Erika Ábrahám[1], József Kovács[1], and Anne Remke[2]

[1] RWTH Aachen University, Aachen, Germany
{abraham|kovacs}@cs.rwth-aachen.de
[2] Westfälische Wilhelms-Universität, Münster, Germany
{anne.remke}@uni-muenster.de

Abstract. SMT (Satisfiability Modulo Theories) solving is a technology for the fully automated solution of logical formulas. Due to their impressive efficiency, SMT solvers are nowadays frequently used in a wide variety of applications. These tools are general purpose and as off-the-shelf solvers, their usage is truly integrated. A typical application (i) encodes real-world problems as logical formulas, (ii) check these formulas for satisfiability with the help of SMT solvers, and - in case of satisfiability - (iii) decodes their solutions back to solutions of the original real-world problem.
In this extended abstract we give some insights into the working mechanisms of SMT solving, discuss a few areas of application, and present a novel application from the domain of simulation.

1 Introduction

Satisfiability checking [2] is a relatively young research area, aiming at the development of fully automated methods for checking the satisfiability of logical formulas. While there are interesting new developments to handle quantified formulas, our focus in this paper will be on *quantifier-free* formulas.

Starting with *SAT solving* for propositional logic, satisfiability checking algorithms have been developed also for numerous first-order-logic theories. The implementation of these technologies in *SMT (SAT Modulo Theories) solvers* [6] turned out to be extremely powerful. This success is due to several key enabling factors. From the practical side, there has been a strong community support, agreeing on a standard input language SMT-LIB, providing a large collection of SMT-LIB benchmarks [5], and organizing annual competitions [1]. From the theoretical side, algorithms from mathematical logic and symbolic computation have been adapted and integrated into satisfiability checking methods, guided by a heuristic and strategic view from the computer science perspective, and resulting in elegant and innovative algorithms with nice synergies between the two disciplines.

Besides some well-known SMT solvers AProVE [13], cvc5 [4], MathSAT5 [8], veriT [7], Yices2 [12] and Z3 [18], we mention our C++ programming library named *SMT Real-Algebraic Toolbox (SMT-RAT)* [9], whose characteristics is a clean modular structure allowing to combine different decision procedures into strategic SMT solving, with the main focus on solving real-algebraic problems.

These and a number of further SMT solvers are available as off-the-shelf tools. That means, they can be used in a black-box style as depicted in Figure 1: feed them with a logical description of a problem to be solved using the SMT-LIB syntax and, if the problem is satisfiable, get a solution from the SMT solver and extract from it a solution for the original problem.

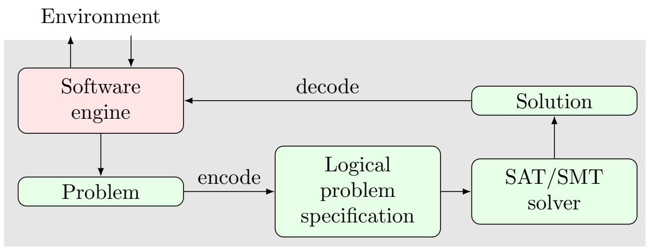

Fig. 1: Embedding SMT solvers in applications

However, unsurprisingly, also SMT solvers have bounded scalability. The way how a real-world problem is encoded logically has a major influence on the effectiveness of SMT solving. Thus, despite their off-the-shelf nature, for the users of SMT solvers it might be helpful to have an idea about the internal working mechanisms of the underlying algorithms.

Therefore, our objectives in this paper are (1) to give an intuition about how SMT solvers work and (2) to provide some examples that demonstrate how SMT solvers can be integrated in different algorithms for solving suitable sub-problems. In the following, we

– give some basic insights into the algorithmic background of SMT solving in Section 2,
– discuss some example application domains in Section 3,
– present a new application to integrate SMT solving in the simulation of hybrid Petri nets in Section 4, and
– conclude the paper with some remarks in Section 5.

2 Satisfiability Checking

2.1 SAT Solving

SAT solvers are designed to check the satisfiability of *propositional logic* formulas, which are Boolean combinations of Boolean variables called *propositions*.

The input first needs to be transformed into *conjunctive normal form (CNF)*, being a conjunction of *clauses*, each clause being the disjunction of *literals*, and each literal being either a proposition or the negation of a proposition. This transformation can be done with the method of Tseitin [25] in polynomial time and space on the cost of additional variables, yielding for each propositional logic formula a satisfiability-equivalent formula in CNF.

SAT solvers have got really impactful since the discovery of an elegant combination of *exploration*, *(Boolean constraint) propagation*, and *(Boolean conflict) resolution* [11, 17]. Instead of a formal description of a state-of-the-art SAT algorithm, we give an illustrative example.

– Assume an input CNF formula $(a \vee b) \wedge (a \vee \neg b)$, where a and b are propositions.
– Exploration might decide to check the existence of a solution if a is assigned *false*.
– Propagation detects that, if a is *false*, then the only chance to satisfy the first clause is assigning *true* to b.
– However, now the second clause is violated, because all of its literals are false. Boolean resolution[3], applied to the clause in conflict (i.e. the second clause) and the clause which implied the value for the last literal in the conflict clause (i.e. the first clause) yields the new clause (a).
– We backtrack by undoing assignments in reverse chronological order until the new clause is not violated anymore; in this example we undo all assignments.
– Propagation in the new clause detects that a needs to be *true*.
– Exploration can now choose any value for b, which will result in a full satisfying assignment.

2.2 SMT Solving

SMT solving typically extends SAT solving to be able to handle (quantifier-free) first-order-logic formulas over some theories. There are three different techniques which we describe in the following; their structures are illustrated in Figure 2.

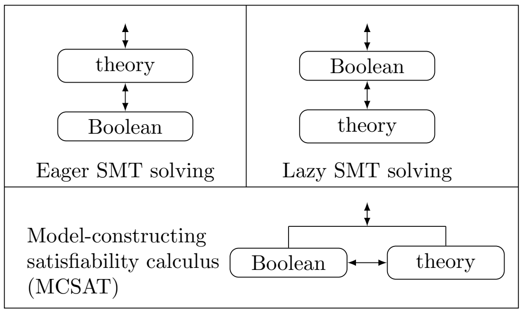

Fig. 2: The structure of SMT solving

Eager SMT solving For some theories, it is possible to transform their formulas to satisfiability-equivalent propositional logic formulas and use SAT solvers for their solution. This is the approach of *eager* SMT solving, where "eager" refers to the fact that the theory constraints are handled "eagerly", before handling the Boolean structure. Again, we avoid formal descriptions and illustrate the idea on an example for equality logic; similar approaches are available for e.g. uninterpreted functions and bit-vector arithmetic ("bit-blasting").

– Assume the formula $\varphi^E \; := \; x_1 = x_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3$ from equality logic, where the variables x_1, x_2 and x_3 can take values from some arbitrary but "large enough" domain.

[3] Given two clauses $(C_1 \vee b)$ and $(C_2 \vee \neg b)$ such that C_1 and C_2 are disjunctions of literals not referring to b, Boolean resolution on a proposition b can be used to derive the clause $(C_1 \vee C_2)$.

- We first replace each equation with a fresh proposition, encoding whether the given equation holds or not. This yields a *Boolean abstraction*, e.g. for our example $\varphi^{abs} := e_1 \wedge e_2 \wedge \neg e_3$.
- The Boolean abstraction is yet an over-approximation, i.e. it has more solutions that the input formula. We need to encode also the *transitivity* of equality by stating $\varphi^{tra} := (e_1 \wedge e_2) \rightarrow e_3$.
- Now $\varphi^{abs} \wedge \varphi^{tra}$ is equi-satisfiable to φ^E, i.e. checking the former for satisfiability(with a SAT solver) will answer also the satisfiability question for the latter.

(Less) lazy SMT solving In contrast to eager SMT solving, *lazy* SMT solving handles the Boolean structure first, before "lazily" considering the semantics of theory constraints. This approach first builds the over-approximative Boolean abstraction of the input problem and checks it for satisfiability using a SAT solver. If the Boolean abstraction is unsatisfiable, then also the input formula is unsatisfiable. Otherwise, given a solution for the Boolean abstraction, one or more theory solver(s) are asked to check the consistency of the Boolean solution in the theory.

- Consider again $\varphi^E := x_1 = x_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3$ and its Boolean abstraction $\varphi^{abs} := e_1 \wedge e_2 \wedge \neg e_3$.
- φ^{abs} is satisfiable by making e_1 and e_2 *true*, and e_3 *false*.
- A suitable theory solver for equations is asked whether the constraints $x_1 = x_2$, $x_2 = x_3$ and $x_1 \neq x_3$ are together satisfiable, which is not the case (due to the transitivity of equality).
- The theory solver returns an *explanation* in the form of a theory lemma, in this case $(e_1 \wedge e_2) \rightarrow e_3$.
- Refining the abstraction with this information makes it unsatisfiable. Thus the input formula is unsatisfiable.

Model constructing satisfiability calculus (MCSAT) [20] avoids master-slave structures and lets the SAT and the theory search evolve hand in hand in a consistent fashion. It does so by introducing exploration, propagation and conflict resolution also for the theory search, dually to the Boolean search.

- Assume again $\varphi^E := x_1 = x_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3$ over the real domain.
- To explore in the theory, we guess a value for x_1, e.g. 0.
- The Boolean search detects that $x_1 = x_2$ needs to hold. Theory propagation yields $x_2 = 0$.
- The Boolean search detects that $x_2 = x_3$ and $x_1 \neq x_3$ needs to hold. However, they have no common solution with $x_1 = x_2 = 0$.
- Thus our guess $x_1 = 0$ was wrong. We can generalize such a wrong gues by e.g. quantifier elimination. In this case, the guess can be generalized to the whole real domain, i.e. the formula cannot be satisfied for any value of x_1. Thus φ^E is unsatisfiable.

3 Applications

At the 2022 edition of the Computer-Aided Verification (CAV) conference, Neha Rungta from Amazon suggested in her keynote titled *A billion SMT queries a day* [23] that innovations at Amazon have "ushered in the golden age of automated reasoning".

Whereas in those applications, Amazon exploits SMT solving mainly for correctness reasoning, SAT and SMT solvers enjoy frequent usage in a diverse spectrum of further application domains. The aim of this section is to give an impression and a few examples about where and how SMT solvers can be employed to solve real-world problems.

3.1 A Toy Encoding Example

Let us start with a toy example to illustrate how a simple combinatorial problem can be encoded in linear real arithmetic. Assume that, after the Covid lockdown times, Eve is eager to make in 2023 scientific visits again.

- She has 100 travel wishes $A_1, \ldots, A_{100}$.
- She is allowed to make only 5 travels.
- She wants to be physically at $A_1 = iFM'23$.
- To coordinate a project, she needs to visit either A_2 or A_3.
- Travel A_i costs C_i EUR.
- Eve can spend up to C EUR.
- Travel A_i takes T_i days.
- Eve wants to travel at least T days.

The following linear real arithmetic formula encodes the solutions to the above problem. Besides the constants used in the problem specification, it uses for each $i \in \{1, \ldots, 100\}$ variables (i) $a_i \in \{0, 1\}$ to encode whether Eve chooses travel A_i ($x_i = 1$) or not ($x_i = 0$) and (ii) c_i and t_i for the costs and time for travel A_i, which are 0 if A_i is not chosen:

$$\left(\bigwedge_{i=1}^{100} \Big((a_i = 0 \wedge c_i = 0 \wedge t_i = 0) \ \vee \ (a_i = 1 \wedge c_i = C_i \wedge t_i = T_i\ ) \Big) \right) \wedge$$
$$\left(\sum_{i=1}^{100} a_i \leq 5 \right) \wedge (a_1 = 1) \wedge (a_2 = 1 \vee a_3 = 1) \wedge \left(\sum_{i=1}^{100} c_i \leq C \right) \wedge \left(\sum_{i=1}^{100} t_i \geq T \right)$$

3.2 Planning with Optimization Modulo Theories

With the advent of Industry 4.0, increasing automation in production processes poses new challenges on production management. The RoboCup Logistics League (RCLL) [21] has been proposed to study these challenges at a comprehensible and manageable scale.

In an RCLL application, two teams of robots share a work space, as illustrated in Figure 3. Each team consists of three robots and owns a set of machines

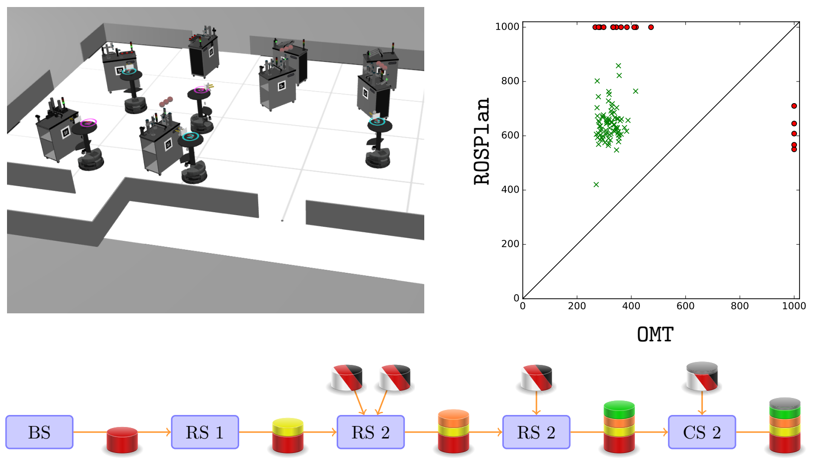

Fig. 3: Planning with optimization modulo theories. Source: E. Ábrahám, G. Lakemeyer, F. Leofante, T. D. Niemüller, A. Tacchella: PhD Leofante, publications in IJCAI'20, Information Systems Frontiers 2019, ECMS'19, AAAI'18, iFM'18, ICAPS'17, PlanRob'17, IRI'17.

(e.g. BS, RS1, RS2 or CS2) which they can use to produce certain products. Orders for products are announced dynamically during runtime. The production of each ordered product requires certain material and certain production steps to be executed under some partial order on certain machines with some required functionalities. The teams get rewards for each completed (or even partially completed) ordered product. The aim is to plan the production steps within a team in a collaborative manner to maximize the received rewards.

The complexity of the corresponding planning problem, due to e.g. temporal aspects, numerical quantities and the collaboration between the robots, makes its solution challenging. In a row of works, see e.g. [16], we proposed several ways to encode different sub-problems logically, and used SMT solvers for satisfiability checking as well as for optimization to solve those logically encoded sub-problems efficiently. Our methodology won the first place in the 2018 Planning and Execution Competition for Logistics Robots in Simulation, held at the International Conference on Automated Planning and Scheduling.

3.3 Reachability Analysis for Hybrid Systems with HyPro

Hybrid systems are systems with mixed discrete-continuous behavior. Typical examples are physical systems that are controlled by discrete controllers, like often present e.g. in the safety-critical automotive domain. *Formal methods* play an important role to assure the safety of such hybrid systems.

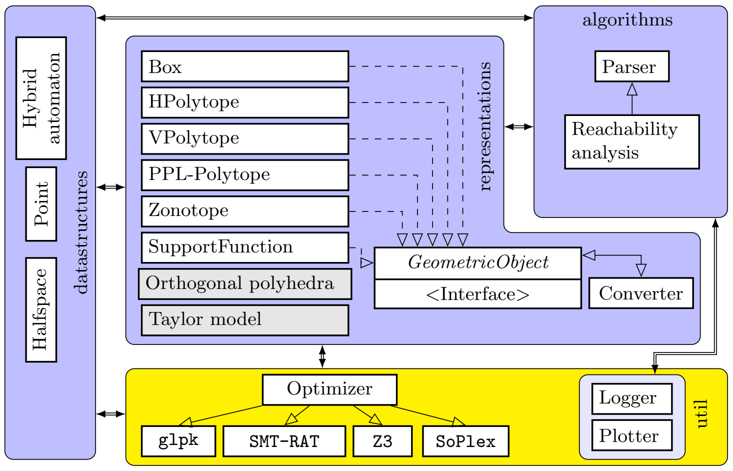

Fig. 4: The structure of the HyPro tool for computing reachability in hybrid systems. Source: S. Schupp, E. Ábrahám, I. Ben Makhlouf, S. Kowalewski. HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In Proc. of NFM'17.

To enable the usage of formal methods, hybrid systems need to be formalized in a suitable modeling language, like e.g. hybrid automata. Furthermore, we need algorithms to compute, for a given formal model and a given set of initial states, all the states that can be reached during the evolution of the system model. Since the reachability problem for hybrid systems is undecidable, most algorithms sidestep to over-approximative computations. Once we know (an over-approximation of) the set of all reachable states, we can check whether it includes any unwanted (dangerous or unsafe) states.

For the automation of these computations, we need *data structures* to represent state sets, along with all the operations on them which are needed for the reachability analysis, like e.g. the linear transformation of a state set, or the union, intersection or the Minkowski sum of two state sets. There are different state set representations are in use, which differ in their precision and efficiency. The HyPro C++ programming library [24] offers implementations for the most popular state set representations, and reachability algorithms using those representations.

The structure of HyPro is depicted in Figure 4. To implement the different set operations on the representations, we often need to solve arithmetic subproblems. Some of the available options to solve these sub-problems is to delegate them to some dedicated SMT solvers. Note that most SMT solvers accept command-line input in the SMT-LIB format, but their *application programming interface (API)* is not standardized (yet), such that adding a new SMT solver back-end requires a new wrapper around its API to fit the reachability analysis calls.

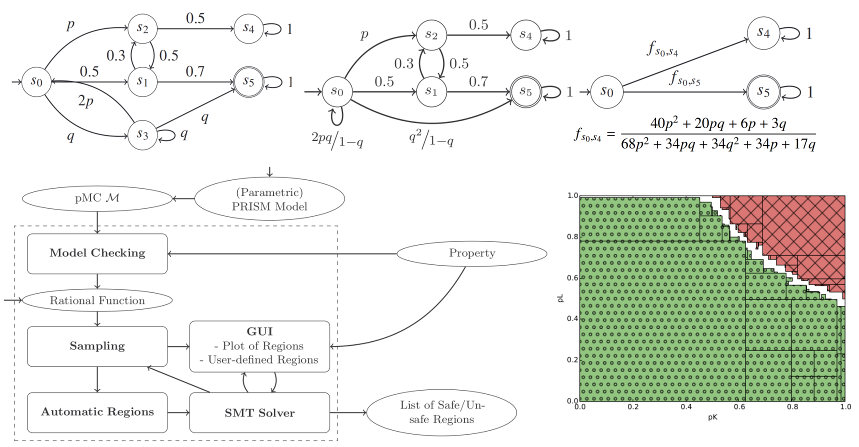

Fig. 5: Parameter synthesis for probabilistic systems. Source: C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, E. Ábrahám. PROPhESY: A probabilistic parameter synthesis tool. In Proc. of CAV'15.

Besides HyPro, also other analysis tools for hybrid systems reachability employ SMT solvers. Here we mention only dReach [15] with its dedicated SMT engine named dReal.

3.4 Parameter Synthesis for Probabilistic Systems

Discrete-time Markov chains (DTMCs) are a popular modeling formalism to describe systems whose behavior involves probabilistic behavior or whose behavior is influenced by uncertainties that can be quantified by probabilistic distributions. When the involved distributions are parameterized we talk about *parametric DTMCs*. Figure 5 depicts an example model on the top left. The circles are the system states; the state s_0 with the incoming arrow without a source state is the initial state. For each state, its outgoing transitions are labeled with probabilities, which should sum up to 1. If parametric expressions are involved, the valid domains for the parameters are such that the transition probabilities build valid probability distributions.

The PROPhESY tool has been developed to determine regions in the valid parameter domain, for which certain reachability probability bounds are provably satisfied or provably violated. For the example in Figure 5, the probability to reach the state s_4 from s_0 satisfies some fixed upper bound in the green areas, this bound is violated in the red areas, whereas no guarantees can be given for the white areas.

To compute such regions, first the reachability probability in question is computed symbolically (using state elimination, the result shown on the top right in the figure). Then an SMT solver is used to check whether all values in the parameter domain satisfy, respectively violate the probability bound. If it is the case then we can "color" the domain green respectively red, otherwise the domain is split into smaller sets for which the check is done recursively. The main challenge here is that for relevant systems, the expression that denotes the symbolic reachability probability can get extremely large with complex high-degree polynomial expressions involved.

4 Hybrid Petri Nets and Rate Adaption

In the previous section we reported on some existing SMT solver embeddings in a few domains. In this section, we propose a novel application of SMT solving in the area of modeling and simulation for hybrid Petri nets.

4.1 Hybrid Petri Nets

A *Petri net* $\mathcal{D} = (\mathcal{P}^d, \mathcal{T}^d, \mathcal{A}^d, \mathbf{m_0})$ as defined by [22], consist of a set of (discrete) *places* $\mathcal{P}^d$, a set of *transitions* $\mathcal{T}^d$, and a set of directed *arcs* $\mathcal{A}^d$, connecting places and transitions and vice versa.

A place $p_i \in \mathcal{P}^d$ contains a discrete number of *tokens* $m_i \in \mathbb{N}$. The *marking* of a Petri net is given as a vector $\mathbf{m} = \{m_1, \ldots, m_{|\mathcal{P}^d|}\}$ indicating the number of tokens currently contained in each place. The initial marking is given by $\mathbf{m_0}$.

Directed arcs are defined by $\mathcal{A}^d \subseteq (\mathcal{P}^d \times \mathcal{T}^d) \cup (\mathcal{T}^d \times \mathcal{P}^d)$. Transitions change the marking of connected places upon firing as follows: A transition $t \in \mathcal{T}^d$ removes a predefined number of tokens from every *input* place p that is connected via an arc (p, t), hence directed towards t. Correspondingly, it adds a number of tokens to every *output* place p that is connected via an arc (t, p) pointing away from t. A transition can only fire if it is *enabled*, i.e., if the marking has a sufficient number of tokens in every input place. To ease notation, here we assume that each transition removes exactly one token from each of its input places and adds exactly one token to each of its output places. Enabled transitions may fire independently and in an arbitrary order in the original definition of Petri nets.

Hybrid Petri nets, as introduced in [3] extend a Petri net $\mathcal{D}$ with time and *continuous* places, transitions and arcs. A hybrid Petri net is defined by the tuple $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathbf{m_0}, \mathbf{x_0}, \Phi)$, which adds an initial continuous marking $\mathbf{x_0}$ and a parameter function Φ to the sets of places $\mathcal{P}$, transitions $\mathcal{T}$ and arcs $\mathcal{A}$ and the initial discrete marking $\mathbf{m_0}$.

The parameter function specifies additional values for places, transitions and arcs $\Phi = (\phi_c^{\mathcal{P}}, \phi_d^{\mathcal{T}}, \phi_c^{\mathcal{T}}, \phi_s^{\mathcal{A}}, \phi_p^{\mathcal{A}})$, which will be formalised below.

The set of places $\mathcal{P} = \mathcal{P}^d \cup \mathcal{P}^c$ is composed from disjoint finite sets of discrete and continuous places. Continuous places $p_i^c \in \mathcal{P}^c$ have a continuous marking $x_i \in \mathbb{R}_0^+$, which is referred to as *fluid* and is lower bounded by 0. The parameter function $\phi_c^{\mathcal{P}} : \mathcal{P}^c \to (\mathbb{Q}_{\geq 0} \cup \{\infty\})$ assigns a *capacity* to every continuous place. We say that a place is *empty* if there is no fluid in it and *full* if the amount of fluid in it equals its capacity. In contrast, discrete places $p_i^d \in \mathcal{P}^d$ contain a discrete marking $m_i \in \mathbb{N}_0$ and have an unbounded (infinite) capacity.

The finite set of transitions $\mathcal{T} = \mathcal{T}^d \cup \mathcal{T}^c$ is composed from discrete transitions $\mathcal{T}^d$ which change the discrete marking and continuous transitions $\mathcal{T}^c$ which change the continuous marking.

Discrete transitions have a deterministic firing time, specified by the parameter function $\phi_d^{\mathcal{T}} : \mathcal{T}^d \to \mathbb{R}^+$. Every deterministic transition t_i^d is associated with a clock c_i, which if enabled evolves with $dc_i/dt = 1$, otherwise $dc_i/dt = 0$. Note that upon disabling, the clock value is preserved. A deterministic transition $t_i^d \in \mathcal{T}^d$ fires when c_i reaches the predefined transitions firing time. Discrete transitions with firing time zero are denoted as *immediate transitions* and fire as soon as they are enabled. If multiple immediate or deterministic transitions are supposed to fire at the same time, this so-called *conflict* is resolved using priorities and weights, which results in a probabilistic decision. For details on *conflict resolution* we refer to [3].

When enabled, continuous transitions $t^c \in \mathcal{T}^c$ fire continuously with a constant nominal flow rate assigned by $\Phi_c^{\mathcal{T}} : \mathcal{T}^c \to \mathbb{R}^+$. We refer to [10] for a detailed discussion of the concept of *enabling*.

Transitions and places are connected via the finite set of arcs $\mathcal{A}$. Discrete arcs $\mathcal{A}^d$ connect discrete places and transitions, while continuous arcs $\mathcal{A}^c$ connect continuous places and continuous transitions, respectively. Note that for simplicity, we omit guard and inhibitor arcs, as well as arc multiplicity and arc weights. Given a discrete marking $\mathbf{m}$, the above simplification leads to the marking $\mathbf{m}'$ after firing a discrete transition $t^d \in \mathcal{T}^d$, where

$$m_i' = \begin{cases} m_i - 1 & \textbf{iff } (p_i^d, t^d) \in \mathcal{A}^d \wedge (t^d, p_i^d) \notin \mathcal{A}^d, \\ m_i + 1 & \textbf{iff } (t^d, p_i^d) \in \mathcal{A}^d \wedge (p_i^d, t^d) \notin \mathcal{A}^d, \\ m_i & \text{otherwise.} \end{cases}$$

In addition to the firing of discrete transitions, the state of a hybrid Petri net changes continuously with time if at least one continuous transition is enabled. Again for simplicity, we restrict continuous transition t^c to have at most one *source* and one *target*, defined as follows:

- *source* : $\mathcal{T}^c \to (\mathcal{P}^c \cup \{\perp\})$: for all $t^c \in \mathcal{T}^c$, if there is a place p^c that is connected via an arc (p^c, t^c) then *source*$(t^c) = p^c$, otherwise *source*$(t^c) = \perp$,
- *target* : $\mathcal{T}^c \to \mathcal{P}^c \cup \{\perp\}$: for all $t^c \in \mathcal{T}^c$, if there is a place p^c that is connected via an arc (t^c, p^c), then *target*$(t^c) = p^c$, otherwise *target*$(t^c) = \perp$.

The *input bag* $I(p^c)$ and *output bag* $O(p^c)$ of a continuous place p^c are defined as follows:

- $I : \mathcal{P}^c \to 2^{\mathcal{T}^c}$: for all $p^c \in \mathcal{P}^c$, $I(p^c) = \{t \in \mathcal{T}^c \mid \mathit{target}(t^c) = p^c\}$ is the set of all transitions with target p^c.
- $O : \mathcal{P}^c \to 2^{\mathcal{T}^c}$: for all $p^c \in \mathcal{P}^c$, $I(p^c) = \{t \in \mathcal{T}^c \mid \mathit{source}(t^c) = p^c\}$ is the set of all transitions with source p^c.

A continuous transition can be disabled if a connected source place is empty or a connected target place is full. In order to model realistic physical behavior, the semantics of hybrid Petri nets *adapts* the nominal flow rate of transitions that are in the input bag of a full continuous place, as well as the nominal flow rate of transitions that are in the output bag of a an empty continuous place. Initially, all continuous transitions fire with their nominal rate, after *rate adaption* as described in Section 4.2 has taken place, the so-called actual flow rate $\theta(t^c)$ for $t^c \in \mathcal{P}^c$ may be smaller than the assigned nominal rate.

During time evolution, the continuous marking changes continuously with the *drift* specified for each continuous place as

$$d(p^c) = \sum_{t^c \in I(p^c)} \theta(t^c) - \sum_{t^c \in O(p^c)} \theta(t^c),$$

as the difference of the actual flow rates of the transitions to which p^c is connected as target or source.

4.2 Rate adaption

If the fluid of a continuous place reaches one of its boundaries, i.e. zero or its upper-bounding capacity, *rate adaption* reduces the actual flow rates of the affected continuous transitions based on the share $\phi_s^{\mathcal{A}}$ and priority $\phi_p^{\mathcal{A}}$ of the corresponding continuous arcs to prevent under- or overflow of that place.

Rate adaption is introduced in [3] as fixed-point iteration. The change of the actual flow rate of a continuous transition potentially modifies the drifts of continuous places, which in turn can trigger rate adaption for further continuous transitions. Hence, the algorithm needs to iteratively check all continuous places that are at either boundary.

In every iteration, for all continuous places at their upper capacity with a positive drift, the inflow is reduced to match the outflow and at all continuous places at the lower boundary with a negative drift, the outflow is reduced to match the inflow.

More specifically, the algorithm iterates over the continuous places that are at a boundary and per place computes the amount of *available fluid*. In case of a place at the lower boundary, the available fluid is given by the sum of the actual flow rates of the connected input transitions, which is then redistributed over the connected output transitions according to their priority and share. The case for full places is analogous, but with a reduction of the input transitions' rates.

The iteration terminates when all fluid places at the upper boundaries have a drift that is smaller or equal to zero and all fluid places at the lower boundaries have a drift that is greater or equal to zero.

As described in [14], the adaption of the actual flow rates can be challenging, depending on the structure of the Petri net at hand. If each continuous transition is involved in at most one conflict, the rate adaption proposed in [10] always terminates with a unique result. In the general case, Algorithm 5.7 from [10] applies an additional partitioning and ordering which assures termination with a unique result. However, different orderings might yield different results, which does not necessarily truly model real physical behavior.

As an example, consider the hybrid Petri net shown in Fig. 6. Continuous places are drawn as double circles and continuous transitions as double rectangles, full continuous places are filled black, empty places are white. The nominal flows are shown besides the transitions. Note that this example contains no discrete components and that transition t_3 is involved in two conflicts, namely at p_1 and at p_2.

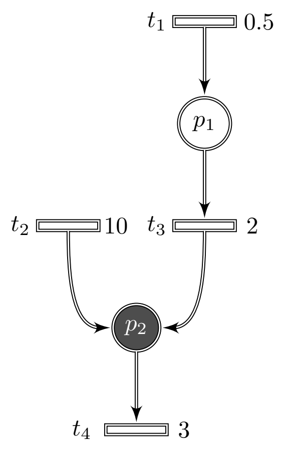

Fig. 6: Hybrid Petri net, where a transitions participates in two conflicts

If the conflict at p_2 is adapted first, its input transitions (t_2 and t_3) are reduced to match the outflow. Assuming equal priority and share, the actual flow rates are set to $\theta(t_2) = 2.5$ and $\theta(t_3) = 0.5$. With these actual flow rates rate adaption terminates, as the conflict at p_1 has been resolved "by chance".

However, if p_1 is processed first, it's outflow is reduced to match the inflow. The actual flow rate of t_3 is then set to $\theta(t_3) = 0.5$. As p_2 is still conflicted, it's incoming flow needs to be reduced by a factor of 2/7 to achieve the new rates $\theta(t_2) = 10 \cdot 2/7 = 20/7$ and $\theta(t_3) = 0.5 \cdot 2/7 = 1/7$, i.e. an inflow of 3 and thus a drift of 0. For this order, the outflow of p_1 is reduced stronger than needed and results in a positive drift at p_1. However, without an explicit control, in reality the drift would be always adapted to zero.

4.3 Formulation as an SMT Problem

Hence, while the ordering applied for rate adaption ensures termination with a unique result, it clearly influences the resulting actual flow rates and does not always yield results that are physically meaningful.

We propose to change the rate adaption mechanism from an iterative fixed-point search to an SMT-based approach: instead of iteratively adapting places, we suggest to use a logical formulation of the smallest fixed-points, and use an SMT solver for finding the required rates.

The advantages are the following: Firstly, out method always terminates, without any restrictions on the shape of the hybrid Petri net. Secondly, we design the formulation to assure not only satisfiability but also a unique solution. Thirdly, this unique solution truly reflects the natural physical behavior, keeping all rate adaptions minimal, i.e. just as small as needed to resolve conflicts.

To emphasize the key ideas, we assume equal priorities and share. Intuitively, for each empty place p in conflict (i.e. with a negative drift) we consider its output bag and reduce the contained transitions with the same factor. However, some transitions from its output bag may be reduced by other places by more than what would be needed to bring the drift of p to zero according to the fixed-point.

In the SMT-based approach, this problem of mutual reduction is circumvented by introducing the concept of *ownership*. We call the place which implies a strongest reduction on a transition its *owner*. Each place then only reduces the

$$
\begin{array}{ll}
(1) & \left(\bigwedge_{p\in P} 0 \leq \texttt{factor}_p \leq 1\right) \wedge \left(\bigwedge_{t\in T_a} 0 \leq \texttt{factor}_t \leq 1\right) \wedge \\
(2) & \left(\bigwedge_{t\in T_a} ((\texttt{owner}_t = \mathit{source}(t) \wedge \texttt{owner}_t \in P_e) \vee (\texttt{owner}_t = \mathit{target}(t) \wedge \texttt{owner}_t \in P_f))\right) \wedge \\
(3) & \Big(\bigwedge_{p\in P} \texttt{in}_p = (\sum_{t\in I(p)\cap T_a} \texttt{factor}_t \cdot \Phi_c^{\mathcal{T}}(t)) + (\sum_{t\in I(p)\cap T_{na}} \Phi_c^{\mathcal{T}}(t)) \wedge \\
 & \qquad \texttt{out}_p = (\sum_{t\in O(p)\cap T_a} \texttt{factor}_t \cdot \Phi_c^{\mathcal{T}}(t)) + (\sum_{t\in O(p)\cap T_{na}} \Phi_c^{\mathcal{T}}(t)) \quad \Big) \wedge \\
(4) & \Big[\bigwedge_{p\in P_e} \Big(\big(\texttt{factor}_p = 1 \vee \bigvee_{t\in O(p)} \texttt{owner}_t = p\big) \wedge \\
 & \qquad \big(\bigwedge_{t\in O(p)} (\texttt{owner}_t = p \rightarrow \texttt{factor}_t = \texttt{factor}_p) \wedge \\
 & \qquad\qquad (\texttt{owner}_t \neq p \rightarrow \texttt{factor}_t < \texttt{factor}_p) \quad \big) \wedge \\
 & \qquad \texttt{in}_p \geq \texttt{out}_p \wedge (\texttt{factor}_p < 1 \rightarrow \texttt{in}_p = \texttt{out}_p) \quad \Big) \Big] \wedge \\
(5) & \Big[\bigwedge_{p\in P_f} \Big(\big(\texttt{factor}_p = 1 \vee \bigvee_{t\in I(p)} \texttt{owner}_t = p\big) \wedge \\
 & \qquad \big(\bigwedge_{t\in I(p)} (\texttt{owner}_t = p \rightarrow \texttt{factor}_t = \texttt{factor}_p) \wedge \\
 & \qquad\qquad (\texttt{owner}_t \neq p \rightarrow \texttt{factor}_t \leq \texttt{factor}_p) \quad \big) \wedge \\
 & \qquad \texttt{in}_p \leq \texttt{out}_p \wedge (\texttt{factor}_p < 1 \rightarrow \texttt{in}_p = \texttt{out}_p) \quad \Big) \Big]
\end{array}
$$

Fig. 7: SMT encoding for the novel rate adaption mechanism

rates of the transitions it owns by the same factor, namely by a maximal factor that just assures zero drift for p.

Recall that the capacity of a place $p \in \mathcal{P}^c$ is given $\Phi_c^{\mathcal{P}}(p)$, while the current amount of fluid for all continuous places is stored in the continuous marking $\mathbf{x}$, where $\mathbf{x}(p_i) = x_i$ for $p_i \in \mathcal{P}^c$.

For readability we introduce some relevant sets of continuous places and continuous transitions for rate adaption. Since we focus on the continuous part of a hybrid Petri net, we omit the superindex c for readability in the newly defined sets.

- P_e: the set of continuous places that are empty, i.e. all $p \in \mathcal{P}^c$ with $\mathbf{x}(p) = 0$.
- P_f: the set of continuous places that are full, i.e. all $p \in \mathcal{P}^c$ with $\mathbf{x}(p) = \phi_c^{\mathcal{P}}(p)$
- $P = P_e \cup P_f$.

- T: the set of all transitions $t \in \mathcal{T}^c$ that are connected to at least one place from P, i.e. $\{\mathit{source}(t), \mathit{target}(t)\} \cap P \neq \emptyset$.
- T_a: the set of all potentially *adaptable* transitions $t \in T$ either with empty source or with full target, i.e., $\mathbf{x}(\mathit{source}(t)) = 0 \vee \mathbf{x}(\mathit{target}(t)) = \phi_c^{\mathcal{P}}(\mathit{target}(t))$.
- $T_{na} = T \setminus T_a$: the set of all *non-adaptable* transitions from T.

Our encoding, shown in Figure 7, makes use of the following variables:

- For all $p \in P$, the variables $\texttt{in}_p$ and $\texttt{out}_p$, both with domain $\mathbb{Q}_{\geq 0}$, encode p's inflow resp. outflow after rate adaption: $\texttt{in}_p = \sum_{t \in I(p)} \theta(t)$ and $\texttt{out}_p = \sum_{t \in O(p)} \theta(t)$.
- For all $t \in T_a$, $\texttt{owner}_t$ with domain P denotes the owner place of transition t.
- For all $p \in P$, by $\texttt{factor}_p$ with domain $[0,1] \subset \mathbb{Q}$ we encode the reduction factor for the rates of all transitions owned by p. All other transitions from p's output resp. input bag need to be reduced by other places by a factor at most $\texttt{factor}_p$ (i.e. at least as strongly reduced as p's reduction factor).
- For all $t \in T_a$, we encode by $\texttt{factor}_t$ with domain $[0,1] \subset \mathbb{Q}$ the rate reduction factor for transitions t, i.e. the reduction factor of its owner place.

The encoding consists of 6 main components:

(1) All rate adaption factors *reduce* the nominal rates and do not change their signs.
(2) The owner of each adaptable transition is either its source place or its target place.
(3) The inflow of a place p accumulates the actual flow rate of all transitions from p's input bag, i.e. the transition's rate adaption factor multiplied by its nominal rate. The case for the outflow of a place is analogous.
(4) A reducing *empty* place p must own some transitions from its *output* bag (or otherwise the place is not reducing, i.e. it has reduction factor 1). All transitions owned by p are reduced with the same factor, namely the place's reduction factor. All other transitions from the output bag must be reduced by other places by a smaller factor (i.e. stronger). Finally, the last line in part (4) assures that the rate reduction resolve all conflicts, but it poses only as much restrictions as needed.
(5) The case for *full* places p is analogous but argues about rate reductions for the transitions in p's *input* bag.

5 Some Final Remarks

After providing a general overview of satifiability checking, the paper showcased its usefulness and applicability on three examples. First a toy example was presented, before applications in (i) planning with optimization modulo theories and (ii) reachability analysis for hybrid systems were given. Summarizing, one of the main advantages of SMT solvers is their use as black box, which entails that the rapid development of SMT solvers automatically improves these applications. This is witnessed by the the wide rage of applications [19], such as optimization, proof and certificate generation, as well as satisfiability checking of quantified formulas.

The remainder of the paper illustrated an additional use-case for SMT solving in more detail: the analysis and simulation of hybrid Petri nets require a so-called rate adaption algorithm, which is traditionally implemented as fixed-point algorithm which converges to a unique result under very specific constraints. This paper proposes a more elegant approach, by formalising rate adaption as SMT problem, which can then be solved by state-of-the-art SMT solvers. Future work will introduce an additional iterative algorithm which is guaranteed to converge to a unique solution. Furthermore, we will proof that this unique solution equals the solution of the SMT encoding, presented here.

References

1. https://smt-comp.github.io/2023/
2. Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
3. Alla, H., David, R.: Continuous and hybrid Petri nets. Journal of Circuits, Systems, and Computers **8**(01), 159–188 (1998)
4. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Proc. of TACAS'22. LNCS, vol. 13243, pp. 415–442. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9"̇24, https://doi.org/10.1007/978-3-030-99524-9_24
5. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org (2016)
6. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 26, pp. 825–885. IOS Press (2009)
7. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: Proc. of CADE-22. LNCS, vol. 5663, pp. 151–156. Springer (2009)
8. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT solver. In: Proc. of TACAS'13, LNCS, vol. 7795, pp. 93–107. Springer (2013)
9. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In: Proc. of SAT 2015. LNCS, vol. 9340, pp. 360–368 (2015). https://doi.org/10.1007/978-3-319-24318-4˙26
10. David, R., Alla, H.: Discrete, continuous, and hybrid Petri nets, vol. 1. Springer (2010)
11. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7**(3), 201–215 (Jul 1960). https://doi.org/10.1145/321033.321034
12. Dutertre, B.: Yices 2.2. In: Proc. of CAV'14. LNCS, vol. 8559, pp. 737–744. Springer (2014)
13. Giesl, J., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Proving termination of programs automatically with AProVE. In: Proc. of IJCAR'14. LNAI, vol. 8562, pp. 184–191. Springer (2014)
14. Gribaudo, M., Remke, A.: Hybrid Petri nets with general one-shot transitions. Performance Evaluation **105**, 22–50 (2016)
15. Kong, S., Gao, S., Chen, W., Clarke, E.M.: dReach: δ-reachability analysis for hybrid systems. In: Proc. of TACAS'15. LNCS, vol. 9035, pp. 200–205. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0"̇15, https://doi.org/10.1007/978-3-662-46681-0_15

16. Leofante, F.: Optimal Planning Modulo Theories. Ph.D. thesis, RWTH Aachen University, Germany (2020)
17. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**, 506–521 (1999)
18. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of TACAS'08. LNCS, vol. 4963, pp. 337–340. Springer (2008)
19. de Moura, L., Dutertre, B., Shankar, N.: A tutorial on satisfiability modulo theories. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification. pp. 20–36. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
20. de Moura, L.M., Jovanovic, D.: A model-constructing satisfiability calculus. In: Proc. of VMCAI'13. LNCS, vol. 7737, pp. 1–12. Springer (2013)
21. Niemueller, T., Lakemeyer, G., Ferrein, A.: The RoboCup Logistics League as a benchmark for planning in robotics. In: Proc. of PlanRob@ICAPS'15 (2015)
22. Petri, C.: Kommunikation mit Automaten. Ph.D. thesis, TU Darmstadt (1962)
23. Rungta, N.: A billion SMT queries a day (invited paper). In: Proc. of CAV'22. LNCS, vol. 13371, pp. 3–18. Springer (2022). https://doi.org/10.1007/978-3-031-13185-1“˙1, https://doi.org/10.1007/978-3-031-13185-1_1
24. Schupp, S., Ábrahám, E., Makhlouf, I.B., Kowalewski, S.: HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In: Proc. of NFM'17. LNCS, vol. 10227, pp. 288–294. Springer (2017). https://doi.org/10.1007/978-3-319-57288-8“˙20, https://doi.org/10.1007/978-3-319-57288-8_20
25. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning, pp. 466–483. Springer (1983)