

A Modern View on MCSat

Thomas Hader^{1,*}, Theo Jauschneg¹, Daniela Kaufmann¹ and Laura Kovács¹

¹TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria

Abstract

The Model Constructing Satisfiability (MCSat) approach has shown strong performance in solving complex SMT problems, in particular in algebraic SMT theories such as non-linear integer and real arithmetic. In this paper we revisit the theory-independent MCSat framework as a proof system to provide a modern perspective that refines the original formulation of MCSat.

By closely formalizing the implementation of MCSat within the Yices2 SMT solver, we incorporate design decisions that diverge from those in the seminal MCSat paper and thereby capture the current state-of-the-art in MCSat-based SMT reasoning.

We present a general, theory-agnostic rule scheme for MCSat and instantiate it for several theories, including propositional logic, non-linear real arithmetic, and uninterpreted functions. We provide several detailed examples to illustrate the applicability of the presented calculus.

Keywords

SMT, MCSat, automated reasoning

1. Introduction

The Model Constructing Satisfiability (MCSat) calculus represents an alternative approach to DPLL(T) in SMT solving by integrating model construction and theory reasoning within a single, unified framework. It was initially developed under the name NLSAT as an efficient approach to solve the satisfiability problem for the theory of quantifier-free non-linear real arithmetic (NRA) [1]. When NLSAT proved to be particularly effective, it was extended to encompass arbitrary theories and later published as MCSat [2, 3]. A theoretical view on the MCSat principles with focus on theory combination has been published in [?].

Unlike the traditional DPLL(T) approach, MCSat maintains a combined candidate model in all present theories throughout the solving process and extends it incrementally, allowing for more implicit (automatic) handling of complex theory combinations [2]. The MCSat-calculus uses conflict analysis to guide the search, drawing inspiration from Conflict-Driven Clause Learning (CDCL) in Boolean satisfiability (SAT) solving and conflict explanation techniques from constraint programming [4]. It, thus, can be seen as a generalization of the prevalent solving techniques from SAT solving to first-order theories where different theories are combined in a general search framework based on a common partial model. As the model search is handled by a common, theory-independent framework, individual theories are merely required to provide conflict explanation in case the partial assignment is detected inconsistent in the respective theory.

To this day MCSat remains to be the state-of-the-art approach in solving NRA and has since been implemented in many SMT solvers, such as Yices2 [5] and SMT-RAT [6] (as a stand-alone solver) as well as Z3 [7] (as a theory back-end for NRA in its original NLSAT version). However, over years of development, current implementations of MCSat may sufficiently differ from the formal presentation in the original literature [2]. This poses additional challenges for interested readers in understanding state-of-the-art MCSat procedures. In this work, we revisit the MCSat theory by presenting an updated definition of MCSat as a proof system that closely captures the implementation of the MCSat reasoning

24th International Workshop on Satisfiability Modulo Theories, July 24–25, 2026, Lisbon, Portugal

*Corresponding author.

✉ thomas.hader@tuwien.ac.at (T. Hader); theo.jauschneg@tuwien.ac.at (T. Jauschneg); daniela.kaufmann@tuwien.ac.at (D. Kaufmann); laura.kovacs@tuwien.ac.at (L. Kovács)

ORCID: [0009-0002-8920-5469](https://orcid.org/0009-0002-8920-5469) (T. Hader); [0000-0002-5645-0292](https://orcid.org/0000-0002-5645-0292) (D. Kaufmann); [0000-0002-8299-2714](https://orcid.org/0000-0002-8299-2714) (L. Kovács)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

engine of the Yices2 SMT solver. While an experienced reader will certainly see close similarity to the original formalization, we aim to provide a compact and example-driven presentation to enable novel readers an easy and understandable introduction to MCSat in general as well as its Yices2 implementation in particular. Based on SMT-LIB [8], we first provide necessary definitions (Section 2) to later define the search schema (Section 3), followed by some examples for theories (Section 4).

2. Preliminaries

Along the definitions of SMT-LIB (version 2.6 [8]), we define the SMT problem as the check for satisfiability in a many-sorted first-order logic with equality, limited to the quantifier free fragment, augmented with *background theories*. A background theory specifies (one or many) *sorts*. Each sort $\mathcal{S}$ contains a *domain* $D_{\mathcal{S}}$ which is a non-empty, possibly infinite, set. We define $\text{dom}(\mathcal{S}) = D_{\mathcal{S}}$. By slight abuse of notation, we sometimes simply write $\mathcal{S}$ for $D_{\mathcal{S}}$. Logical expressions in an SMT problem are defined using the language of *well-sorted terms*, thus each term is associated with a unique sort. Terms are constructed using *functions*. A function consists of a *function symbol* (i.e. its name) and a *signature*, an ordered list of sorts of parameters and the sort it returns. We write $f : \mathcal{S}_1 \times \dots \times \mathcal{S}_n \rightarrow \mathcal{S}$ to denote a function with symbol f with a signature that maps sorts $\mathcal{S}_1$ up to $\mathcal{S}_n$ to sort $\mathcal{S}$. A term t generated by f is of sort $\mathcal{S}$, denoted by $\text{sort}(t) = \mathcal{S}$. Functions without parameters are *constants*. An *interpretation* I is a total function that maps a function to its semantics, i.e. its meaning. To apply the interpretation I on an arbitrary formula or term a we write a^I . For a constant, this is an element of $\mathcal{S}$, for all other functions, this is a total function (in the mathematical sense). Functions can either be *interpreted* or *uninterpreted*. Interpreted functions have a fixed interpretation defined by the respective theory, e.g. addition in the theory of reals. Finding the semantics of an uninterpreted function is part of the search problem. A term t defined by function f is written as $f(t_1, \dots, t_n)$ where $t_1, \dots, t_n$ are terms, called *sub-terms* of t . We denote $\text{subterms}(t) = \{t_1, \dots, t_n\}$. We define the notion of *interpreted* and *uninterpreted* term accordingly. We further denote by $\text{terms}(t)$ the transitive closure of the sub-term relation. For better readability, we denote (dis-)equality inside of a term with $\approx$ and $\not\approx$.

An SMT *formula* φ is a term of the distinguished *Boolean* sort $\mathbb{B}$. Naturally, we have $\text{dom}(\mathbb{B}) = \{\text{true}, \text{false}\}$ and we can define all regular Boolean operators as interpreted functions accordingly. We write $\perp$ and $\top$ to denote the constant terms evaluated as false and true, respectively. A function c that maps to $\mathbb{B}$ is denoted a *constraint* and a term of c a *constraint term*. The sort $\mathbb{B}$ is available in all SMT problems by definition. The formula φ is *satisfiable* if there exists an interpretation I such that φ^I evaluates to true and we denote any such interpretation a *model* of φ .

As usual in literature, we assume that any Boolean structure in φ is expressed in conjunctive normal form (CNF). We denote a constraint or its Boolean negation a *literal*, a (disjunctive) set of literals a *clause* and treat formulas as (conjunctive) sets of clauses. We present a clause C as a tuple of literals and define $\text{terms}(C) = \bigcup_{l \in C} \text{terms}(l)$ and $\text{terms}(\mathcal{C}) = \bigcup_{C \in \mathcal{C}} \text{terms}(C)$ for a set of clauses $\mathcal{C}$.

Example 1. *Let*

$$\varphi = 1 + x < f(12.7)$$

be an SMT formula, with sorts propositional logic ($\mathbb{B}$), reals ($\mathbb{R}$) and integers ($\mathbb{Z}$). The interpreted function symbols are $F_i = \{<, +, 1, 12.7\}$ where $< : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$ is a constraint, $+ : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ and the constants 1 and 12.7 are of sort integer and real, respectively.¹ They are all interpreted in the standard way. The uninterpreted function symbols are $F_u = \{x, f\}$ where x is a function symbol of arity 0 of sort integer, and $f : \mathbb{R} \rightarrow \mathbb{Z}$. Note that we mix pre- and infix notation however appropriate.

Given an n -ary interpreted function f , we define the *evaluation function* eval_f to allow evaluating terms without a full interpretation function given. We thus extend each sort in the signature of f with the new special domain element undef . Given the signature of f and corresponding domain elements

¹Note that, by slight abuse of notation, we do not distinguish between the (constant) term 1 and its interpretation, the domain element $1 \in \text{dom}(\mathbb{Z})$.

$s_1 \in \mathcal{S}_1 \cup \{\text{undef}\}, \dots, s_n \in \mathcal{S}_n \cup \{\text{undef}\}$, then $\text{eval}_f(s_1, \dots, s_n) \mapsto s$ for the correct $s \in \mathcal{S} \cup \{\text{undef}\}$ according to the interpretation of f . Along with SMT-LIB, we require f to be a total function, thus it must return a domain element if all its parameters are domain elements. In case one or more parameters are undef, eval_f may still return a domain value, provided the value can be computed independently of the unspecified parameters; otherwise, it returns undef. For a term t defined by f , we may write eval_t for eval_f .

Example 2. Assume term $t = +(t_1, t_2)$ of sort $\mathbb{R}$ with $+$ as real addition. Thus, $\text{eval}_+(4, 3) = 7$, and $\text{eval}_+(2, \text{undef}) = \text{undef}$. Let $u = \times(t_1, t_2)$ be a multiplicative term in $\mathbb{R}$ then $\text{eval}_\times(0, \text{undef}) = 0$ and $\text{eval}_\times(2, \text{undef}) = \text{undef}$. Assume the constraint term $t_3 \leq 0$, then $\text{eval}_{\leq 0}(2) = \text{false}$.

3. MCSat Search Schema

To solve an SMT formula φ , we need to find an assignment to all terms in φ . For the rest of this text, we consider φ and the set of all terms in the search problem T as fixed.

A *term assignment* $t \mapsto \alpha$ assigns to a term $t \in T$ of sort $\mathcal{S}$ a value $\alpha \in \text{dom}(\mathcal{S})$. Given a set S of term assignments, we say that S is *unambiguous* if for every $(t_1 \mapsto \alpha_1) \in S$ and $(t_2 \mapsto \alpha_2) \in S$, whenever $t_1 = t_2$ then $\alpha_1 = \alpha_2$. As we only consider unambiguous sets of term assignments we can define the *term assignment function* of S as

$$\nu_S(t) = \begin{cases} \alpha & (t \mapsto \alpha) \in S \\ \text{undef} & \text{otherwise} \end{cases}$$

We say that S is *consistent*, denoted as $\text{consistent}(S)$, if and only if for all $(t \mapsto \alpha) \in S$ with $t = f(t_1, \dots, t_m)$ we have that $\text{eval}_f(\nu_S(t_1), \dots, \nu_S(t_m))$ is either undef or α . We define the closure $\bar{S}$ of a term assignment S inductively: We start with $\bar{S} = S$; as long as there is a term $t \in T$ such that $\nu_{\bar{S}}(t) = \text{undef}$ and $\text{eval}_t(\nu_{\bar{S}}(t_1), \dots, \nu_{\bar{S}}(t_m)) = \alpha$ with $\alpha \neq \text{undef}$, add $(t \mapsto \alpha)$ to $\bar{S}$. Intuitively, the assignment $\bar{S}$ extends S by exhaustively applying the evaluation functions until a fixed point is reached. Note that the closure of the empty term assignment set $\bar{\emptyset}$ contains all term assignments induced by the background theories. We assume that $\bar{\emptyset}$ is consistent. Further note that a consistent S does not guarantee $\bar{S}$ to be consistent. Given a term assignment S such that $\bar{S}$ is consistent and a yet unassigned term t , we denote $\text{feasible}(S, t) \subseteq \text{sort}(t)$ as the set of values that can be assigned to t such that $\bar{S}$ remains consistent.

Example 3. Assume a term assignment with integer terms x and y :

$$S = \{(x + y \approx 0) \mapsto \text{true}, x \mapsto 2, y \mapsto 3\}$$

then $\text{consistent}(S)$ holds. Towards constructing the closure $\bar{S}$, the evaluation function eval_+ is applied such that we get $\text{eval}_+(2, 3) = 5$ and add the term assignment $(x + y) \mapsto 5$ to $\bar{S}$. Then $\text{eval}_{\approx}(x + y, 0) = \text{false}$. This, however, contradicts $((x + y \approx 0) \mapsto \text{true}) \in S \subseteq \bar{S}$, therefore, $\bar{S}$ is not consistent.

Consider integer term z and the term assignment $S' = \{(z^2 \approx 1) \mapsto \text{true}\}$, then both S' and $\bar{S}'$ are consistent and $\text{feasible}(S', z) = \{-1, 1\}$.

A *trail* M is a sequence of term assignments that captures the current search state. We write $\llbracket \cdot \rrbracket$ to denote a trail and treat M as a set of term assignments whenever required. By slight abuse of notation, we write $\llbracket M, t \mapsto \alpha \rrbracket$ for the extension of trail M with an assignment to t . Each assignment on the trail is either a *decision* or a *propagation*. For a decision, the assigned value was chosen without any reasoning (usually) out of multiple options, on the other hand, a propagation assigns the only feasible value, i.e. $\text{feasible}(M, t) = \{\alpha\}$, supported by theory specific reasoning. To distinguish between propagated and decided term assignments on a trail we may write $t \mapsto^* \alpha$ or $t \mapsto \alpha$, respectively. The decision level of M is the number of decisions on M . Whenever appropriate, we append a suffix to a trail to make its decision level explicit. We may also write $t \mapsto_n \alpha$ to denote the n -th decision on a trail. During the search, we aim to extend M while keeping the consistency of $\bar{M}$.

We extend the term assignment function to evaluate clauses. Given a clause C , and a set of term assignments S , we define

$$\nu_S(C) = \begin{cases} \text{true} & \exists l \in C . \nu_S(l) = \text{true} \\ \text{false} & \forall l \in C . \nu_S(l) = \text{false} \\ \text{undef} & \text{otherwise} \end{cases}$$

We say that a term assignment S *satisfies* a set of clauses $\mathcal{C}$, denoted by $\text{satisfies}(S, \mathcal{C})$, if all clauses are satisfied:

$$\begin{aligned} \text{satisfies}(S, \mathcal{C}) &= \text{consistent}(\bar{S}) \\ &\quad \wedge (\forall t \in \text{terms}(\mathcal{C}) . \nu_{\bar{S}}(t) \neq \text{undef}) \\ &\quad \wedge (\forall C \in \mathcal{C} . \nu_{\bar{S}}(C) = \text{true}) \end{aligned}$$

We further say that S is *infeasible* for $\mathcal{C}$, denoted as $\text{infeasible}(S, \mathcal{C})$, when S cannot be extended such that the clauses are satisfied:

$$\text{infeasible}(S, \mathcal{C}) = \nexists S' . S' \supseteq S \wedge \text{satisfies}(S', \mathcal{C})$$

When we reach a trail M such that $\text{satisfies}(M, \mathcal{C})$, then we can construct a model I using $\bar{M}$ and deduce that $\mathcal{C}$ are satisfiable.²

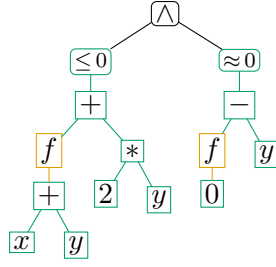
3.1. Theory Reasoning and Explanations

Theory Plugins. The general MCSat search is independent of any background theory. As mentioned above, the only sort natively supported in all MCSat style search procedures is the Boolean sort $\mathbb{B}$. In order to add an SMT-LIB theory including its sorts and interpreted function symbols, a *theory reasoning plugin* $\mathcal{T}$ is attached to the MCSat search. A plugin $\mathcal{T}$ has a set of sorts $\text{sort}(\mathcal{T})$ which values it can interpret as well as a set of functions $\text{func}(\mathcal{T})$ on those sorts. We say that a plugin can *handle* any sort that it has and *supports an interpreted function* f if it provides eval_f . While more than one plugin may be able to handle a particular sort, we require all plugins supporting an interpreted function to agree on its evaluation. A plugin $\mathcal{T}$ *supports an uninterpreted function* u by ensuring congruence, i.e. that two terms t_1, t_2 of u are assigned the same value if their sub-terms are assigned the same values. Usually there is one dedicated plugin to ensure congruence for all uninterpreted functions (see Section 4). Each interpreted or uninterpreted function must be supported by at least one theory plugin to ensure congruence and correct interpretation. We further say that a plugin *supports a term* if it is constructed by a supported function.

Assume a plugin $\mathcal{T}$ and a supported term t that contains a non-supported sub-term $t' \in \text{subterms}(t)$. Then $\mathcal{T}$ can handle $\text{sort}(t)$. For $\mathcal{T}$, t' is an atomic term that cannot be evaluated any further, but it can assign a value to t' and use any assigned value to evaluate t . We say that t' is a *variable* for $\mathcal{T}$. If a supported term t contains a non-supported $t' \in \text{subterms}(t)$, $\mathcal{T}$ treats t' as an atomic term, the same way as a constant that cannot be further evaluated. An uninterpreted atomic term is considered a *variable* for the respective plugin. It may assign values to t' as for any constant and must handle assignments to t' by a different plugin.

Example 4. Consider the formula $\varphi = [f(x + y) + 2y \leq 0] \wedge [f(0) - y \approx 0]$ where x, y are of sort integer ($\mathbb{Z}$). The term structure of φ is depicted below. Rounded corners indicate sort $\mathbb{B}$, all other terms are of sort $\mathbb{Z}$. Colors indicate the responsible plugin for each term, where *green* and *orange* indicate the plugins of integer arithmetic $\mathcal{I}$ and uninterpreted functions $\mathcal{U}$, respectively. Boolean terms, which are handled by the MCSat reasoning procedure directly, are depicted black.

²This is formally proven in Lemma 5 of the appendix (Section A).



Consider the term $f(x + y)$ which is of sort $\mathbb{Z}$. It is a variable for $\mathcal{I}$, thus the plugin assigns and reads the term's (integer) value like any constant. The $\mathcal{U}$ plugin can do the same and ensures the congruence property of f when its sub-term $x + y$ has a value assigned. The same happens for the unary constraint terms (≤ 0 and ≈ 0) which are variables in MCSat's $\mathbb{B}$ reasoning. $\mathcal{I}$ can set their boolean value (by evaluating the constraints using their evaluation function) or the MCSat search can force them to be either true or false using Boolean reasoning, requiring $\mathcal{I}$ to set values accordingly.

Note that in this example constraints have been normalized as comparisons to 0 and defined as unary constraints. This is common in computer algebra and, of course, a possible alternative formalization in an MCSat procedure that is actually implemented in Yices2. However, for better readability, we are using binary comparison constraints in other examples.

Explanations. While different plugins interact with each other using the assigned terms on the trail, they need to translate theory-specific reasoning for those assignments into the common MCSat search procedure. This is done using a plugin-specific *explanation function*. Given a formula φ as the search problem, a trail M , and a (constraint) term p of sort $\mathbb{B}$, then lemma clause E is constructed by

$$E = (e_1, \dots, e_n, p) = \text{explain}(M, p)$$

fulfilling the following conditions: (i) E is a valid lemma in the search problem, i.e. for every interpretation I such that $\varphi^I = \text{true}$, there exists an $e \in E$ such that $e^I = \text{true}$.³ (ii) E justifies p , i.e. for every e_i with $1 \leq i \leq n$ we have $\nu_{\bar{M}}(e_i) = \text{false}$. (iii) E may introduce new literals, as long as they stem from a finite basis. The implementation of explain is fully dependent on the theory plugin that produces the lemma. In fact, admitting an explain function is all a background theory must support for integrating it in an MCSat search (c.f. [2]). Note that the explanation function might generate valid theory lemmas (independent of φ), thus ensuring for any interpretation I , there exists $e \in E$ such that $e^I = \text{true}$.

Example 5. Assume a search in real arithmetic with real terms x and y reaches a trail

$$M = \llbracket (xy \leq 0) \mapsto \text{true}, x \mapsto 1, y \mapsto 1 \rrbracket$$

which clearly leads to a not consistent $\bar{M}$. An explanation function $E = \text{explain}(M, \perp)$ is used to generate an explanation clause that translates this arithmetic reasoning to a clause:

$$\begin{aligned} E &= ((xy \leq 0) \wedge (x > 0) \wedge (y > 0) \implies \perp) \\ &= ((xy \leq 0) \wedge (x > 0) \implies (y \leq 0)) \\ &= (\neg(xy \leq 0) \vee \neg(x > 0) \vee (y \leq 0)) \end{aligned}$$

Clearly, this fulfills the requirements of an explanation function. It is a valid lemma (requirement (i)), as it is true for any assignment to x and y which can be seen by trivial algebraic reasoning. It justifies $\perp$ on the trail M (requirement (ii)), as $\neg(xy \leq 0)$ is on the trail in opposite polarity, and $\nu_{\bar{M}}(\neg(x > 0))$ as well as $\nu_{\bar{M}}(y \leq 0)$ are false. The finite basis condition (iii) is a property of the explanation procedure as a whole. The explanation introduced the new terms $x > 0$ and $y \leq 0$ and these terms must come from a finite set of terms the procedure can generate. For example, it must not generate arbitrary many terms like $\{x > 0, x > 1, x > 2, \dots\}$ as this would violate termination.

³Note that this is a weaker definition than presented in previous literature which requires a valid lemma in **any** interpretation, i.e. for any interpretation I , there exists an $e \in E$ such that $e^I = \text{true}$.

During the search the explain function is used in two cases: (i) Whenever theory-specific procedures detect that $\bar{M}$ is inconsistent, then $C = \text{explain}(M, \perp)$ is used to derive the *conflict clause* C . By the construction requirements of the explanation function, this excludes the current trail and forces the search to pivot. (ii) When propagating a term assignment, i.e. when a theory related procedure detects that $\text{feasible}(M, t) = \{\alpha\}$ for a yet unassigned term t , we use $J = \text{explain}(M, t = \alpha)$ to generate a *justification* J that explains the propagation with regard to the current trail M . By construction of J , we have that $(t = \alpha) \mapsto \text{true}$ on M and, thus, $(t \mapsto^* \alpha)$ is added to M . We denote $J = \text{just}(t \mapsto^* \alpha)$ and write $J = \text{just}(t)$ as a shorthand. It must be available whenever required by conflict resolution as long as the propagation is on the trail. This enables justifications to be calculated lazily, i.e. only when required by conflict analysis. This is important, as calls to the explain function might be costly, depending on the theory.

While MCSat performs all Boolean propagations, any propagation for other sorts is optional. Propagations are helpful for performance in general, constructing the required justification may be infeasible in many cases and a plugin is not required to perform a possible propagation. In case a propagation of term t to value α is detected, but no explanation can be constructed (with reasonable effort), it is, in general, still beneficial to add $t \mapsto \alpha$ to M as a decision (c.f. [9]).

3.2. Search Rules

As in the previous publications, we are describing the MCSat search as a system of symbolic proof rules. Given a set of clauses $\mathcal{C}$, we start with an empty trail $M = \llbracket \rrbracket_0$ and end in one of the two terminal states *sat* or *unsat*.

An active search can be in one of two states: *search state* or *conflict state*. While in a search state, M gets extended with either decisions or propagations until everything is assigned or $\bar{M}$ is detected to be inconsistent. In a conflict state, we aim to recover from an inconsistent $\bar{M}$ or detect the unsatisfiability of $\mathcal{C}$. We explain the conflict as a conflict clause C , resolve C with previous propagations and remove assignments from M until a relevant decision is reached. In the rules below, we denote a search state as a pair $\langle M, \mathcal{C} \rangle$ of trail M and a set of clauses $\mathcal{C}$. A state in conflict mode with conflict clause C is written as $\langle M, \mathcal{C} \rangle \vdash C$. We denote rules that take a conflict state as *conflict rules*, all other rules as *search rules*.

Running Example. Let $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5\}$ be an instance over propositional logic $\mathbb{B}$ and real arithmetic $\mathbb{R}$.

$$\begin{aligned} C_1 &= (x^2 \approx 4 \vee y \approx 8) & C_4 &= (x > 5 \vee x + y \approx 1) \\ C_2 &= (x^2 + 2y > y^2 \vee y \approx 1) & C_5 &= (x \approx 7y \vee x \not\approx y + 3) \\ C_3 &= (1 < -x \vee (x - 2)y \not\approx 3) \end{aligned}$$

Here x and y are uninterpreted constants of sort $\mathbb{R}$. In this running example we denote the number of applied rules in the superscript of the trail. We start the search in the state $\langle M_0^0, \mathcal{C} \rangle$. By definition, we have $M_0^0 = \llbracket \rrbracket_0$. For brevity we write p , p_n , and p^* on the trail instead of $p \mapsto \text{true}$, $p \mapsto_n \text{true}$, and $p \mapsto^* \text{true}$, respectively.

Decide

Assume a trail M_n and a term x such that $\nu_{\bar{M}}(x) = \text{undef}$, then any value $\alpha \in \text{feasible}(M, x)$ can be decided for x .

$$\langle M_n, \mathcal{C} \rangle \longrightarrow \langle \llbracket M, x \mapsto_{(n+1)} \alpha \rrbracket, \mathcal{C} \rangle$$

While a general term can be decided upon, in practice, deciding is limited to variables, i.e. terms that are atomic for a dedicated theory plugin. All non-atomic terms can be evaluated using the plugin's eval function and, thus, will get an assignment using the closure of the trail $\bar{M}$. Although more than one theory plugin can support $\text{sort}(t)$, we usually dedicate one plugin to make decisions for each particular sort. While this rule does not limit the choice for x and α , in any practical implementation, those choices are important considerations for the performance of the solver. Heuristics are essential to achieve reasonable performance in practice [9, 10].

Running Example. In the first search state $\langle M_0^0, C \rangle$, we decide $x \mapsto_1 2$ using *Decide*:

$$\langle \llbracket x \mapsto_1 2 \rrbracket_1^1, C \rangle$$

and denote the new trail M_1^1 . Note that in the new search state $\langle M_1^1, C \rangle$ we cannot decide on the propositional variable $(x - 2)y \not\approx 3$, because $\nu_{M_1^1}((x - 2)y \not\approx 3) = \text{true}$, but on $x^2 + 2y > y^2 \mapsto_2 \text{true}$ to get:

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2 \rrbracket_2^2, C \rangle$$

Propagate

Assume a clause of the form $C = \text{explain}(M, x = t) = (l_1 \wedge \dots \wedge l_\ell \implies x = t) = (\neg l_1 \vee \dots \vee \neg l_\ell \vee x = t)$ with a term t such that $\nu_{\bar{M}}(t) = \alpha \in \text{sort}(t)$ can be derived from the clause database. By construction, for all $1 \leq i \leq \ell$ we have that $\nu_{\bar{M}}(l_i) = \text{true}$, thus we can set α for x .

$$\langle M_n, C \rangle \longrightarrow \langle \llbracket M, x \mapsto^* \alpha \rrbracket_n, C \rangle$$

Note that not all possible propagations are actually required to be performed by a theory plugin. Even if a plugin realizes that there is only one feasible value for a term, it may decide not to propagate the value, because a justification is either too expensive or even impossible to calculate. In such cases, it is beneficial to force a decision of that variable soon (c.f. [9]). For completeness of the system, propagations are not required, as the system could also decide an assignment to the propagated value. For infinite domains, it might even be impossible to generate any justification as this would violate the finite basis requirement of the explain function (c.f. Example 5).

Running Example. We continue with M_2^2 as the current trail. The clause C_3 has only one literal that does not evaluate to false. So we use Boolean unit propagation to get the new state:

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2, ((x - 2)y \not\approx 3)^*_2 \rrbracket_2^3, C \rangle$$

This is possible because we take $\text{explain}(\langle M_2^2, C \rangle, (x - 2)y \not\approx 3) = C_3 = (1 < -x \vee (x - 2)y \not\approx 3) = (\neg(1 < -x) \rightarrow (x - 2)y \not\approx 3)$ and $\nu_{M_2^2}(\neg(1 < -x)) = \text{true}$, so we can propagate $(x - 2)y \not\approx 3 \mapsto^* \text{true}$ with justification C_3 . We can perform the similar propagation $x + y \approx 1 \mapsto^* \text{true}$ with $\text{just}(x + y \approx 1) = C_4 = (x > 5 \vee x + y \approx 1)$ and get:

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2, ((x - 2)y \not\approx 3)^*, (x + y \approx 1)^*_2 \rrbracket_2^4, C \rangle$$

Finally, we perform the arithmetic theory propagation $y \mapsto^* -1$ with justification $\text{explain}(\llbracket \dots \rrbracket, y \approx -1) = (x \not\approx 2 \vee \neg(x + y \approx 1) \vee y \approx -1)$ and get:

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2, ((x - 2)y \not\approx 3)^*, (x + y \approx 1)^*, y \mapsto^* -1 \rrbracket_2^5, C \rangle$$

Conflict

Given $\text{infeasible}(M, C)$, let $E = \text{explain}(M, \perp) = (e_1 \wedge \dots \wedge e_\ell \implies \perp) = (\neg e_1 \vee \dots \vee \neg e_\ell)$ such that for all $1 \leq i \leq \ell$ we have $\nu_{\bar{M}}(e_i) = \text{false}$, then the search switches to the conflict mode with E as its initial conflict clause:

$$\langle M_n, C \rangle \longrightarrow \langle M_n, C \rangle \vdash E$$

Running Example. We continue in the search state $\langle M_2^5, C \rangle$, with $M_2^5 = \llbracket x \mapsto_1 2, \dots, y \mapsto^* -1 \rrbracket_2$. Note that $C_5 = (x \approx 7y \vee x \not\approx y + 3) \in C$, so we get $\nu_{M_2^5}(C_5) = \text{false}$, which is a propositional conflict. Therefore, we use rule *Conflict* with $E = \text{explain}(\langle M_2^5, C \rangle, \perp) = C_5$ to get (with same trail $M_2^6 = M_2^5$):

$$\langle M_2^6, C \rangle \vdash C_5$$

Resolve

When in conflict mode, we can resolve a previous made propagation using its justification if the propagated term x occurs in the conflict clause C , i.e. $x \in \text{terms}(C)$:

$$\langle \llbracket M, x \mapsto^* \alpha \rrbracket_n, C \rangle \vdash C \longrightarrow \langle M_n, C \rangle \vdash R$$

Then by the construction of the propagation, the justification is of the form $\text{just}(x) = (l_1 \wedge \dots \wedge l_\ell \implies x \approx t)$ and the resolved conflict clause $R = C[x/t] \cup (\neg l_1 \vee \dots \vee \neg l_\ell)$.

Before continuing with the running example, let us first consider two other applications of **Resolve**:

Example 6. Assume $\mathbb{B}$ variables a, b , and c , the clause $D = (a \vee c)$, and a trail that assigns a to false. Then the propagation rule can use D for the propagation $\neg a \implies c$ (by simplifying $c \approx \text{true}$). Later, a conflict clause $C = (b \vee \neg c)$ becomes $R = (b \vee a)$, which is Boolean resolution.

Example 7. Let y and z be integer variables, $D = (y < 3 \vee z \approx 5)$ a clause, and a trail that assigns y to 6. Then the propagation rule can use D for the propagation $\neg(y < 3) \implies z \approx 5$. Later a conflict clause $C = (y \cdot z \approx 3 \vee \neg(z \approx 5))$ becomes $R = (y \cdot 5 \approx 3 \vee y < 3)$, which is arithmetic resolution.

Running Example. In the conflict state

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2, ((x - 2)y \not\approx 3)^*, (x + y \approx 1)^*, y \mapsto^* -1 \rrbracket_2^6, C \rangle \vdash C_5$$

we can apply the rule **Resolve**, because $y \in \text{terms}(C_5)$. Recall $\text{just}(y) = (x \not\approx 2 \vee \neg(x + y \approx 1) \vee y \approx -1)$ so the resolvent is $R = (x \approx 7y \vee x \not\approx y + 3)[y/-1] \cup (x \not\approx 2 \vee \neg(x + y \approx 1)) = (x \approx -7 \vee x \not\approx 2 \vee \neg(x + y \approx 1))$. This yields the new conflict state:

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2, ((x - 2)y \not\approx 3)^*, (x + y \approx 1)^* \rrbracket_2^7, C \rangle \vdash R$$

Because $(x + y \approx 1) \in R$, we can perform a similar resolution on $x + y \approx 1 \mapsto^* \text{true}$, which has the justification $C_4 = (x > 5 \vee x + y \approx 1)$. So $R' = R[x + y \approx 1/\top] \cup (x > 5) = (x \approx -7 \vee x \not\approx 2 \vee \neg(\top) \vee x > 5) = (x \approx -7 \vee x \not\approx 2 \vee x > 5)$. The new conflict state is:

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2, ((x - 2)y \not\approx 3)^* \rrbracket_2^8, C \rangle \vdash R'$$

Note that for the propagation $(x - 2)y \not\approx 3 \mapsto^* \text{true}$, with justification $(1 < -x \vee (x - 2)y \not\approx 3)$, the **Resolve** rule cannot be applied, because $((x - 2)y \not\approx 3) \notin \text{terms}(R')$.

Consume

In conflict mode, whenever a propagation is not part of the conflict clause, it may be omitted. Assume $x \notin \text{terms}(C)$, then

$$\langle \llbracket M, x \mapsto^* \alpha \rrbracket_n, C \rangle \vdash C \longrightarrow \langle M_n, C \rangle \vdash C$$

Running Example. In the current conflict state, we can use rule **Consume** to get rid of the last trail element and reach state

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2 \rrbracket_2^9, C \rangle \vdash R',$$

Drop

When a decision does not influence the current conflict clause C , then the decision can be dropped. Assuming $\nu_{\bar{M}_n}(C) = \text{false}$, any decision of level $n + 1$ can be dropped:

$$\langle \llbracket M_n, x \mapsto_{n+1} \alpha \rrbracket, C \rangle \vdash C \longrightarrow \langle M_n, C \rangle \vdash C$$

In practice, one can calculate the decision level n of C and drop all decisions and corresponding propagations with level greater than n at once.

Running Example. In the conflict state

$$\langle \llbracket x \mapsto_1 2, (x^2 + 2y > y^2)_2 \rrbracket_2^9, \mathcal{C} \rangle \vdash (x \approx -7 \vee x \not\approx 2 \vee x > 5),$$

the rule **Drop** is applied because $\nu_{\bar{M}'}(x \approx -7 \vee x \not\approx 2 \vee x > 5) = \text{false}$ with $M' = \llbracket x \mapsto 2 \rrbracket$:

$$\langle \llbracket x \mapsto_1 2 \rrbracket_1^{10}, \mathcal{C} \rangle \vdash (x \approx -7 \vee x \not\approx 2 \vee x > 5)$$

We cannot apply the **Drop** further as $\nu_{\bar{\emptyset}}(x \approx -7 \vee x \not\approx 2 \vee x > 5) = \text{undef}$.

Backjump

When a decision on the trail influences the conflict clause C , then undoing the decision changes $\nu_{\bar{M}_n}(C)$ to undef. Then there exists a subset of literals $C' \subseteq C$ such that $\nu_{\bar{M}_n}(L) = \text{undef}$ for all $L \in C'$. In case $C' = \{L\}$ then L can be propagated by C . Thus a propagation is put on the trail with $\text{just}(L) = C$:

$$\langle \llbracket M_n, x \mapsto_{n+1} \alpha \rrbracket, \mathcal{C} \rangle \vdash C \longrightarrow \langle \llbracket M_n, L \mapsto^* \text{true} \rrbracket, \mathcal{C} \rangle$$

Otherwise, we need to perform a Boolean decision and pick a choice $L \in C'$ to satisfy C :

$$\langle \llbracket M_n, x \mapsto_{n+1} \alpha \rrbracket, \mathcal{C} \rangle \vdash C \longrightarrow \langle \llbracket M_n, L \mapsto_{n+1} \text{true} \rrbracket, \mathcal{C} \rangle$$

When choosing L we need to make sure that $x \in \text{terms}(L)$, i.e. $x \neq L$.

The last condition for the second case is important to ensure termination. In case x is a literal as well as a sub-term of another literal (for example if x is a propositional term and $C = \{x, f(x)\}$), we need to ensure not to replace a decision on x with another decision on L (indefinitely).

Running Example. We are in the conflict state:

$$\langle \llbracket x \mapsto_1 2 \rrbracket_1^{10}, \mathcal{C} \rangle \vdash (x \approx -7 \vee x \not\approx 2 \vee x > 5)$$

Here the decision $x \mapsto_1 2$ affects the evaluation of the conflict clause R' because $\nu_{\bar{\emptyset}}(R') = \text{undef}$. Before leaving the conflict state, we apply the **Learn** rule (see below) in order to add the final resolvent R' to the clause set. Then we can apply the **Backjump** to leave the conflict state. Since multiple literals of R' evaluate to undef, we replace the decision on the trail with one of the following decisions in $R'_u = R' = \{(x \approx -7)_1, (x \not\approx 2)_1, (x > 5)_1\}$ and pick $x \approx -7 \mapsto_1 \text{true}$. Then the new state is

$$\langle \llbracket (x \approx -7)_1 \rrbracket_1^{11}, \mathcal{C} \cup \{R'\} \rangle$$

Sat

Assume a trail M such that $\text{satisfies}(M, \mathcal{C})$, then

$$\langle M, \mathcal{C} \rangle \longrightarrow \text{sat}$$

Note that this is also the case when for each term t , we have that $\nu_{\bar{M}}(t) \neq \text{undef}$ and not infeasible(M). In the sat case, the model can be constructed by taking the closure of the last trail and assign all still unassigned terms with arbitrary consistent values.

Running Example. We propagate $x \mapsto^* -7, x + y \approx 1 \mapsto^* \text{true}$ and $y \mapsto^* 8$ to get:

$$\langle \llbracket (x \approx -7)_1, x \mapsto^* -7, (x + y \approx 1)^*, y \mapsto^* 8 \rrbracket_1^{14}, \mathcal{C} \cup \{R'\} \rangle$$

The Sat rule can be applied here because $\text{satisfies}(M_1^{14}, \mathcal{C} \cup \{R'\})$ holds.

Unsat

Assume a trail M and a conflict at decision level 0, then

$$\langle M_0, \mathcal{C} \rangle \vdash C \longrightarrow \text{unsat}$$

3.3. Learning and Forgetting

The rules presented above are sufficient to perform an MCSat search. For practical performance reasons it is, however, beneficial to add, i.e. learn, clauses to the clause database $\mathcal{C}$ as well as delete, i.e. forget, previously learned clauses. Note that any conflict clause C is logically implied by the clause database $\mathcal{C}$. This is because any explanation clause is implied by $\mathcal{C}$ by requirement (i) of explain and the resolvent of two implied clauses with rule **Resolve** keeps this property.⁴ Adding or removing an implied clause to a set of clauses does not change the models.

There are two opportunities to learn clause: (i) any clause during conflict analysis, i.e. the original conflict clause or any resolvent thereof, as well as (ii) any justification that was generated during a propagation. In practice, the most important clause to learn tends to be the last conflict clause right before exiting conflict analysis with rule **Backjump** as this clause is used for pivoting the search.

Learn

Assume a search state in conflict with clause C , then C can be added to the clause database $\mathcal{C}$ if $C \notin \mathcal{C}$:

$$\langle M, \mathcal{C} \rangle \vdash C \longrightarrow \langle M, \mathcal{C} \cup \{C\} \rangle \vdash C$$

Learn-Justification

Assume the search propagated $x \mapsto^* \alpha$, then $J = \text{just}(x \mapsto^* \alpha)$ can be added to the clause database $\mathcal{C}$ if $J \notin \mathcal{C}$:

$$\langle \llbracket M, x \mapsto^* \alpha \rrbracket, \mathcal{C} \rangle \longrightarrow \langle \llbracket M, x \mapsto^* \alpha \rrbracket, \mathcal{C} \cup \{J\} \rangle$$

If, for practical performance considerations, the justification J is calculated lazily, i.e. only when required by conflict analysis, it can still be learned upon calculation. Since, in the proof system, we assume that justifications are available right away, we limit applications of **Learn-Justification** to be used directly after **Propagate** for technicalities the termination proof.

Forget

Assume clause $C \in \mathcal{C}$ is a learned clause, then we may forget C :

$$\langle M, \mathcal{C} \rangle \longrightarrow \langle M, \mathcal{C} \setminus \{C\} \rangle$$

The **Forget** rule is optional but important for practical performance. As for any choice in the search, heuristics can greatly help in deciding which clauses to forget.

It is important to note that a justification $J = \text{just}(x)$ of a propagation $x \mapsto^* \alpha$ on the trail can not be forgotten, as $\text{just}(x)$ is a property of the trail element and not an element of $\mathcal{C}$. If J had previously been learned using **Learn-Justification**, it can be removed from $\mathcal{C}$, however J is available as $\text{just}(x)$ as long as $x \mapsto^* \alpha$ is remains on the trail.

⁴This is formally proven in Lemma 2 of the appendix (Section A).

3.4. Soundness, Completeness, and Termination

The main theorem of [2] is the soundness and completeness of the calculus. For our adapted set of rules, we provide the same result and give the proof in the appendix (Section A).

Theorem 1. *Given a set of clauses $\mathcal{C}$, and assuming an explanation function explain , any derivation starting from the initial state $\langle \llbracket \rrbracket, \mathcal{C} \rangle$ will terminate either in a state sat , when $\mathcal{C}$ is satisfiable (soundness), or in the state unsat , when $\mathcal{C}$ is unsatisfiable (completeness).*

4. MCSat Theories

In Section 3.1 we have outlined the requirements of a theory reasoning plugin and a corresponding explanation function. Based on these definitions we will provide some examples for specific theories.

Propositional Logic. Although not an actual theory plugin, we can define an explanation function for a Boolean conflict. Recall that the Boolean sort $\mathbb{B} = \{\text{true}, \text{false}\}$ together with the Boolean connectives translated in CNF are natively handled by MCSat. Given a clause database $\mathcal{C}$ and a trail M , a conflict is a clause $C \in \mathcal{C}$ that evaluates to false under M . The explanation function $\text{explain}(M, \perp)$ simply returns the clause C . C trivially justifies $\perp$ and does not introduce new literals, thus satisfying property (ii) and (iii) of an explanation function. Although C is not a valid lemma in general, it is (trivially) implied by the clause database $\mathcal{C}$ and thus it is a lemma within the search problem.

The same works for propagation. The propagate rule can be used for any clause $C \in \mathcal{C}$ that is unit with regard to M . The explanation is simply C .

Arithmetic. Non-linear real arithmetic has been the first theory to be applied to MCSat in its original NLSAT publication [1]. It is based on cylindrical algebraic decomposition (CAD) which is a well known technique from computer algebra. The theory can solve polynomial constraints with integer coefficients and defines the sort of reals $\mathbb{R}$ containing all algebraic numbers, i.e. the subset of real numbers that can be the root of a polynomial with integer coefficients. Note that $\mathbb{R}$ is of infinite size. To gain a finite basis explain function, the theory plugin uses CAD to describe an n -dimensional area in which all polynomials of the conflict are sign-invariant. As all possible decompositions of the (finitely many) input polynomials contain only finitely many such areas, there are only finitely many terms to describe those areas. For further details on the implementation of this theory we refer to [1] as well as to the SMT solver SMT-RAT [6] which contains one of the most advanced implementations of this MCSat theory. In addition to reals, there haven been MCSat theories for linear integers [11] as well as non-linear integers [12].

Uninterpreted Functions. The theory of uninterpreted functions (UF) does not contain its own sort. It rather tracks the assignments of its terms by other theories plugins and detects conflicts and possible propagations using equality graphs (e-graphs). Therefore, each generated explanation E given by explain only contains existing literals, trivially fulfilling property (iii) of an explanation function. E contains terms from the trail that are equal due to the e-graph, by the correctness of the e-graph, this satisfies (i). The third property (ii) is fulfilled by the construction of E , as only assigned terms are considered. Based on the theory UF, an MCSat implementation of the theory of arrays has been implemented [13].

Further theories. Further interesting theories that have been implemented in an MCSat search are the theory of fixed-size bit-vectors (BV) [14] and the theory of finite fields (FF) [15].

5. Conclusion

Implementation Notes. The general idea of this adapted version of the MCSat calculus was based on its implementation within Yices2 [3]. The software architecture of the MCSat engine includes a general MCSat core independent of any theory. Each theory (including the theory of propositional logic) is implemented as a plugin that provides, using a well-defined interface, functionality for decisions, propagations as well as conflict detection and explanation. Each plugin can register terms and types that it is capable of handling. This lead us to defining the transition systems on term assignments instead of a strict separation in theory and propositional variables. The core MCSat engine queries all plugins for all available propagations before proceeding with the next decision. It is up to the individual plugin to choose which propagations to perform. See [3, 9] for a description of the general MCSat core algorithm in Yices2.

While the MCSat proof system requires an explanation function to be finite basis for proving termination, the implementation in Yices2 departs from this requirement: by letting the arithmetic variable order for decisions change dynamically using the Variable State Independent Decaying Sum (VSIDS) heuristic, explanations generated by CAD the finite basis cannot be guaranteed. Although, this is a bad choice for termination, VSIDS improves performance for practical instances significantly [9].

Calculus Updates. Notable changes to the MCSat proof calculus as presented in this work compared to the original publications [1, 2, 3] are:

- *Term assignments instead of theory variables.* Previous publications defined two distinct kinds of variables: propositional literals and theory variables. The former were any constraints over a term in a theory, the latter an uninterpreted constant in a particular theory. An (overloaded) value function evaluates literals according to assignment to theory variables. We changed this to better match the definitions of functions and sorted terms in SMT-LIB and allow an arbitrary assignments to terms. This enables us to present theory combination in MCSat closer to implementation, especially in the presence of uninterpreted functions (see Example 4).
- *Combining Boolean and theory reasoning.* Due to the clear distinction between theory variables and propositional literals, previous publications maintained two sets of proof rules to cater to both cases. This led to similar rules with minor differences, depending on whether they are handling theory terms or the propositional structure. In this work, we implemented Boolean reasoning closer to theory reasoning to allow a single set of rules to handle both cases (see, for example, rule [Propagate](#) or rule [Decide](#)). While Boolean reasoning is still a special case due to additional rules for conflict analysis, this clarifies the meaning of the rules and avoids minor differences in similar rules.
- *Common framework for propagation.* The original publications of MCSat did not provide a way to propagate theory values as depicted in rule [Propagate](#). This form of propagation was introduced as a side note in a later publication [?]. In this work, the [Propagate](#) and [Resolve](#) rules have been adapted to support both theory and Boolean propagation. By adding the [Learn-Justification](#) rule, learning a potentially expensive justification for a theory-based propagation has been enabled, which was not possible by [Learn](#).
- *Updated explain function.* The original publications require the explain function to return a valid lemma E . A valid lemma is a clause that is true under any interpretation, i.e. $\models E$. We relax this requirement to E to be true under any model of the search problem φ , i.e., $\varphi \models E$. This is a strictly weaker definition, as any valid explanation still fulfills the requirement but allows modeling Boolean conflicts as regular explanations. Furthermore, this change gives more flexibility to the implementation of the explain function for particular theories, for example, allowing it to return an empty clause if the theory's reasoning procedure detects unsatisfiability of φ .

Conclusion. In this work we have presented an updated version of the original formal description of the MCSat calculus. In contrast to the original version we describe the calculus on arbitrary term assignments and do not distinguish between propositional and theory rules. We presented all relevant concepts, showed the calculus’ rules as well as provided examples.

Acknowledgments

The authors thank Ahmed Irfan, Stéphane Graham-Lengrand, and Jasper Nalbach for discussion and feedback. We further thank the anonymous reviewers for exceptional feedback. This research was funded in whole or in part by the ERC Consolidator Grant ARTIST 101002685; the WWTF ICT22-007 grant ForSmart; the FWF 10.55776/ESP666 grant, and by the SBA Research COMET Center SBA-K1 NGC managed by the FFG.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] D. Jovanovic, L. M. de Moura, Solving non-linear arithmetic, in: B. Gramlich, D. Miller, U. Sattler (Eds.), Intl. Joint Conf. on Automated Reasoning (IJCAR), volume 7364 of *LNCS*, Springer, 2012, pp. 339–354. URL: https://doi.org/10.1007/978-3-642-31365-3_27. doi:10.1007/978-3-642-31365-3_27.
- [2] L. de Moura, D. Jovanovic, A model-constructing satisfiability calculus, in: R. Giacobazzi, J. Berdine, I. Mastroeni (Eds.), Intl. Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI), volume 7737 of *LNCS*, Springer, 2013, pp. 1–12. URL: https://doi.org/10.1007/978-3-642-35873-9_1. doi:10.1007/978-3-642-35873-9_1.
- [3] D. Jovanovic, C. Barrett, L. de Moura, The design and implementation of the model constructing satisfiability calculus, in: Intl. Conf on Formal Methods in Computer-Aided Design (FMCAD), IEEE, 2013, pp. 173–180. doi:10.1109/FMCAD.2013.7027033.
- [4] K. Korovin, N. Tsiskaridze, A. Voronkov, Conflict resolution, in: Springer (Ed.), Principles and Practice of Constraint Programming, volume 5732 of *Lecture Notes in Computer Science*, Springer Nature, 2009, pp. 509–523.
- [5] B. Dutertre, Yices 2.2, in: CAV, volume 8559 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 737–744.
- [6] F. Corzilius, G. Kremer, S. Junges, S. Schupp, E. Ábrahám, SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving, in: SAT, volume 9340 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 360–368.
- [7] L. M. de Moura, N. S. Bjørner, Z3: an efficient SMT solver, in: TACAS, volume 4963 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 337–340.
- [8] C. Barrett, P. Fontaine, C. Tinelli, SMT-LIB Standard (Version 2.6), <https://smt-lib.org/>, 2021.
- [9] T. Hader, A. Irfan, S. Graham-Lengrand, Decision heuristics in mcsat, in: International Conference on Computer Aided Verification, Springer, 2025, pp. 42–56.
- [10] E. Lipparini, Boosting mcsat modulo nonlinear integer arithmetic via local search, in: Automated Deduction–CADE 30: 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28–31, 2025, Proceedings, Springer Nature, 2025, p. 95.
- [11] D. Jovanovic, L. M. de Moura, Cutting to the chase solving linear integer arithmetic, in: N. S. Bjørner, V. Sofronie-Stokkermans (Eds.), Intl. Conf on Automated Deduction (CADE), volume 6803 of *LNCS*, Springer, 2011, pp. 338–353. URL: https://doi.org/10.1007/978-3-642-22438-6_26. doi:10.1007/978-3-642-22438-6_26.
- [12] D. Jovanovic, Solving nonlinear integer arithmetic with MCSAT, in: VMCAI, volume 10145 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 330–346.
- [13] A. Irfan, S. Graham-Lengrand, Arrays reasoning in MCSat, in: SMT@CAV, volume 3725 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 24–35.
- [14] S. Graham-Lengrand, D. Jovanovic, B. Dutertre, Solving bitvectors with MCSAT: explanations from bits and pieces, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), Intl. Joint Conf. on Automated Reasoning (IJCAR), Part I, volume 12166 of *LNCS*, Springer, 2020, pp. 103–121. URL: https://doi.org/10.1007/978-3-030-51074-9_7. doi:10.1007/978-3-030-51074-9_7.
- [15] T. Hader, D. Kaufmann, A. Irfan, S. Graham-Lengrand, L. Kovács, MCSat-based finite field reasoning in the Yices2 SMT solver (short paper), in: IJCAR (1), volume 14739 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 386–395.

A. Proofs

We adapt the soundness and correctness proof [2] (Theorem 1) to suite our definitions. While the proof is more detailed and structured differently, the general ideas remains the same.

Lemma 1. *Given a set of clauses $\mathcal{C}$, in any derivation of the transition system starting from the initial state $\langle \square, \mathcal{C} \rangle$, any terminating derivation of the transition system ends in sat or unsat.*

Proof. To show that the derivation cannot be stuck in a state outside sat and unsat, we show that, given an arbitrary state $S \notin \{\text{sat}, \text{unsat}\}$, we can apply a rule to reach a different state. We distinguish two cases on S :

1. $S = \langle M, \mathcal{C} \rangle$, i.e. S is not in conflict. If there exists a term t such that $\text{feasible}(M, t) \neq \emptyset$ we apply rule **Decide** to reach state S' . If all terms are assigned and $\text{satisfies}(M, \mathcal{C})$ holds, then we use rule **Sat** to reach state $S' = \text{sat}$. Otherwise, if there are only unassigned terms with $\text{feasible}(M, t) = \emptyset$, then clearly $\text{infeasible}(M, \mathcal{C})$ and we use rule **Conflict** to reach a conflict state S' . In any case we have $S \neq S'$.
2. $S = \langle M, \mathcal{C} \rangle \vdash C$, i.e. S is in conflict with clause C . Then we can apply rules **Resolve**, **Consume**, and **Drop** exhaustively until we either reach a state S' such that either rule **Backjump** can be used to reach a state $\langle M, \mathcal{C} \rangle$ not in conflict or rule **Unsat** to reach state unsat. In both cases the new state is different from S .

Since this clearly holds for $S = \langle \square, \mathcal{C} \rangle$, the desired property holds for any reachable state by induction. $\square$

Lemma 2. *Given a set of clauses $\mathcal{C}$, in any derivation of the transition system starting from the initial state $\langle \square, \mathcal{C} \rangle$, all $\vdash C$ clauses are implied by $\mathcal{C}$.*

Proof. Let the implication of a clause C by a set of clauses $\mathcal{C}$ be denoted as $\mathcal{C} \models C$. Given a state S such that it is either of form $\langle M, \mathcal{C} \rangle$ or $\langle M, \mathcal{C} \rangle \vdash C$ where $\mathcal{C} \models C$. We show that applying an arbitrary rule $\mathcal{R}$ to reach state S' , the property holds for S' as well. We split on $\mathcal{R}$:

- Rule **Conflict**: Given $S = \langle M, \mathcal{C} \rangle$ to apply **Conflict**, we have that $C = \text{explain}(M, \perp)$. By the requirements of **explain**, $\mathcal{C} \models C$ or a valid lemma, thus $S' = \langle M, \mathcal{C} \rangle \vdash C$ fulfills the property.
- Rule **Resolve**: Let $S = \langle \llbracket M, x \mapsto^* \alpha \rrbracket, \mathcal{C} \rangle \vdash C$ and $J = \text{just}(x)$. Then J was created using **explain** on a previous application of the rule **Propagate**, thus $\mathcal{C} \models J$ and $J = (\neg l_1 \vee \dots \vee \neg l_\ell \vee x = t)$. By assumption, we have $\mathcal{C} \models C$. We need to show that $\mathcal{C} \models C'$ for $C' = C[x/t] \cup (\neg l_1 \vee \dots \vee \neg l_\ell)$. From $\mathcal{C} \models J$ we have that each model I of $\mathcal{C}$ either $l_i^I = \text{false}$ or $x^I = t^I$. In the first case $C'^I = \text{true}$ as $\neg l_i \in C'$, in the second case, since $x^I = t^I$ together with $C^I = \text{true}$, $C[x/t]^I = \text{true}$ and, thus, $C'^I = \text{true}$.
- Any other rule does not generate a new clause $\vdash C$ clause.

Since the property holds for the initial state $\langle \square, \mathcal{C} \rangle$ trivially, the property holds for any reachable state in the derivation by induction. $\square$

Lemma 3. *Given a set of clauses $\mathcal{C}$, in any derivation of the transition system starting from the initial state $\langle \square, \mathcal{C} \rangle$, we have that $\nu_{\bar{M}}(C) = \text{false}$ for the $\vdash C$ clauses.*

Proof. We perform the same induction as for Lemma 2 and split on the applied rule $\mathcal{R}$:

- Rule **Conflict**: By construction of $C = \text{explain}(M, \perp)$ in the resulting state $S' = \langle M, \mathcal{C} \rangle \vdash C$ we have that $\nu_{\bar{M}}(C) = \text{false}$.

- Rule **Resolve**: Let $S = \langle \llbracket M, x \mapsto^* \alpha \rrbracket, C \rangle \vdash C$ and $J = \text{just}(x)$. By the induction hypothesis we have $\nu_{\bar{M}}(C) = \text{false}$. Since J was created using explain on a previous application of the rule **Propagate**, $J = (\neg l_1 \vee \dots \vee \neg l_\ell \vee x = t)$ and $\nu_{\bar{M}}(l_i) = \text{false}$ for $1 \leq i \leq \ell$, thus $\nu_{\bar{M}}(x) = \nu_{\bar{M}}(t)$. Therefore, $\nu_{\bar{M}}(C) = \nu_{\bar{M}}(C[x/t])$ and, therefore, $\nu_{\bar{M}}(C') = \text{false}$.
- Rule **Consume**: As the propagated term x is not contained in C , we have $\nu_{\bar{M}}(C) = \text{false}$ trivially.
- Rule **Drop**: The condition holds for the updated trail trivially as the desired property is a prerequisite of the rule.
- Any other rule does not generate a state with a $\vdash C$ clause.

Since the property holds for the initial state $\langle \llbracket \rrbracket, C \rangle$ trivially, the property holds for any reachable state in the derivation by induction. $\square$

Lemma 4 (Congruence). *Given a set of clauses $\mathcal{C}$, in any derivation of the transition system starting from the initial state $\langle \llbracket \rrbracket, C \rangle$, for every reachable state with trail M , every uninterpreted function symbol f , and every pair of $f(t_1, \dots, t_n), f(t'_1, \dots, t'_n)$ with $\nu_{\bar{M}}(f(t_1, \dots, t_n)), \nu_{\bar{M}}(f(t'_1, \dots, t'_n)) \neq \text{undef}$, we have*

$$(\forall i. \nu_{\bar{M}}(t_i) = \nu_{\bar{M}}(t'_i)) \implies \nu_{\bar{M}}(f(t_1, \dots, t_n)) = \nu_{\bar{M}}(f(t'_1, \dots, t'_n)).$$

Proof. By induction on the derivation. The base case $\langle \llbracket \rrbracket, C \rangle$ holds automatically. Only **Decide**, **Propagate**, and the decision case of **Backjump** assign a new value to a term; in each case the assigned term is handled by its owning plugin, which we require to respect congruence. All other rules only remove trail elements or leave M unchanged, so the property is inherited from the induction hypothesis. $\square$

Lemma 5 (Model Existence). *Given a set of clauses $\mathcal{C}$ and a term assignment S such that satisfies($S, \mathcal{C}$) holds, $\mathcal{C}$ is satisfiable.*

Proof. Define interpretation I to agree with $\bar{S}$ on terms($\mathcal{C}$): for interpreted symbols this is fixed by the evaluation function, and for uninterpreted f , set $f^I(\nu_{\bar{S}}(t_1), \dots, \nu_{\bar{S}}(t_n)) = \nu_{\bar{S}}(f(t_1, \dots, t_n))$ for each $f(t_1, \dots, t_n) \in \text{terms}(\mathcal{C})$, extending arbitrarily elsewhere. This is well-defined by Lemma 4, and by consistent($\bar{S}$) we get $t^I = \nu_{\bar{S}}(t)$ for every $t \in \text{terms}(\mathcal{C})$ by induction on term structure. Since satisfies($S, \mathcal{C}$) gives $\nu_{\bar{S}}(C) = \text{true}$ for every $C \in \mathcal{C}$, we obtain $C^I = \text{true}$ for every $C \in \mathcal{C}$, so I is a model of $\mathcal{C}$. $\square$

Lemma 6 (Termination). *Any derivation of the transition system starting from the initial state $\langle \llbracket \rrbracket, C \rangle$ terminates.*

Proof. It is obvious that the search leaves the conflict mode after a finite number of rule applications towards the search mode or the state unsat. This is because each rule in conflict mode removes an element from the trail. We define a big-step transition relation $\longrightarrow_{bs}$ that covers a transition from a search state, applying a finite number of conflict rules to the next search state.

Since explain returns new terms from a finite basis, we can assume a finite set of terms T that contains all possible terms. In order to show progress of the search, we define a partial order $M_1 \prec M_2$ on trails. A trail contains two different kinds of elements: propagations ($x \mapsto^* \alpha$) and decision ($x \mapsto \alpha$). We consider propagations to be heavier than any decision and define the weight of a decision by the size of the term. We define the size of a term t by $|\text{terms}(t)|$ and the maximum term size $\Omega = \max_{t \in T} |\text{terms}(t)|$. We capture this in a weight function w as follows:

$$\begin{aligned} w(x \mapsto^* \alpha) &= \Omega + 1 \\ w(x \mapsto \alpha) &= |\text{terms}(x)| \end{aligned}$$

We define $M_1 \prec M_2$ using a lexicographical partial order based on the weights of the trail elements by

$$\llbracket \rrbracket \prec M \quad \text{if } M \neq \llbracket \rrbracket$$

$$\llbracket a, M_1 \rrbracket \prec \llbracket b, M_2 \rrbracket \quad \text{if } w(a) < w(b) \text{ or } (w(a) = w(b) \wedge M_1 \prec M_2)$$

Clearly, $\llbracket \cdot \rrbracket$ is the minimal element and for all trails M and $N \neq \llbracket \cdot \rrbracket$ we have $M \prec \llbracket M, N \rrbracket$ and since $|T|$ is finite, there is a maximal element. Any rule that adds a new element to the trail is creating a bigger trail with regard to $\prec$. This already covers most rules. Now let us consider the big-step $\rightarrow_{bs}$ transition from a state $\langle M_1, C_1 \rangle$ to $\langle M_2, C_2 \rangle$. Since the only rule to exit the conflict state is **Backjump**, any big step $\rightarrow_{bs}$ will end in exactly one application of that rule. Note that no rule in conflict analysis adds an element to the trail, with the sole exception of **Backjump** that replaces one trail element. Thus, M_1 and M_2 share a common prefix followed by the one trail element $(x \mapsto \alpha) \in M_1$ that was replaced by $(x' \mapsto \alpha') \in M_2$. There are two cases in the application of **Backjump**. In the first case the decision $x \mapsto \alpha$ is replaced with a propagation $x' \mapsto^* \alpha'$, thus $M_1 \prec M_2$ as $w(x \mapsto \alpha) < w(x' \mapsto^* \alpha')$. In the second case, since there are multiple literals of C which are evaluated to undef by unassigning x , there is at least one such literal $x \neq x'$ such that $x \in \text{terms}(x')$ and thus $w(x \mapsto \alpha) < w(x' \mapsto \alpha')$. Therefore, we have that $M_1 \prec M_2$.

Regarding clause learning and forgetting: Observe that any application of the rule **Learn** is part of one big step $\rightarrow_{bs}$ application. Any application of **Learn-Justification** can only happen immediately after an application of **Propagate**, thus any trail after the application of the two rules (as a big-step) will be bigger than before as the propagation has been added. To justify the **Forget** rule we define another partial order $\langle M_1, C_1 \rangle \ll \langle M_2, C_2 \rangle$ by

$$M_1 \prec M_2 \text{ or } (M_1 = M_2 \text{ and } |C_1| > |C_2|)$$

With this definition we have that removing a clause generates a bigger state with regard to $\ll$. A maximal element $\langle M_{\max}, C_0 \rangle$ of $\ll$ is a maximal element of $\prec$ and C_0 , the initial set of clauses. Since all rules are producing bigger states and we cannot increase forever, the termination of the system follow. $\square$

With the above lemmas, we can finally proof Theorem 1.

Proof of Theorem 1. Assume a set of clauses $\mathcal{C}$ and an explanation function explain . Starting from the initial state $\langle \llbracket \cdot \rrbracket, \mathcal{C} \rangle$, by Lemma 1 and Lemma 6 we have that all derivations terminate in either sat or unsat. In case the derivation terminates in sat, the last applied rule was **Sat**, i.e., $\text{satisfies}(M, \mathcal{C})$ holds; by Lemma 5, $\mathcal{C}$ is satisfiable. Thus we have that the original problem is indeed satisfied.

Otherwise, if the derivation terminates in unsat, then rule **Unsat** was applied, and the previous state was $\langle M_0, \mathcal{C} \rangle \vdash C$ where M_0 contains propagations only. By Lemma 3, $\nu_{\bar{M}}(C) = \text{false}$. Resolving C with all remaining propagations on M_0 to C' , we have $\nu_{\emptyset}(C') = \text{false}$. Then by Lemma 2, $\mathcal{C}$ implies C . Thus, there is no interpretation that satisfies $\mathcal{C}$ and the search problem is unsatisfiable. $\square$

B. Examples

This section provides further examples of different theories including theory combination using the conversion functions between different sorts.

We denote the sort of m -bit vectors as $\mathcal{S}_{BV}^{[m]}$, with the corresponding domain $\text{dom}(\mathcal{S}_{BV}^{[m]}) = \{0, 1\}^m$, i.e. the set of all bit strings of length m . Sorts for different sizes m are disjoint, but they share a common set of function symbols: (i) Constants for each bit-vector of fixed width, e.g., `#b1011` for $m = 4$, (ii) Bitwise operators such as `and`, `or`, `xor`, and `not`, (iii) arithmetic operators, (iv) as well as comparison operators, as usual, and (v) conversion operators such as `bv2int`, `bv2nat`, and `nat2bv`.

For a prime power $q = p^n$ with p prime and $n \geq 1$, we denote the finite field with q elements as $\mathbb{F}_q$. For $n = 1$, $\mathbb{F}_q$ corresponds to the integers modulo p ; for $n > 1$, the field is constructed as a quotient ring of polynomials over $\mathbb{F}_p$ modulo an irreducible polynomial of degree n . To indicate that a constant $x \in \mathbb{F}_p$ we write x_p .

Additional Example 1 Assume we have the following instance $\mathcal{C}_\alpha = \{C_1, C_2, C_3, C_4\}$:

$$\begin{aligned} C_1 &= (g(x) \approx x) \\ C_2 &= (x - 50 \approx 3) \\ C_3 &= (z >_u \#b10 \vee f(g(g(x))) \approx 16) \\ C_4 &= (g(f(53)) \not\approx g(16) \vee y_7^3 \approx 27) \end{aligned}$$

Here, $x \in \mathbb{R}$, y from $\mathbb{F}_7$, and z from $\mathcal{S}_{BV}^2$ are uninterpreted constants. Furthermore, $f : \mathbb{R} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$ are uninterpreted functions. In our search, we obtain the following search states by applying the rules of the calculus:

1. $\langle \llbracket \rrbracket, \mathcal{C}_\alpha \rangle$
 $\downarrow$ Propagate with $\text{just}(g(x) \approx x) = (g(x) \approx x)$
2. $\langle \llbracket (g(x) \approx x)^* \rrbracket, \mathcal{C}_\alpha \rangle$
 $\downarrow$ Propagate with $\text{just}(x - 50 \approx 3) = (x - 50 \approx 3)$
3. $\langle \llbracket (g(x) \approx x)^*, (x - 50 \approx 3)^* \rrbracket, \mathcal{C}_\alpha \rangle$
 $\downarrow$ Propagate with $\text{just}(x \mapsto^* 53) = (x - 50 \approx 3)$
4. $\langle \llbracket (g(x) \approx x)^*, (x - 50 \approx 3)^*, x \mapsto^* 53 \rrbracket, \mathcal{C}_\alpha \rangle$
 $\downarrow$ Decide
5. $\langle \llbracket (g(x) \approx x)^*, (x - 50 \approx 3)^*, x \mapsto^* 53, z \mapsto_1 \#b01 \rrbracket, \mathcal{C}_\alpha \rangle$
 We introduce the trail $M_1 = \llbracket (g(x) \approx x)^*, (x - 50 \approx 3)^*, x \mapsto^* 53, z \mapsto_1 \#b01 \rrbracket$.
 $\downarrow$ Propagate with $\text{just}(f(g(g(x))) \mapsto^* 16) = (z >_u \#b10 \vee f(g(g(x))) \approx 16)$
6. $\langle \llbracket M_1, f(g(g(x))) \mapsto^* 16 \rrbracket, \mathcal{C}_\alpha \rangle$
 $\downarrow$ Propagate with $\text{just}(g(f(53)) \not\approx g(16) \mapsto^* \text{false}) =$
 $(\neg(g(x) \approx x) \vee \neg(x \approx 53) \vee \neg(f(g(g(x))) \approx 16) \vee \neg(g(f(53)) \not\approx g(16)))$
7. $\langle \llbracket M_1, f(g(g(x))) \mapsto^* 16, \neg(g(f(53)) \not\approx g(16))^* \rrbracket, \mathcal{C}_\alpha \rangle$
 $\downarrow$ Propagate with $\text{just}(y_7^3 \approx 27 \mapsto^* \text{true}) = (g(f(53)) \not\approx g(16) \vee y_7^3 \approx 27)$
8. $\langle \llbracket M_1, f(g(g(x))) \mapsto^* 16, \neg(g(f(53)) \not\approx g(16))^*, (y_7^3 \approx 27)^* \rrbracket, \mathcal{C}_\alpha \rangle$
 $\downarrow$ Conflict from $\mathcal{T}_{FF}$
9. $\langle \llbracket M_1, f(g(g(x))) \mapsto^* 16, \neg(g(f(53)) \not\approx g(16))^*, (y_7^3 \approx 27)^* \rrbracket, \mathcal{C}_\alpha \rangle \vdash (y_7^3 \not\approx 27)$
 $\downarrow$ Resolve with $\text{just}(y_7^3 \approx 27 \mapsto^* \text{true})$
10. $\langle \llbracket M_1, f(g(g(x))) \mapsto^* 16, \neg(g(f(53)) \not\approx g(16))^* \rrbracket, \mathcal{C}_\alpha \rangle \vdash (g(f(53)) \not\approx g(16))$
 $\downarrow$ Resolve with $\text{just}(g(f(53)) \not\approx g(16) \mapsto^* \text{false})$
11. $\langle \llbracket M_1, f(g(g(x))) \mapsto^* 16 \rrbracket, \mathcal{C}_\alpha \rangle \vdash (\neg(g(x) \approx x) \vee \neg(x \approx 53) \vee \neg(f(g(g(x))) \approx 16))$
 $\downarrow$ Resolve with $\text{just}(f(g(g(x))) \mapsto^* 16)$
12. $\langle \llbracket (g(x) \approx x)^*, (x - 50 \approx 3)^*, x \mapsto^* 53, z \mapsto_1 \#b01 \rrbracket, \mathcal{C}_\alpha \rangle \vdash$
 $(\neg(g(x) \approx x) \vee \neg(x \approx 53) \vee z >_u \#b10)$
 Let $\mathcal{C}'_\alpha = \mathcal{C}_\alpha \cup \{(\neg(g(x) \approx x) \vee \neg(x \approx 53) \vee z >_u \#b10)\}$.
 $\downarrow$ Backjump
13. $\langle \llbracket (g(x) \approx x)^*, (x - 50 \approx 3)^*, x \mapsto^* 53, (z >_u \#b10)^* \rrbracket, \mathcal{C}'_\alpha \rangle$
 $\downarrow$ Propagate with $\text{just}(z \mapsto^* \#b11) = (\neg(z >_u \#b10) \vee z \approx \#b11)$
14. $\langle \llbracket (g(x) \approx x)^*, (x - 50 \approx 3)^*, x \mapsto^* 53, (z >_u \#b10)^*, z \mapsto^* \#b11 \rrbracket, \mathcal{C}'_\alpha \rangle$
 Let $N_0 = \llbracket (g(x) \approx x)^*, (x - 50 \approx 3)^*, x \mapsto^* 53, (z >_u \#b10)^*, z \mapsto^* \#b11 \rrbracket$.
 $\downarrow$ Propagate with $\text{just}(g(53) \mapsto^* 53) = (\neg(x \approx 53) \vee \neg(g(x) \approx x) \vee g(53) \approx 53)$
15. $\langle \llbracket N_0, g(53) \mapsto^* 53 \rrbracket, \mathcal{C}'_\alpha \rangle$
 $\downarrow$ Decide
16. $\langle \llbracket N_0, g(53) \mapsto^* 53, g(16) \mapsto_1 0 \rrbracket, \mathcal{C}'_\alpha \rangle$
 $\downarrow$ Decide
17. $\langle \llbracket N_0, g(53) \mapsto^* 53, g(16) \mapsto_1 0, f(53) \mapsto_2 0 \rrbracket, \mathcal{C}'_\alpha \rangle$
 $\downarrow$ Decide
18. $\langle \llbracket N_0, g(53) \mapsto^* 53, g(16) \mapsto_1 0, f(53) \mapsto_2 0, g(0) \mapsto_3 53 \rrbracket, \mathcal{C}'_\alpha \rangle$
 $\downarrow$ Decide

19. $\langle \llbracket N_0, g(53) \mapsto^* 53, g(16) \mapsto_1 0, f(53) \mapsto_2 0, g(0) \mapsto_3 53, y_7 \mapsto_4 0_7 \rrbracket, \mathcal{C}'_\alpha \rangle$
 $\downarrow \text{Sat}$
20. $\langle \text{sat}, \llbracket N_0, g(53) \mapsto^* 53, g(16) \mapsto_1 0, f(53) \mapsto_2 0, g(0) \mapsto_3 53, y_7 \mapsto_4 0_7 \rrbracket \rangle$

In this example, we can see that the decision $z \mapsto_1 \#b01$ applied in line 4 caused the conflict. The conflict was detected in line 8 after three propagations. The propagation in line 6 from the theory of uninterpreted functions uses that $g(x) \approx x \wedge x \approx 53 \wedge f(g(g(x))) \approx 16$ implies $g(f(53)) \approx g(16)$, which is equivalent to $\neg(g(f(53)) \approx g(16))$. The subsequent resolution steps (lines 9 – 12) systematically eliminate the assignments that led to the conflict, ultimately learning a new clause that forces a different value for z . This learned constraint guides the search towards a valid model, which is successfully constructed through a series of decisions in the final steps (lines 16 – 19).

Additional Example 2 Assume we have the following instance $\mathcal{C}_\beta = \{C_1, C_2, C_3\}$:

$$\begin{aligned} C_1 &= (z_{53} - z_{53}^2 \approx 51_{53}) \\ C_2 &= (\text{nat2bv}(\text{bv2nat}(x)) \approx y) \\ C_3 &= (\neg(f(x) \approx f(y))) \end{aligned}$$

Here, z from $\mathbb{F}_{53}$ and x, y from $\mathcal{S}_{BV}^8$ are uninterpreted constants. Furthermore, $f : \mathcal{S}_{BV}^8 \rightarrow \mathbb{Z}$ is an uninterpreted function of arity 1. By applying the transition rules of our calculus, we obtain the following trace of search states:

1. $\langle \llbracket \rrbracket, \mathcal{C}_\beta \rangle$
 $\downarrow \text{Propagate with just}(z_{53} - z_{53}^2 \approx 51_{53}) = (z_{53} - z_{53}^2 \approx 51_{53})$
2. $\langle \llbracket (z_{53} - z_{53}^2 \approx 51_{53})^* \rrbracket, \mathcal{C}_\beta \rangle$
 $\downarrow \text{Propagate with just}(\text{nat2bv}(\text{bv2nat}(x)) \approx y) = (\text{nat2bv}(\text{bv2nat}(x)) \approx y)$
3. $\langle \llbracket (z_{53} - z_{53}^2 \approx 51_{53})^*, (\text{nat2bv}(\text{bv2nat}(x)) \approx y)^* \rrbracket, \mathcal{C}_\beta \rangle$
 $\downarrow \text{Propagate with just}(f(x) \approx f(y)) = (\neg(f(x) \approx f(y)))$
4. $\langle \llbracket (z_{53} - z_{53}^2 \approx 51_{53})^*, (\text{nat2bv}(\text{bv2nat}(x)) \approx y)^*, \neg(f(x) \approx f(y))^* \rrbracket, \mathcal{C}_\beta \rangle$
 $\downarrow \text{Decide}$
5. $\langle \llbracket (z_{53} - z_{53}^2 \approx 51_{53})^*, (\text{nat2bv}(\text{bv2nat}(x)) \approx y)^*, \neg(f(x) \approx f(y))^*, z_{53} \mapsto_1 2_{53} \rrbracket, \mathcal{C}_\beta \rangle$
Let $M_1 = \llbracket (z_{53} - z_{53}^2 \approx 51_{53})^*, (\text{nat2bv}(\text{bv2nat}(x)) \approx y)^*, \neg(f(x) \approx f(y))^*, z_{53} \mapsto_1 2_{53} \rrbracket$.
 $\downarrow \text{Decide}$
6. $\langle \llbracket M_1, x \mapsto_2 \#b01100010 \rrbracket, \mathcal{C}_\beta \rangle$
 $\downarrow \text{Propagate with just}(y) = (\text{nat2bv}(\text{bv2nat}(x)) \approx y)$
7. $\langle \llbracket M_1, x \mapsto_2 \#b01100010, y \mapsto^* \#b01100010 \rrbracket, \mathcal{C}_\beta \rangle$
 $\downarrow \text{Conflict from } \mathcal{T}_{UF}$
8. $\langle \llbracket M_1, x \mapsto_2 \#b01100010, y \mapsto^* \#b01100010 \rrbracket, \mathcal{C}_\beta \rangle \vdash (\neg(x \approx y))$
 $\downarrow \text{Resolve with just}(y) = (\text{nat2bv}(\text{bv2nat}(x)) \approx y)$
9. $\langle \llbracket M_1, x \mapsto_2 \#b01100010 \rrbracket, \mathcal{C}_\beta \rangle \vdash (\neg(x \approx \text{nat2bv}(\text{bv2nat}(x))))$
Let $\mathcal{C}'_\beta = \mathcal{C}_\beta \cup \{(\neg(x \approx \text{nat2bv}(\text{bv2nat}(x))))\}$.
 $\downarrow \text{Backjump}$
10. $\langle \llbracket M_1, \neg(x \approx \text{nat2bv}(\text{bv2nat}(x)))^* \rrbracket, \mathcal{C}'_\beta \rangle$
 $\downarrow \text{Conflict from } \mathcal{T}_{BV}$
11. $\langle \llbracket M_1, \neg(x \approx \text{nat2bv}(\text{bv2nat}(x)))^* \rrbracket, \mathcal{C}'_\beta \rangle \vdash (x \approx \text{nat2bv}(\text{bv2nat}(x)))$
 $\downarrow \text{Resolve with just}(x \approx \text{nat2bv}(\text{bv2nat}(x))) = (x \approx \text{nat2bv}(\text{bv2nat}(x)))$
12. $\langle \llbracket (z_{53} - z_{53}^2 \approx 51_{53})^*, (\text{nat2bv}(\text{bv2nat}(x)) \approx y)^*, \neg(f(x) \approx f(y))^*, z_{53} \mapsto_1 2_{53} \rrbracket, \mathcal{C}'_\beta \rangle \vdash (\perp)$
 $\downarrow \text{Drop}$
13. $\langle \llbracket (z_{53} - z_{53}^2 \approx 51_{53})^*, (\text{nat2bv}(\text{bv2nat}(x)) \approx y)^*, \neg(f(x) \approx f(y))^* \rrbracket, \mathcal{C}'_\beta \rangle \vdash (\perp)$
 $\downarrow \text{Unsat}$
14. **unsat**

The instance $\mathcal{C}_\beta$ is unsat because of the clauses $C_2 = (\text{nat2bv}(\text{bv2nat}(x)) \approx y)$ and $C_3 = (\neg(f(x) \approx f(y)))$. The core contradiction arises because C_2 equates y to a function of x that is semantically equivalent to x itself, effectively forcing $x = y$. This directly contradicts C_3 , which requires $f(x) \neq f(y)$ and therefore $x \neq y$. C_2 forces $x = y$ to hold, and C_3 implies $x \neq y$, which leads to the unsatisfiability. In line 9, the Backjump rule is applied to learn $(\neg(x \approx \text{nat2bv}(\text{bv2nat}(x))))$ and propagate $(x \approx \text{nat2bv}(\text{bv2nat}(x))) \mapsto^* \text{false}$ with the justification $\text{just}(x \approx \text{nat2bv}(\text{bv2nat}(x))) = (\neg(x \approx \text{nat2bv}(\text{bv2nat}(x))))$. The learned clause $(\neg(x \approx \text{nat2bv}(\text{bv2nat}(x))))$ captures the fundamental inconsistency that the term $\text{nat2bv}(\text{bv2nat}(x))$ cannot be equal to x under the constraints imposed by the other clauses. This propagated literal immediately leads to the conflict in line 10 because there is no feasible value for x .

Additional Example 3 Let $\mathcal{C}_\gamma$ be the following instance:

$$\begin{aligned} C_1 &= (2x \leq 37) \\ C_2 &= (\text{bv2nat}(y) < x - 3) \\ C_3 &= (\text{nat2bv}(x) \approx \text{or}(y, \#b0011) \vee 20 - x < 2) \end{aligned}$$

Here, $x \in \mathbb{Z}$ and y from $\mathcal{S}_{BV}^4$ are uninterpreted constants. By applying the rules of our calculus, we obtain the following trace of search states:

1. $\langle \llbracket \rrbracket, \mathcal{C}_\gamma \rangle$
 $\downarrow$ Propagate with $\text{just}(2x \leq 37) = (2x \leq 37)$
2. $\langle \llbracket (2x \leq 37)^* \rrbracket, \mathcal{C}_\gamma \rangle$
 $\downarrow$ Propagate with $\text{just}(\text{bv2nat}(y) < x - 3) = (\text{bv2nat}(y) < x - 3)$
3. $\langle \llbracket (2x \leq 37)^*, (\text{bv2nat}(y) < x - 3)^* \rrbracket, \mathcal{C}_\gamma \rangle$
 $\downarrow$ Decide
4. $\langle \llbracket (2x \leq 37)^*, (\text{bv2nat}(y) < x - 3)^*, (20 - x < 2)_1 \rrbracket, \mathcal{C}_\gamma \rangle$
 $\downarrow$ Conflict
5. $\langle \llbracket (2x \leq 37)^*, (\text{bv2nat}(y) < x - 3)^*, (20 - x < 2)_1 \rrbracket, \mathcal{C}_\gamma \rangle \vdash$
 $(\neg(2x \leq 37) \vee \neg(20 - x < 2))$
 Let $\mathcal{C}'_\gamma = \mathcal{C}_\gamma \cup \{(\neg(2x \leq 37) \vee \neg(20 - x < 2))\}$.
 $\downarrow$ Backjump
6. $\langle \llbracket (2x \leq 37)^*, (\text{bv2nat}(y) < x - 3)^*, \neg(20 - x < 2)^* \rrbracket, \mathcal{C}'_\gamma \rangle$
 Let $M = \llbracket (2x \leq 37)^*, (\text{bv2nat}(y) < x - 3)^*, \neg(20 - x < 2)^* \rrbracket$.
 $\downarrow$ Propagate with $\text{just}(\text{nat2bv}(x) \approx \text{or}(y, \#b0011)) =$
 $(\text{nat2bv}(x) \approx \text{or}(y, \#b0011) \vee 20 - x < 2)$
7. $\langle \llbracket M, (\text{nat2bv}(x) \approx \text{or}(y, \#b0011))^* \rrbracket, \mathcal{C}'_\gamma \rangle$
 $\downarrow$ Decide
8. $\langle \llbracket M, (\text{nat2bv}(x) \approx \text{or}(y, \#b0011))^*, x \mapsto_1 -1 \rrbracket, \mathcal{C}'_\gamma \rangle$
 $\downarrow$ Conflict
9. $\langle \llbracket M, (\text{nat2bv}(x) \approx \text{or}(y, \#b0011))^*, x \mapsto_1 -1 \rrbracket, \mathcal{C}'_\gamma \rangle \vdash (0 < x - 3)$
 Let $\mathcal{C}''_\gamma = \mathcal{C}'_\gamma \cup \{(0 < x - 3)\}$.
 $\downarrow$ Backjump
10. $\langle \llbracket M, (\text{nat2bv}(x) \approx \text{or}(y, \#b0011))^*, (0 < x - 3)^* \rrbracket, \mathcal{C}''_\gamma \rangle$
 $\downarrow$ Decide
11. $\langle \llbracket M, (\text{nat2bv}(x) \approx \text{or}(y, \#b0011))^*, (0 < x - 3)^*, x \mapsto_1 7 \rrbracket, \mathcal{C}''_\gamma \rangle$
 $\downarrow$ Conflict
12. $\langle \llbracket M, (\text{nat2bv}(x) \approx \text{or}(y, \#b0011))^*, (0 < x - 3)^*, x \mapsto_1 7 \rrbracket, \mathcal{C}''_\gamma \rangle \vdash$
 $(\neg(2x \leq 37) \vee \neg(\text{bv2nat}(y) < x - 3) \vee \neg(\text{nat2bv}(x) \approx \text{or}(y, \#b0011)))$
 $\downarrow$ Drop
13. $\langle \llbracket M, (\text{nat2bv}(x) \approx \text{or}(y, \#b0011))^*, (0 < x - 3)^* \rrbracket, \mathcal{C}''_\gamma \rangle \vdash$
 $(\neg(2x \leq 37) \vee \neg(\text{bv2nat}(y) < x - 3) \vee \neg(\text{nat2bv}(x) \approx \text{or}(y, \#b0011)))$

↓ Unsat

14. unsat

We have shown that the instance $\mathcal{C}_\gamma$ is unsat. In this example, the final conflict in line 11 was only detected after a different conflict was managed. The final conflict could have been detected in line 7 already. For the conflict detected in line 8, we use the conflict clause $(0 < x - 3)$, which is generated by an oracle that uses the fact $\text{bv2nat}(y) \geq 0$. The clause $(0 < x - 3)$ learned from the first conflict effectively prunes the search space by restricting x to values greater than 3. Here the solver explores multiple possible assignments for x until the final conflict is derived. The final conflict demonstrates that even when x is assigned a value like 7, which satisfies all individual theory constraints, the specific combination of constraints from C_1 , C_2 , and C_3 is inherently unsatisfiable. The Drop rule can be applied in line 12 because all literals of the conflict clause $(\neg(2x \leq 37) \vee \neg(\text{bv2nat}(y) < x - 3) \vee \neg(\text{nat2bv}(x) \approx \text{or}(y, \#b0011)))$ evaluate to false under the current trail. This is due to the fact that their negations $2x \leq 37$, $\text{bv2nat}(y) < x - 3$ and $\text{nat2bv}(x) \approx \text{or}(y, \#b0011)$ are on the current trail and evaluate to true.