Non-Classical Logics in Satisfiability Modulo Theories

Clemens Eisenhofer, Ruba Alassaf, Michael Rawson, and Laura Kovács

[1] TU Wien, Austria
[2] University of Manchester, UK
{clemens.eisenhofer,laura.kovacs,michael.rawson}@tuwien.ac.at
ruba.alassaf@manchester.ac.uk

Abstract. We show that tableau methods for satisfiability in non-classical logics can be supported naturally in SMT solving via the framework of user-propagators. By way of demonstration, we implement the description logic $\mathcal{ALC}$ in the Z3 SMT solver and show that working with user-propagators allows us to significantly outperform encodings to first-order logic with relatively little effort. We discuss extensions of our approach to theories and promote user-propagators for creating non-classical solvers based on tableau calculi.

Keywords: SMT · Non-Classical Logics · User-Propagators · Tableaux

1 Introduction

Satisfiability modulo theory (SMT) solvers, e.g. [3,11,22], mostly implement CDCL($\mathcal{T}$) [5,20] to combine propositional satisfiability (SAT) solving with theory-specific decision procedures. Due to the modular nature of the underlying CDCL($\mathcal{T}$) algorithm, not only can SMT solvers reason in combinations of theories, but it is even possible to add and control custom first-order theories by attaching new decision procedures, as recently introduced in the user-propagator framework [7]. The underlying logic in the SMT solving community is classical first-order logic. When moving towards non-classical logics, such as modal or description logics [2,8,16], tableau calculi provide common ground [10]. The resulting proof procedures behave very differently to SMT solvers [12,17].

In this paper, we argue that *it is time to join forces*. We show that tableau methods can be integrated naturally into SMT solving (Section 3). In so doing, we promote user-propagators [7] for guiding non-classical reasoning within SMT solving. We demonstrate our work within the Z3 SMT solver [22] and show that this approach outperforms two standard Z3 implementations based on quantification (Section 4). We discuss supporting other non-classical logics (Section 5).

SMT driven by instantiation rules from modal and description logic have been investigated [1,26,27], as has porting classical tableau rules to SMT [9]. Our work can be seen as a framework for implementing non-classical semantics, similar to MetTeL 2 [29,30] where solvers are automatically synthesized from tableau rules. Translation to first-order [24,25] or higher-order [14,15] classical logics exist, but

work poorly on satisfiable instances [19,31]. Our approach combines well with theory reasoning, thanks to CDCL($\mathcal{T}$). In addition, the user-propagator allows applying expert knowledge about the considered logic.

2 Background and Challenges

Background. We assume familiarity with classical first-order logic [28], SMT solving [6], and the description logic $\mathcal{ALC}$ [2]. To avoid confusion with first-order quantifiers, we use modal syntax to write $\mathcal{ALC}$ formulas φ as

$$\varphi ::= \top \mid A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Box_r\varphi$$

where A is a (theory[3]) atom and r a modality/role. The logical connectives $\Rightarrow$, $\wedge$, and $\bot$ are defined as usual. The modal operator $\Diamond_r$ is defined as the dual of $\Box_r$. We assume a problem in $\mathcal{ALC}$ is given by a *knowledge base* $\langle TBox, ABox\rangle$. Elements in $TBox$ are of the form $global(\varphi)$[4] and are intended to be true in all worlds. Elements in $ABox$ are of the form $w_i : \varphi$, asserting "φ holds in world w_i"; or $r_k : (w_i, w_j)$, asserting "r_k relates worlds w_i and w_j". In case no $ABox$ is given, we assume the existence of an implicit world w_0. The truth-value of a formula φ under such a Kripke interpretation is given as in [2].

SMT Challenges for First-Order Translation of Description Logics. We motivate our work by considering the $\mathcal{ALC}$ knowledge base

$$TBox = \{global(\Diamond_r(A \wedge \Diamond_r\neg A))\}. \tag{1}$$

One may reason about this formula by (i) translating it into classical first-order logic via the *standard translation* [8]; and (ii) using a decision procedure handling uninterpreted functions and quantifiers to establish satisfiability of the translated formula. In particular, step (i) translates (1) into the first-order formula

$$\forall x(\exists y(reach^r(x, y) \wedge A(y) \wedge \exists z(reach^r(y, z) \wedge \neg A(z)))) \tag{2}$$

where $reach^r$ is an uninterpreted function symbol. Then, in step (ii) SMT solving over (2) instantiates the universally-quantified variable x with w_0, using for example model-based quantifier instantiation (MBQI) [13]. Skolemization introduces two new constants w_1 and w_2, which results in the quantifier-free instance:

$$reach^r(w_0, w_1) \wedge reach^r(w_1, w_2) \wedge A(w_1) \wedge \neg A(w_2), \tag{3}$$

from which the partial interpretation

$$reach^r(x, y) : \text{ if } (((x = w_0 \wedge y = w_1) \vee (x = w_1 \wedge y = w_2))) \text{ then } \top \text{ else } *. \tag{4}$$

can be deduced. The symbol $*$ is undetermined and represents an arbitrary Boolean value. Assume that the SMT solver sets $*$ to $\bot$ in order to complete the

[3] this is an addition to the classical definition of $\mathcal{ALC}$

[4] we write the more usual form $\varphi_1 \sqsubseteq \varphi_2$ as $global(\varphi_1 \Rightarrow \varphi_2)$

rule: Some constraints $P_1, \ldots, P_n$

$sign_{1,1} : \varphi_{1,1} \in \mathcal{L}(w_{1,1})$	$\ldots$	$sign_{n,1} : \varphi_{n,1} \in \mathcal{L}(w_{n,1})$
$\ldots$	$\ldots$	$\ldots$
$sign_{1,m_1} : \varphi_{1,m_1} \in \mathcal{L}(w_{1,m_1})$	$\ldots$	$sign_{n,m_n} : \varphi_{n,m_n} \in \mathcal{L}(w_{n,m_n})$

Fig. 1. Abstract tableau calculus rule.

partial model (4) for checking (2). As the solver cannot derive equalities among the world constants w_0, w_1, w_2, the solver has to check all three constants with respect to the universal quantifier of (2). As w_1 and w_2 violate the universal quantifier, further constants are generated by Skolemization, but (2) remains violated and the sequence of MBQI steps repeat indefinitely. Choosing $\top$ for $*$ avoids such failure, but increases the burden of SMT solving, as the solver must consider all potential relations among all constants (here, w_0, w_1 and w_2) and eliminate such relations again as they lead to conflicts. Randomly choosing $\top$ or $\bot$ for completing the partial model (4) of (2) is not a solution either, as it combines the disadvantages of both approaches.

3 Tableau as a Decision Procedure in CDCL($\mathcal{T}$)

Addressing the above challenges, we advocate user-propagators for tailored SMT solving, providing efficient implementations of custom tableau reasoners. We propose using the lemma generation process of CDCL($\mathcal{T}$), explained below, to simulate rule application of tableau calculi.

In a nutshell, the CDCL($\mathcal{T}$) infrastructure [5] introduces fresh Boolean variables to name theory atoms of an input formula; the resulting propositional skeleton is then solved by an ordinary SAT solver. If a propositional model is found, theory solvers are asked if the model is correct with respect to theory atoms. These specialized procedures may introduce further "lemma" formulas to the Boolean abstraction or report conflicts directly, forcing the SAT solver to "correct" the Boolean interpretation. This is repeated until all theory solvers agree on the Boolean assignment or the Boolean abstraction becomes unsatisfiable.

User-Propagators in CDCL($\mathcal{T}$) with tableau methods. Our solution builds a custom reasoner using the user-propagator framework [7]. Algorithm 1 shows underlined parts relevant for the following discussion. The custom reasoner is implemented by providing the methods `push`, `pop`, `fixed` and `final` in some programming language. The method `abstr(f)` is a method to be applied *a priori* solving. All other methods are those of the SMT solver.

We can simulate a tableau calculus whose rules are of the abstract form shown in Figure 1[5]. Each P_i is either containment, $sign : \circ(\bar{\varphi}) \in \mathcal{L}(w)$, or non-containment, $sign : \circ(\bar{\varphi}) \notin \mathcal{L}(w)$, with $\bar{\varphi}$ a sequence of arbitrary operands, and $\mathcal{L}(w)$ a label[6]. Rules may only add signed formulas to labels and create new

[5] a possible implementation is shown in Algorithm 2 in the appendix

[6] a set of formulas with known truth-value for some node w on the current branch

Algorithm 1: Simple CDCL($\mathcal{T}$) Algorithm.
Methods that can be provided by a user-propagator are underlined.

```
1  Method CDCLT(f):
2      f ← abstr(f)                                              ▷ Sect. 3.2
3      Loop
4          if conflict(f) then
5              if backtrack(f) = failed then return UNSAT
6              foreach s ∈ T-solvers do s.pop()                  ▷ Sect. 3.5
7          while can_unit_propagate(f) do assign(get_up(f))
8          if ¬contains_unassigned(f) then
9              foreach s ∈ T-solvers do s.push()                 ▷ Sect. 3.5
10             assign(guess_variable(f))
11         else
12             foreach s ∈ T-solvers do s.final()                ▷ Sect. 3.4
13             if ¬new_formulas_propagated() then return SAT
14 Method assign(x, value):
15     foreach s ∈ T-solvers do
16         if is_associated(s, x) ∧ is_relevant(x) then
17             s.fixed(x, value)                                 ▷ Sect. 3.3
```

branches. We consider *confluent, non-destructive tableaux* with *signed formulas* [28] and *explicit labelled nodes* [18], which are straightforward in our framework. Many calculi [10], including those for propositional logics, first-order logics, various modal/description logics, and many-valued logics, can naturally be expressed within Figure 1. The main steps of our work towards integrating tableau reasoning in SMT solving can be illustrated using the following running example.

Example 1 (Running Example). Consider the $\mathcal{ALC}$ knowledge base:

$$TBox = \{global(Hum \Rightarrow (\Box_p(Alive \Rightarrow age \leq recordLifespan) \wedge \Diamond_p Hum))\}$$
$$ABox = \{eva : Hum \vee \Diamond_f \neg Hum, \quad par : (eva, paul)\}$$

where $Alive$ (Alive), Hum (Human), and age depend on the current world, but $recordLifespan$ does not; age and $recordLifespan$ are of integral sort; p (parent) and f (friend) denote roles; and eva and $paul$ are named worlds.

3.1 SMT-LIB Encoding and Custom SMT Theory

To enable SMT-based tableau reasoning, we encode non-classical logic features directly in an extension of the SMT-LIB input standard [4]. In particular, we encode non-classical logic symbols with the help of uninterpreted function symbols and sorts, yielding an SMT theory of non-classical logic.

Example 2 ($\mathcal{ALC}$ Knowledge Base in SMT-LIB). For $\mathcal{ALC}$, we introduce the uninterpreted *Relation* and $World$ sorts and the following functions:

$box:$	$Relation \times B \mapsto B$	$dia:$	$Relation \times B \mapsto B$
$global:$	$B \mapsto B$	$world:$	$\emptyset \mapsto W$
$reachable:$	$Relation \times W \times W \mapsto B$		

where B is the sort of Booleans and *world* represents the current world[7]. Functions may have an extra "World" argument to denote their dependency on some world. With these syntactic features on top of SMT-LIB, Example 1 is encoded as

```
(declare-fun Hum (World) Bool)     (declare-fun Alive (World) Bool)
(declare-fun age (World) Int)      (declare-const recordLifespan Int)
(declare-const eva World)          (declare-const paul World)
(declare-const p Relation)         (declare-const f Relation)
(assert (global
    (=> (Hum world) (and
        (box p (=> (Alive world) (<= (age world) recordLifespan)))
        (dia p (Hum world))))))
(assert (global (=> (= world eva) (or (Hum world) (dia f (Rob world))))))
(assert (reachable p eva paul))
```

3.2 Preprocessing (`abstr`)

We next traverse the syntax tree of the parsed problem and introduce fresh user-function symbols to abstract away operands we want to observe. All instances of introduced user-functions are automatically *associated* with our user-propagator and thus Boolean assignments to those instances might be reported by the SMT core by calling the `fixed` method. We might add a node parameter of an uninterpreted sort to user-functions to store additional information, such as the current world in Kripke semantics. As we go, we build a tree-shaped *abstraction* data structure for keeping track of abstracted subformulas and efficiently applying tableau rules. Only the root of the abstraction is passed to the SMT solver. Furthermore, we apply (logic-specific) simplifications.

Example 3 (Preprocessing and Abstraction). Recall Example 1. We replace all operators handled by tableau rules[8] by fresh user-functions: here, for the occurrences of $\Box_r\varphi$, $global(\varphi)$, and for theory atoms. World-dependent terms and some operators, such as $\Box$, require a node argument denoting the world in which they are evaluated. To ease instantiating multiple instances of the formulas, we use an unbounded variable x as the node argument. We obtain the SMT abstraction of Example 1 given in Figure 2. G denotes applications of the *global*-rule, M applications of $\Box$, and T arbitrary theory atoms. $ABox$ elements are encoded directly by instantiating the node arguments accordingly (e.g., $\neg M_1^f(eva)$).

[7] which will be eliminated during preprocessing
[8] shown in detail in Figure 3

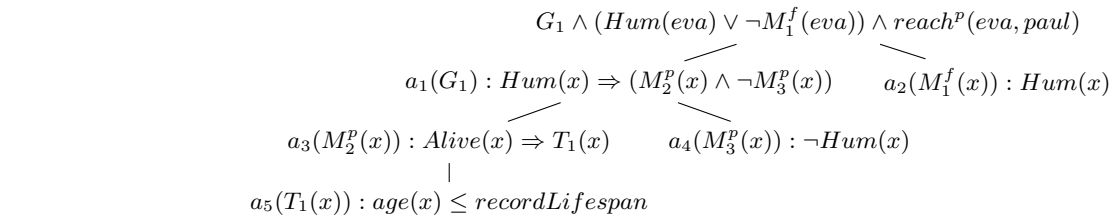

Fig. 2. Abstraction tree for Example 1. For simplicity, we rewrote $\Box_r A$ as $\neg\Diamond_r\neg A$.

3.3 Populating Languages (`fixed`)

Whenever the SAT core assigns a variable $V_i(w) \mapsto value$, we look up the operator $\circ$ and its operands abstracted by V_i during preprocessing. We add $\circ$, together with the auxiliary symbol and its operands $\bar{\varphi}_i$, to the respective label set[9] such that $\hat{\mathcal{L}}(w) := \hat{\mathcal{L}}(w) \cup \{(value : \circ, V_i, \bar{\varphi}_i)\}$ As the user-propagator reports only assignments to formulas that were previously abstracted away by user-functions, we might also need to abstract away other formulas for which we are not interested in adding additional rules, in order to be notified when these elements are added to some labels. For example, if we must observe $0 : (\varphi_1 \wedge \varphi_2) \in \mathcal{L}(w)$, we can replace $\wedge$ by a user-function. Usually, the tableau is closed (i.e. conflict) automatically if we have formulas of different sign. If the calculus has more complicated closing conditions, they can be reported explicitly by propagating a conflict.

Example 4 (Tracking Assignments to Arbitrary Subformulas). To keep track of all relevant Boolean assignments to atoms, we replace all atoms by user-functions, including complex theory atoms such as $age(w) \leq recordLifespan$ as shown in Figure 2. To preserve semantics, we add the definitions of the abstracted atoms by propagation For example, within Example 1 we might eagerly propagate

$$T_1(w) = value \vdash ((age(w) \leq recordLifespan) = value),$$

as soon as $T_1(w)$ is assigned the Boolean $value$.

3.4 Rule Application (`final`)

Whenever the solver found a Boolean assignment such that the propositional abstraction of its extended SMT problem (Section 3.1) is satisfied, we apply logic-specific tableau rules by iterating over the set $\hat{\mathcal{L}}(w)$ for every node w until no more tableau rules are applicable. A *propagation claim* is of the form $J_1, \ldots, J_m \vdash C$. An arbitrary number of them can be added by the user-propagator within *fixed* and *final*, indicating that the SAT core needs to assign $C \mapsto 1$ justified by the expressions $J_1, \ldots, J_m$; here, C may be an arbitrary Boolean expression.

[9] $\hat{\mathcal{L}}(w)$ are sets maintained by the user-propagator code to simulate $\mathcal{L}(w)$.

Consider a tableau rule R as in Figure 1 and assume that R is applied because $\{P'_1, \ldots, P'_m\} \subseteq \{P_1, \ldots, P_n\}$ are satisfied, obtaining

$$Just(P'_1), \ldots, Just(P'_m) \vdash C, \tag{5}$$

where $Just(P'_i)$ is J_i. We give C as a formula in disjunctive normal form (DNF)

$$\bigvee_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq m_i} (\varphi_{i,j}(w_{i,j}) = sign_{i,j}) \tag{6}$$

simulating application of rule R. We note that by using relevancy propagation [21] SMT solving may enjoy tableau-style branching, such that only one disjunct of the above DNF is chosen and reported assigned; unnecessary Boolean assignments are not reported to the user-propagator. We distinguish between two types of P'_i in (5): (i) those asserting elements are in the label, where P'_i is $sign : \circ(\bar{\varphi}) \in \mathcal{L}(w)$; and (ii) those that assert the opposite, where P'_i is $sign : \circ(\bar{\varphi}) \notin \mathcal{L}(w)$.

Justifying (i) is straightforward, as there must be an auxiliary user-function denoting that the respective element is contained in the label. We therefore have $sign : \circ(\bar{\varphi}), V, \bar{\varphi} \in \hat{\mathcal{L}}(w)$ and define $Just(P'_i)$ to be the equality $V = sign$. Case (ii) cannot be justified in general in our encoding because some assignments might not have been reported due to relevancy propagation. However, justifications for non-containment constraints may be omitted in the following scenarios:

1. The expression C can be simplified to $\top$ with respect to the current SAT assignment and hence lemma (5) and its justifications are irrelevant. Consider $M(w) \mapsto 0$ where $M(w)$ replaces $A \wedge B$ and $0 : A \in \hat{\mathcal{L}}(w)$[10]. Propagating $M(w) \vdash A(w) = \bot \vee B(w) = \bot$ has no effect, as the SMT solver detects that the consequent is already satisfied and ignores (5).
2. Applying R without satisfying the negative containment condition does not affect soundness or completeness and we make sure that we do not apply R infinitely often. Consider $M(w) \mapsto 0$ where $M(w)$ replaces $\Box A$. Applying this rule once or finitely often does not affect soundness or completeness in $\mathcal{ALC}$.

In either scenario, we do not justify that the respective conditions P'_i are satisfied, but only check P'_i before application of R (e.g. checking if a world is blocked). We hence set $Just(P'_i)$ to $\top$.

Example 5 (Applying Rules). Recall Example 1. Consider $1 : M_2^p \in \hat{\mathcal{L}}(eva)$, $0 : M_3^p \in \hat{\mathcal{L}}(eva)$ and $1 : G \in \hat{\mathcal{L}}$. SMT solving may propagate in `final`

$$M_3^p(eva) = \bot \vdash (\neg Hum(mary)) = \bot \wedge reach^p(eva, mary) = \top$$

by a $0 : \Box$-rule instance of Figure 1, where $mary$ is a fresh world. The next `final` callback might then propagate (because of the $1 : \Box$ rule and $1 : global$ rules)

$$\begin{aligned} M_2^p(eva) = \top \wedge reach^p(eva, mary) &= \top \\ &\vdash (Alive(mary) \Rightarrow T_1(mary) = \top \\ G_1 = \top \wedge reach^p(eva, mary) &= \top \\ &\vdash (Hum(mary) \Rightarrow (M_2^p(mary) \wedge \neg M_3^p(mary))) = \top. \end{aligned}$$

[10] for details on specific rules, refer to Figure 3

Table 1. Experimental results for benchmarks in the modal logic K.

	satisfiable (400)	unsatisfiable (185)	total (585)
standard translation	221 (55.3%)	81 (43.8%)	302 (51.6%)
model building	219 (54.8%)	78 (42.2%)	297 (50.8%)
user-propagator	**269 (67.3%)**	**132 (71.4%)**	**401 (68.5%)**

3.5 Backtracking (`push+pop`)

Backtracking in the CDCL core of SMT solving uses justifications provided for propagation claims. Our SMT-based tableau reasoner has to reset (*pop*) its state to a previously-saved state (*push*), by reverting labels $\hat{\mathcal{L}}(w)$. However, unlike tableau calculi, subformulas introduced by rule application may persist after backtracking because of conflict learning and similar techniques, which can result in the solver assigning these atoms unnecessarily. These spurious assignments correspond to adding elements to some label $\mathcal{L}(w)$ without a respective rule being applicable and hence, it might happen that $\hat{\mathcal{L}}(w) \neq \mathcal{L}(w)$. We can nonetheless apply rules resulting from spurious assignments as if they were not spurious: mostly, the solver will either justify the spurious elements anyway later or, in the case of a conflict, backtrack and undo these assignments.

Example 6 (Spurious Assignments). Recall Example 1. Suppose *paul* has a parent *mary*, generated by $M_3^p(paul) \mapsto 0$ using the $0 : \Box$-rule. Further, assume *mary* has a parent *sam*, generated by $M_3^p(mary) \mapsto 0$. On conflict, the SMT solver might backtrack to a state before assigning $M_3^p(paul) \mapsto 0$. The tableau-based theory solver removes $reach^p(sam)$ from $\hat{\mathcal{L}}(mary)$, as well as $reach^p(mary)$ from $\hat{\mathcal{L}}(paul)$. However, the solver may not "forget" the existence of atoms $M_3^p(mary)$ and $M_3^p(paul)$. It may therefore happen that $M_3^p(mary)$ is assigned later without first generating *mary* via $M_3^p(paul) \mapsto 0$. We ignore this spurious assignment, as the solver may later again assign $M_3^p(paul) \mapsto 0$, *ex post facto* justifying the existence of *mary*. If this justification is not given later and we encounter a conflict, the solver backtracks and removes the spurious assignment. If it leads to a model, we ignore everything in the model resulting from the spurious assignment.

4 Implementation and Experiments

We implemented[11] our tableau reasoning approach from Section 3 in the Z3 SMT solver [22]. We compare our implementation applying user propagation over the custom SMT theory of Section 3.1 against our implementation using two translations of modal logic to first-order logic, *viz.* the standard translation [8] and iterative deepening using cardinality assumptions. We considered altogether 400 satisfiable and 185 unsatisfiable benchmarks in the modal logic K [23]. Our initial experiments using a 60-second timeout are summarized in Table 1, showing that

[11] https://github.com/CEisenhofer/ModalZ3

applying our user-propagator framework performs the best[12]. This is partially so because quantifier reasoning in Z3 comes with MBQI overhead (Section 2). Finite model building performs poorly for large minimal models.

5 Conclusion and Discussion

We introduce an SMT-based reasoning framework for tableau methods, encoding tableau rules directly in SMT and applying user-propagators for custom reasoning. When implemented and evaluated using the Z3 SMT solver, our results outperform alternative encodings of the modal logic K. However, implementing logics via user-propagators *requires further knowledge about the considered non-classical logics* for tailored support towards, e.g., conflict learning and theory reasoning.

Beyond the Boolean Basis and Alternative Encodings. We so far considered an assignment $V \mapsto value$ to denote that $value : V \in \mathcal{L}(w)$ and only capture $value : V \notin \mathcal{L}(w)$ implicitly. This can be generalized to n mutually-exclusive truth values by using $\lceil log_2(n) \rceil$ Boolean variables. If, on the other hand, we need to justify that some element is *not* in our label, we can use a different encoding with each potential value encoded by a single Boolean. In this case, we use $bit_{sign}(V) = true$ to represent $V \in \mathcal{L}(w)$ instead of $V = sign$.

Example 7 (Ternary Logic). Consider a three-valued logic with values true, false, and undefined. The first encoding represents each truth value as a list of two bits where 00 represents false, 01 true, and 10 undefined respectively. The case of 11 is invalid. The second uses a list of three bits, one for each potential value. For each introduced subformula, we additionally propagate the cardinality constraint that exactly one bit has to be set to 1. This encoding incorporates the usual assumption that $value_1 : \circ \in \mathcal{L}(w)$ and $value_2 : \circ \in \mathcal{L}(w)$ with $value_1 \neq value_2$ represents a conflict, but could be dropped in cases where this is not desired.

Theories and Non-Classical Logic A challenging question arises when considering theories in combination with non-Boolean based logics. As we abstract away theory atoms (Example 3) and add them again on demand (Example 4), we can customize what and how theory atoms are passed to the SMT solver. For ternary logic, we might propagate the theory atom positively when assigned true, for false its negation, and nothing when the value is undefined.

Acknowledgements We thank Nikolaj Bjørner for discussions on this topic. We acknowledge funding from the ERC Consolidator Grant ARTIST 101002685, the TU Wien SecInt Doctoral College, and the FWF SFB project SpyCoDe F8504.

[12] see Figure 4 in the appendix for more details

References

1. Areces, C., Fontaine, P., Merz, S.: Modal satisfiability via SMT solving. In: Software, Services, and Systems. pp. 30–45 (2015). https://doi.org/10.1007/978-3-319-15545-6_5
2. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic (2017)
3. 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: TACAS. pp. 415–442 (2022). https://doi.org/10.1007/978-3-030-99524-9_24
4. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). http://SMT-LIB.org (2016)
5. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories. In: Handbook of Satisfiability - Second Edition, vol. 336, pp. 1267–1329 (2021)
6. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability - Second Edition, pp. 1267–1329 (2021). https://doi.org/10.3233/FAIA201017
7. Bjørner, N.S., Eisenhofer, C., Kovács, L.: Satisfiability Modulo Custom Theories in Z3. In: VMCAI. pp. 91–105 (2023). https://doi.org/10.1007/978-3-031-24950-1_5
8. Blackburn, P., van Benthem, J.: Modal logic: a semantic perspective. In: Handbook of Modal Logic, pp. 1–84 (2007). https://doi.org/10.1016/s1570-2464(07)80004-8
9. Bury, G., Cruanes, S., Delahaye, D.: SMT solving modulo tableau and rewriting theories. In: SMT (2018)
10. D'Agostino, M., Gabbay, D.M., Hähnle, R., Posegga, J.: Handbook of tableau methods (2013). https://doi.org/10.1007/978-94-017-1754-0
11. Dutertre, B.: Yices 2.2. In: CAV. pp. 737–744 (2014). https://doi.org/10.1007/978-3-319-08867-9_49
12. Fitting, M.: Tableau methods of proof for modal logics. Notre Dame J. Formal Log. **13**(2), 237–247 (1972). https://doi.org/10.1305/ndjfl/1093894722
13. Ge, Y., de Moura, L.M.: Complete instantiation for quantified formulas in satisfiability modulo theories. In: CAV. pp. 306–320 (2009). https://doi.org/10.1007/978-3-642-02658-4_25
14. Gleißner, T., Steen, A.: The MET: The art of flexible reasoning with modalities. In: RuleML+RR. pp. 274–284 (2018). https://doi.org/10.1007/978-3-319-99906-7_19
15. Gleißner, T., Steen, A., Benzmüller, C.: Theorem provers for every normal modal logic. In: LPAR. pp. 14–30 (2017). https://doi.org/10.29007/jsb9
16. Goré, R., Nguyen, L.A.: Analytic cut-free tableaux for regular modal logics of agent beliefs. In: CLIMA. pp. 268–287 (2007). https://doi.org/10.1007/978-3-540-88833-8_15
17. Goré, R., Olesen, K., Thomson, J.: Implementing tableau calculi using BDDs: BDDTab system description. In: IJCAR. pp. 337–343 (2014). https://doi.org/10.1007/978-3-319-08587-6_25
18. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In: LPAR. pp. 161–180 (1999). https://doi.org/10.1007/3-540-48242-3_11
19. Horrocks, I., Voronkov, A.: Reasoning support for expressive ontology languages using a theorem prover. In: FoIKS. pp. 201–218 (2006). https://doi.org/10.1007/11663881_12

20. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-Driven Clause Learning SAT Solvers. In: Handbook of Satisfiability - Second Edition, vol. 336, pp. 133–182 (2021)
21. de Moura, L., Bjørner, N.: Relevancy Propagation. Technical Report MSR-TR-2007-140, Microsoft Research, Tech. Rep. (2007), https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2007-140.pdf
22. de Moura, L.M., Bjørner, N.S.: Z3: An efficient SMT solver. In: TACAS. pp. 337–340 (2008). https://doi.org/10.1007/978-3-540-78800-3_24
23. Nalon, C., Hustadt, U., Papacchini, F., Dixon, C.: Local reductions for the modal cube. In: IJCAR. pp. 486–505 (2022). https://doi.org/10.1007/978-3-031-10769-6_29
24. Schmidt, R.A., Hustadt, U.: The axiomatic translation principle for modal logic. ACM Trans. Comput. Log. **8**(4), 19 (2007). https://doi.org/10.1145/1276920.1276921
25. Schneider, M., Sutcliffe, G.: Reasoning in the OWL 2 full ontology language using first-order automated theorem proving. In: CADE. pp. 461–475 (2011). https://doi.org/10.1007/978-3-642-22438-6_35
26. Sebastiani, R.: From KSAT to delayed theory combination: Exploiting DPLL outside the SAT domain. In: FroCoS. pp. 28–46 (2007). https://doi.org/10.1007/978-3-540-74621-8_2
27. Sebastiani, R., Vescovi, M.: Automated reasoning in modal and description logics via SAT encoding: the case study of K(m)/ALC-satisfiability. J. Artif. Intell. Res. **35**, 343–389 (2009). https://doi.org/10.1613/jair.2675
28. Smullyan, R.M.: First-order logic (1995). https://doi.org/10.1007/978-3-642-86718-7
29. Tishkovsky, D., Schmidt, R.A., Khodadadi, M.: MetTeL^2: Towards a tableau prover generation platform. In: PAAR. pp. 149–162 (2012). https://doi.org/10.29007/1c73
30. Tishkovsky, D., Schmidt, R.A., Khodadadi, M.: The tableau prover generator MetTeL2. In: JELIA. pp. 492–495 (2012). https://doi.org/10.1007/978-3-642-33353-8_41
31. Tsarkov, D., Riazanov, A., Bechhofer, S., Horrocks, I.: Using Vampire to reason with OWL. In: ISWC. pp. 471–485 (2004). https://doi.org/10.1007/978-3-540-30475-3_33

Appendix

$$\neg\ rule: \frac{1 : \neg\varphi \in \mathcal{L}(w)}{0 : \varphi \in \mathcal{L}(w)} \qquad \neg\ rule: \frac{0 : \neg\varphi \in \mathcal{L}(w)}{1 : \varphi \in \mathcal{L}(w)}$$

$$\wedge\ rule: \frac{1 : \varphi_1 \wedge \varphi_2 \in \mathcal{L}(w)}{\begin{array}{c}1 : \varphi_1 \in \mathcal{L}(w)\\ 1 : \varphi_2 \in \mathcal{L}(w)\end{array}} \qquad \wedge\ rule: \frac{0 : \varphi_1 \wedge \varphi_2 \in \mathcal{L}(w) \text{ and } 0 : \varphi_1, \varphi_2 \notin \mathcal{L}(w)}{0 : \varphi_1 \in \mathcal{L}(w) \mid 0 : \varphi_2 \in \mathcal{L}(w)}$$

$$\Box\ rule: \frac{1 : \Box_r\varphi \in \mathcal{L}(w_i) \text{ and } 1 : r(w_j) \in \mathcal{L}(w_i) \text{ and } w_i \text{ not blocked}}{1 : \varphi \in \mathcal{L}(w_j)}$$

$$\Box\ rule: \frac{0 : \Box_r\varphi \in \mathcal{L}(w_i) \text{ and } \nexists w_j(1 : r(w_j) \in \mathcal{L}(w_i) \wedge \varphi \in \mathcal{L}(w_j)) \text{ and } w_i \text{ not blocked}}{\begin{array}{c}0 : \varphi \in \mathcal{L}(w_j) \text{ with fresh } w_j\\ 1 : r(w_j) \in \mathcal{L}(w_i)\end{array}}$$

$$global\ rule: \frac{1 : global(\varphi) \in \mathcal{L}, w \text{ not blocked, and } w \text{ occuring in some } \mathcal{L}(w')}{1 : \varphi \in \mathcal{L}(w)}$$

$$individual\ rule: \frac{1 : (w : \varphi) \in \mathcal{L}}{1 : \varphi \in \mathcal{L}(w)} \qquad reach\ rule: \frac{1 : (r : (w_i, w_j)) \in \mathcal{L}}{1 : r(w_j) \in \mathcal{L}(w_i)}$$

with w being blocked iff $\exists succ_1, r_1, \ldots, succ_n, r_n(r_1(w) \in \mathcal{L}(succ_1) \wedge r_2(succ_1) \in \mathcal{L}(succ_2) \wedge \ldots \wedge r_n(succ_{n-1}) \in \mathcal{L}(succ_n) \wedge \mathcal{L}(w) \subseteq \mathcal{L}(succ_n))$

Fig. 3. Rules for the $\mathcal{ALC}$ Description Logic. For simplicity, we assume $TBox$ and $ABox$ elements are added with sign 1 to the label $\mathcal{L}$.

Algorithm 2: Naive Implementation with Nodes

```
1  Class UP:
2      stack⟨map⟨w, set⟨expr⟩⟩⟩ labels
3      abstraction_tree tree
4
5      Method abstr(formula):
6          foreach direct_subformula ∈ formula do
7              if is_special(direct_subformula) then
8                  aux ← fresh_user_function()
9                  replace(direct_subformula, aux)
10                 tree.add(aux, direct_subformula)
11     Method push:
12         labels.push(clone(labels.peek())
13     Method pop:
14         labels.pop()
15     Method fixed(expr = value):
16         stack.peek()[expr.arg(0)].add(value : expr)
17     Method final:
18         repeat
19             rule_applied ← false
20             foreach (w, label) ∈ labels.peek() do
21                 foreach value : expr ∈ label do
22                     rule ← tree.get_rule(value : expr)
23                     applied ← applied | rule.apply(w, tree.get(expr))
24         until rule_applied = false
```

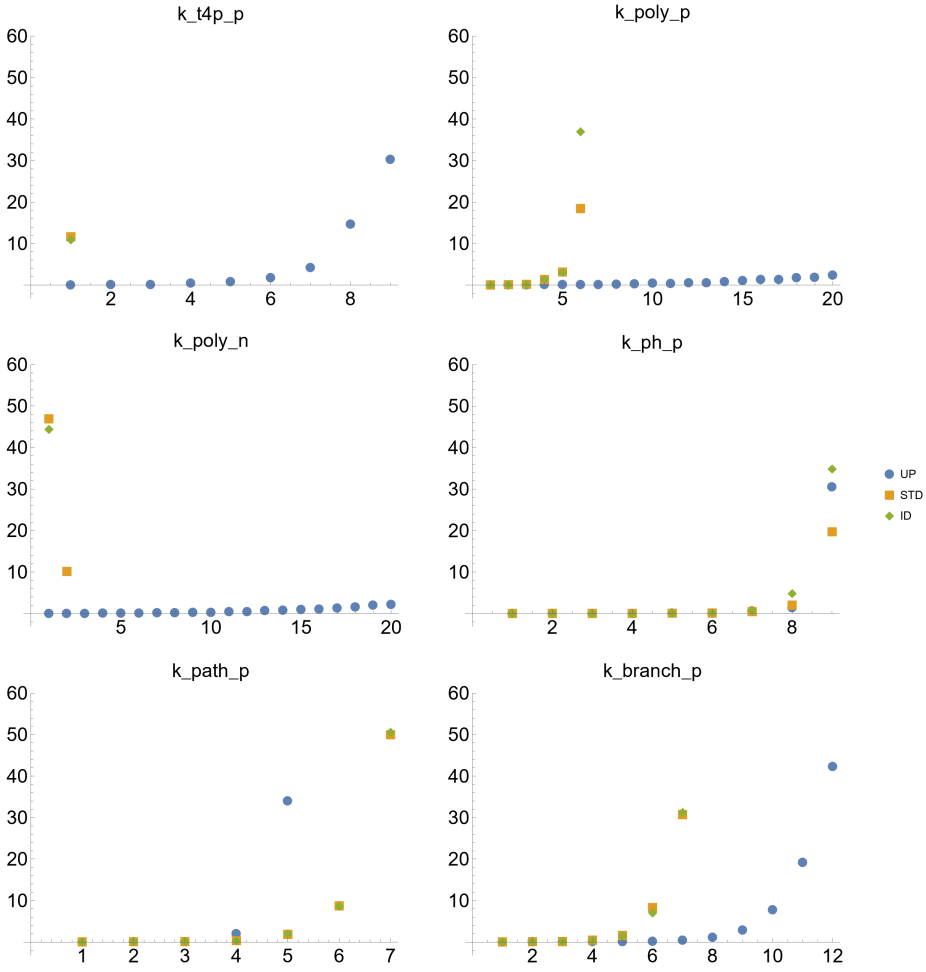

Fig. 4. Subset of the K Benchmarks Experiments