Non-Classical Logics in Satisfiability Modulo Theories

Clemens Eisenhofer\textsuperscript{1(2)}, Ruba Alassaf\textsuperscript{2}, Michael Rawson\textsuperscript{1}, and Laura Kovács\textsuperscript{1}

\textsuperscript{1} TU Wien, Vienna, Austria
\{clemens.eisenhofer, michael.rawson, laura.kovacs\}@tuwien.ac.at
\textsuperscript{2} University of Manchester, Manchester, UK
rub.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 $ALC$ 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 promote user-propagators for creating solvers for non-classical logics based on tableau calculi.

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

1 Introduction

Satisfiability modulo theory (SMT) solvers, e.g. \cite{4,14,29}, mostly implement CDCL($T$) \cite{6,27} to combine propositional satisfiability (SAT) solving with theory-specific decision procedures. Due to the modular nature of the underlying CDCL($T$) 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 \cite{8}. 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 \cite{2,9,21}, tableau calculi provide common ground \cite{13}. The resulting proof procedures behave very differently to SMT solvers \cite{16,22}.

In this paper, we argue that it is time to join forces. We show that tableau methods can be integrated naturally into SMT solving (Sect. 3). In so doing, we promote user-propagators \cite{8} for guiding non-classical reasoning within SMT solving. We demonstrate our work within the Z3 SMT solver \cite{29} and show that this approach outperforms two standard Z3 implementations based on quantification (Sect. 4). Finally, we discuss an alternative encoding for non-boolean based logics capable of dealing with explicit non-containment (Sect. 5).

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.
**Related Work.** SAT/SMT solving driven by instantiation rules from modal and description logic tableaux have been investigated [1,20,33], as has porting classical tableau rules to SMT [10], as has intuitionistic logic [12,15]. Our work applies user propagation as a framework for implementing non-classical logics, but also for theories that have tableau rules, such as strings [26] or finite sets [3]. MetTeL 2 [37,38] can automatically synthesize solvers from tableau rules expressed in a domain-specific input language: complex features that cannot be expressed in the input language can be implemented by manually changing the output program generated by the tool.

Another approach to non-classical logics translates non-classical input to SAT/SMT [11,23], first-order or higher-order logic [18,19,31,32,35,36] via a shallow embedding. After translation, a SAT/SMT solver or automatic theorem provers (ATPs) can be used for reasoning. ATPs typically work poorly especially on satisfiable instances from such translations [25,39,40]. Solvers do not usually take into account meta-logical properties of the considered non-classical logic. If at all, such properties are communicated to a solver via further lemmas or fine-tuning the solver’s configuration. Our approach allows us to directly encode expert knowledge of the considered logic. Additionally, our approach allows reasoning in multiple non-classical logics simultaneously and supports theory reasoning.

## 2 Background and Challenges

**Background.** We assume familiarity with basics of classical first-order logic [34], SMT solving [7], and the description logic $\mathcal{ALC}$ [2]. To avoid confusion with first-order quantifiers, we use modal syntax to write $\mathcal{ALC}$ formulas $\varphi$ as

$$
\varphi ::= \top | A | \neg \varphi | \varphi_1 \land \varphi_2 | \square_r \varphi
$$

where $A$ is a (theory\(^1\)) atom and $r$ a modality/role. The logical connectives $\Rightarrow$, $\land$, 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 $(TBox, ABox)$. Elements in $TBox$ are of the form $global(\varphi)\(^2\)$ and are intended to be true in all worlds. Elements in $ABox$ are of the form $w_i : \varphi$, asserting “$\varphi$ 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 $\varphi$ 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

$$
TBox = \{global(\diamond_r (A \land \diamond_r \neg A))\}. \tag{1}
$$

\(^1\) this is an addition to the classical definition of $\mathcal{ALC}$.

\(^2\) we write the more usual form $\varphi_1 \sqsubseteq \varphi_2$ as $global(\varphi_1 \Rightarrow \varphi_2)$.  

One may reason about this formula by (i) translating it into classical first-order logic via the standard translation [9]; 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

\[ \forall x (\exists y (\text{reach}^r(x, y) \land A(y) \land \exists z (\text{reach}^r(y, z) \land \lnot A(z)))) \]  

(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) [17]. Skolemization introduces two new constants w_1 and w_2, which results in the quantifier-free instance:

\[ \text{reach}^r(w_0, w_1) \land \text{reach}^r(w_1, w_2) \land A(w_1) \land \lnot A(w_2), \]  

(3)

from which the partial interpretation

\[ \text{reach}^r(x, y): \text{if } (((x = w_0 \land y = w_1) \lor (x = w_1 \land y = w_2))) \text{ then } \top \text{ else } *. \]  

(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 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 stepwise 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(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(T), explained below, to simulate rule application of tableau calculi.

In a nutshell, the CDCL(T) infrastructure [6] 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(T) with Tableau Methods.** Our solution builds a custom reasoner using the user-propagator framework [8]. 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 Fig. 1. We use signed formulas of the form sign : ◦(ϕ), where sign is a member of a fixed set, usually truth values, and ◦ is a logical operator applied to operands/subformulas ϕ. Each Pi asserts that a signed formula is (not) contained in a label L(w). Labels are sets of signed formulas with known sign at some node w on the current branch. Rules may only add signed formulas to labels and create new branches. We assume the input is satisfiable, in case no more rule is applicable.

This means, we consider sound, confluent, and non-destructive tableaux with signed formulas [34] and explicit labelled nodes [24], which are straightforward in
Algorithm 1: Simple CDCL($T$) Algorithm.
Methods that can be provided by a user-propagator are underlined.

```plaintext
1 Method CDCLT($f$):
2   $f \leftarrow \text{abstr}(f)$  ▷ Sect. 3.2
3   Loop
4     if conflict($f$) then
5       if backtrack($f$) = failed then return UNSAT  ▷ Sect. 3.5
6       foreach $s \in T$-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 $s \in T$-solvers do $s$.push()  ▷ Sect. 3.5
10      assign(guess_variable($f$))
11     else
12       foreach $s \in T$-solvers do $s$.final()  ▷ Sect. 3.4
13       if ¬new_formulas_propagated() then return SAT
14 Method assign($x, value$):
15   foreach $s \in T$-solvers do
16     if is_associated($s, x$) ∧ is_relevant($x$) then
17       $s$.fixed($x, value$)  ▷ Sect. 3.3
```

our framework. Many calculi [13], including those for propositional logics, first-order logics, various modal/description logics, and several many-valued logics, can naturally be expressed within Fig. 1. The main steps of our work towards integrating tableau reasoning in SMT solving can be illustrated using a running example in $\mathcal{ALC}$. The tableaux rules for $\mathcal{ALC}$ in our notation are detailed in Fig. 2.

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

$$
\begin{align*}
TBox &= \{ \text{global}(\text{Hum} \Rightarrow (\Box p(\text{Alive} \Rightarrow \text{age} \leq \text{recordLifespan}) \land \Diamond p\text{Hum})) \} \\
ABox &= \{ \text{eva} : \text{Hum} \lor \Diamond \neg\text{Hum}, \text{ par} : (\text{eva}, \text{paul}) \}
\end{align*}
$$

where $\text{Alive}$ ($\text{Alive}$), $\text{Hum}$ ($\text{Human}$), and $\text{age}$ depend on the current world, but $\text{recordLifespan}$ does not; $\text{age}$ and $\text{recordLifespan}$ are of integral sort; $p$ (parent) and $f$ (friend) denote roles; and $\text{eva}$ and $\text{paul}$ 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 [5]. 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 (ALC Knowledge Base in SMT-LIB). For ALC, we introduce the uninterpreted Relation and World sorts and the following functions:

<table>
<thead>
<tr>
<th>Relation</th>
<th>World</th>
</tr>
</thead>
<tbody>
<tr>
<td>$box$ : Relation $\times B \mapsto B$</td>
<td>$dia$ : Relation $\times B \mapsto B$</td>
</tr>
<tr>
<td>$global$ : $B \mapsto B$</td>
<td>$world$ : $\emptyset \mapsto W$</td>
</tr>
<tr>
<td>$reachable$ : Relation $\times W \times W \mapsto B$</td>
<td></td>
</tr>
</tbody>
</table>

where $B$ is the sort of Booleans and $world$ represents the current world\(^3\). 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

\[
\begin{align*}
(\text{declare-fun Hum (World) Bool}) & \quad (\text{declare-fun Alive (World) Bool}) \\
(\text{declare-fun age (World) Int}) & \quad (\text{declare-const recordLifespan Int}) \\
(\text{declare-const eva World}) & \quad (\text{declare-const paul World}) \\
(\text{declare-const p Relation}) & \quad (\text{declare-const f Relation}) \\
(\text{assert (global (=> (Hum world) (and (box p (=> (Alive world) (<= (age world) recordLifespan)) (dia p (Hum world))))))}) & \\
(\text{assert (global (=> (= world eva) (or (Hum world) (dia f (Rob world))))))}) & \\
(\text{assert (reachable p eva paul)}) &
\end{align*}
\]

3.2 Preprocessing (Abstr)

Next, we traverse the syntax tree of the parsed problem and introduce fresh user-function symbols to abstract away subformulas 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 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 Fig. 3. $G$ denotes applications of the global-rule, $M^r$ applications of $\Box_r$, and $T$ arbitrary theory atoms. ABox elements are encoded directly by instantiating the node arguments accordingly (e.g., $\neg M^f_1(eva)$).

\(^3\) which will be eliminated during preprocessing.
3.3 Populating Languages (Fixed)

Whenever the SAT core assigns a variable $V_i(w) \mapsto \text{value}$, 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\(^4\) such that $\hat{\mathcal{L}}(w) := \mathcal{L}(w) \cup \{(\text{value} : \circ, V_i, \bar{\varphi}_i)\}$ 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 $0: (\varphi_1 \land \varphi_2) \in \mathcal{L}(w)$, we can replace $\land$ 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 $\text{age}(w) \leq \text{recordLifespan}$ as shown in Fig. 3. To preserve semantics, we add the definitions of the abstracted atoms by propagation. For example, within Example 1 we might eagerly propagate 

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

as soon as $T_1(w)$ is assigned the Boolean $\text{value}$.

3.4 Rule Application (Final)

Whenever the solver found a Boolean assignment such that the propositional abstraction of its extended SMT problem (Sect. 3.1) is satisfied, we apply logicspecific 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.

---

\(^4\) $\hat{\mathcal{L}}(w)$ are sets maintained by the user-propagator code to simulate $\mathcal{L}(w)$.
Consider a tableau rule $R$ as in Fig. 1 and assume that $R$ is applied because \{${P'_1, \ldots, P'_m}$\} $\subseteq$ \{${P_1, \ldots, P_n}$\} are satisfied, obtaining
\[
Just(P'_1), \ldots, Just(P'_m) \vdash C,
\]
where $Just(P'_i)$ is $J_i$. We give $C$ as a formula in disjunctive normal form (DNF)
\[
\bigvee_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq m_i} (\varphi_{i,j}(w_{i,j}) = \text{sign}_{i,j})
\]
simulating application of the rule $R$. We note that by using relevancy propagation [28] 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 $\text{sign} : \circ(\vec{\varphi}) \in \mathcal{L}(w)$; and (ii) those that assert the opposite, where $P'_i$ is $\text{sign} : \circ(\vec{\varphi}) \notin \mathcal{L}(w)$.

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
\[
\text{sign} : \circ(\vec{\varphi}), V, \vec{\varphi} \in \hat{\mathcal{L}}(w)
\]
and define $Just(P'_i)$ to be the equality $V = \text{sign}$.

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 $F(w) \mapsto 0$ where $F(w)$ is a user-function used to replace $A \land B$ in some node $w$ (see $\land$ rule in Fig. 2) and $0 : A \in \hat{\mathcal{L}}(w)$. Propagating $F(w) \vdash A(w) = \bot \lor B(w) = \bot$ 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 $F(w) \mapsto 0$ where $F(w)$ replaces $\square A$ in some node $w$ (see $\square$ rule in Fig. 2). 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 $1: M^p_2 \in \hat{\mathcal{L}}(eva)$, $0: M^p_3 \in \hat{\mathcal{L}}(eva)$ and $1: G \in \hat{\mathcal{L}}$. SMT solving may propagate in final
\[
M^p_3(eva) = \bot \vdash (\neg \text{Hum}(mary)) = \bot \land \text{reach^p}(eva, mary) = \top
\]
by a 0: □-rule instance of Fig. 1, where mary is a fresh world. The next final callback might then propagate (because of the 1: □ and 1: global rules)

\[
M_2^p(eva) = \top \land \text{reach}^p(eva, mary) = \top
\]

\[
\vdash (\text{Alive}(mary) \Rightarrow T_1(mary) = \top)
\]

\[
G_1 = \top \land \text{reach}^p(eva, mary) = \top
\]

\[
\vdash (\text{Hum}(mary) \Rightarrow (M_2^p(mary) \land \neg M_3^p(mary))) = \top.
\]

### 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 restoring the value of \( \hat{L}(w) \) to the one it had in the previous state. 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 \( \hat{L}(w) \) without a respective rule being applicable and hence, it might happen that \( \hat{L}(w) \neq L(w) \). 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 \( M_3^p(paul) \mapsto \top \) using the 0: □-rule. Further, assume mary has a parent sam, generated by \( M_3^p(mary) \mapsto \top \). On conflict, the SMT solver might backtrack to a state before assigning \( M_3^p(paul) \mapsto \top \). The tableau-based theory solver removes \( \text{reach}^p(sam) \) from \( \hat{L}(mary) \), as well as \( \text{reach}^p(mary) \) from \( \hat{L}(paul) \). 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 \( M_3^p(paul) \mapsto \top \). We ignore this spurious assignment, as the solver may later again assign \( M_3^p(paul) \mapsto \top \), *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\(^5\) our tableau reasoning approach from Sect. 3 in the Z3 SMT solver [29]. We compare our implementation applying user propagation over the custom SMT theory of Sect. 3.1 against our implementation using two translations of modal logic to first-order logic, *viz.* the standard translation [9] and iterative deepening using cardinality assumptions. We considered altogether 400

\(^5\) https://github.com/CEisenhofer/ModalZ3.
satisfiable and 185 unsatisfiable benchmarks in the modal logic $K$ [30]. Our initial experiments using a 60-second timeout are summarized in Table 1, showing that applying our user-propagator framework performs the best. This is partially so because quantifier reasoning in Z3 comes with MBQI overhead (Sect. 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 $V = \text{sign}$.

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.
References

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.