A Two-Watched Literal Scheme for First-Order Logic

Martin Bromberger[1], Tobias Gehl[1], Lorenz Leutgeb[1,2] and Christoph Weidenbach[1]

[1]*Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany*
[2]*Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany*

Abstract

The two-watched literal scheme for propositional logic is a core component of any efficient CDCL (Conflict Driven Clause Learning) implementation. The family of SCL (Clause Learning from Simple Models) calculi also learns clauses with respect to a partial model assumption built by decisions and propagations similar to CDCL. We show that the well-known two-watched literal scheme can be lifted to SCL for first-order logic.

Keywords

CDCL, SCL, two-watched literal scheme, first-order logic

1. Introduction

The two-watched literal scheme is an indispensable part of any CDCL SAT solver implementation [1, 2]. CDCL SAT solvers build an explicit partial model assumption, called a *trail*, and check whether clauses with respect to this model assumption propagate or whether they are falsified. Propagation here means the clause is false except for one undefined literal. This literal needs then to be propagated in order to satisfy the clause. For example, with respect to the partial model assumption $[Q]$ the clause $P \vee \neg Q \vee R$ is neither false nor does it propagate. With respect to propagation and falsification, there is no need to do any update on the status of a clause, as long as it has two undefined literals or it is true in the current model assumption. This leads to the two-watched literal scheme. In any clause two literals are watched, e.g., we can watch P and R in the above clause represented by the triple $(P \vee \neg Q \vee R; P; R)$. Then only the watched literals are indexed with respect to the trail, i.e., for a trail $[Q]$ the above clause is not visited because $\neg Q$ is not watched in the clause. The first benefit of the two-watched literal scheme is that the clause only needs to be considered on trail extensions if a watched literal is concerned. For a trail $[\neg P]$ the above clause is visited, P is false with respect to the trail $[\neg P]$ and the watched literals are updated resulting in $(P \vee \neg Q \vee R; \neg Q; R)$. Now

PAAR'22: 8th Workshop on Practical Aspects of Automated Reasoning, August 11–12, 2022, Haifa, Israel
EMAIL: mbromber@mpi-inf.mpg.de (M. Bromberger); tgehl@mpi-inf.mpg.de (T. Gehl); lorenz@mpi-inf.mpg.de (L. Leutgeb); weidenb@mpi-inf.mpg.de (C. Weidenbach)
ORCID: 0000-0001-7256-2190 (M. Bromberger); 0000-0003-0391-3430 (L. Leutgeb); 0000-0001-6002-0458 (C. Weidenbach)

if the trail is extended to $[\neg PQ]$ the clause will be visited again and it will be detected that it propagates R. The general invariant kept by the watched literal scheme for a clause with respect to a trail is: either one of the watched literals is true or if there is a watched false literal then there are no literals in the clause that are true or undefined and not watched. The second benefit is that the two-watched literal scheme invariant is invariant with respect to a shrinking trail, because this just turns true/false literals into undefined literals.

In this paper we lift the propositional two-watched literal scheme to first-order logic without equality. Our trail consists of ground first-order literals and clauses are full first-order clauses containing implicitly universally quantified variables. Again we want to detect propagating and false clauses by the two-watched literal scheme by only considering the watched literals for trail extensions. For first-order logic the two-watched literals scheme gets more sophisticated because of variable instantiation. The first extension concerns universally quantified variables. Variable instantiation may result in merging literals and different clauses may produce identical instances by variable instantiation. For example, the first-order clause $R(x, y) \vee R(a, z) \vee R(u, b)$ where a, b are constants and x, y, z, u are variables represents already the propagating ground instance $R(a, b)$. Therefore, our two-watched literal scheme contains the rule FactorizeWatched for considering common instances of watched literals and assumes that all starting units, including units built by instantiation are performed right from the start. Furthermore, updates on the trail induce further instances of clauses, in general, represented by the rule CreateInstance. The second extension addresses potentially infinite trails. Consider the trail $[P(a)]$ and the clause $\neg P(x) \vee P(g(x))$. Already this clause produces an infinite trail if propagation is done exhaustively $[P(a), P(g(a)), P(g(g(a))), \dots]$. Even in a first-order setting without non-constant function symbols, the trail may grow exponentially with respect to the maximal arity of a predicate symbol. Thus exhaustive propagation cannot be afforded in first-order logic, in general. On the other hand any CDCL style calculus typically breaks if a decided (guessed) literal immediately results in a false clause, a conflict. Our solution to this is a one step propagation look-ahead, represented by the rule DetectPropLiteral and by separating the trail from a set of potentially propagating literals. This way our two-watched literal scheme serves the SCL family of calculi considering a first-order language [3, 4].

The paper is organized as follows: after a section on preliminaries, Section 2, we present the TWFO calculus and prove that it detects all conflicts and propagations, Section 3. Then we present some intuitive example runs of TWFO, Section 4. Next we show in Section 5 that TWFO finds all conflicts and propagations with a minimal amount of overhead: it only needs to check a clause instance for propagations and conflicts if the clause instance was recently derived or if an instance of one of its watched literals was assigned to false. Following this, we show how the interaction between the TWFO calculus and the SCL calculus works in detail. The paper ends with a discussion of the obtained results and directions for future work, Section 7.

2. Preliminaries

We assume a first-order language without equality where N denotes a clause set; C, D denote clauses; L, K, H denote literals; A, B denote atoms; P, Q, R denote predicates; t, s terms; f, g, h function symbols; a, b, c constants; and x, y, z variables. Atoms, literals, clauses and clause sets are considered as usual, where in particular clauses are identified both with their disjunction and multiset of literals. The complement of a literal is denoted by the function comp. Semantic entailment $\models$ is defined as usual where variables in clauses are assumed to be universally quantified. Substitutions σ, τ are total mappings from variables to terms, where $\mathrm{dom}(\sigma) := \{x \mid x\sigma \neq x\}$ is finite and $\mathrm{codom}(\sigma) := \{t \mid x\sigma = t, x \in \mathrm{dom}(\sigma)\}$. Their application is extended to literals, clauses, and sets of such objects in the usual way. A term, atom, clause, or a set of these objects is *ground* if it does not contain any variable. A substitution σ is *ground* if $\mathrm{codom}(\sigma)$ is ground. A substitution σ is *grounding* for a term t, literal L, clause C if $t\sigma$, $L\sigma$, $C\sigma$ is ground, respectively. A substitution σ is the *minimally grounding substitution* for a term/atom/literal/clause Z if $Z\sigma$ is ground and there exist no substitutions τ and ρ such that $\tau\rho = \sigma$ and $Z\tau$ is ground.

The function mgu denotes the *most general unifier* of two terms, atoms, literals. We assume that any mgu of two terms or literals does not introduce any fresh variables and is idempotent. The function $\mathrm{vars}(Z)$ returns the set of variables in a term/atom/literal/clause/clause set Z.

Let $\prec_B$ denote a well-founded, total, strict ordering on ground literals such that for any ground literal L there are only finitely many ground literals K with $K \prec_B L$. For example, a Knuth-Bendix ordering has this property [5]. Let L be a ground literal L and let K be a non-ground literal. Then we write $K \prec_B L$ in order to denote that there exists a grounding τ for K such that $K\tau \prec_B L$ and $C \prec_B L$ in order to denote that there exists a grounding τ for C such that $K\sigma\tau \prec_B L$ for all $K \in C$.

3. The Two-Watched Literal Calculus for First-Order Clauses

The Two-Watched Literal Calculus for First-Order Clauses (TWFO) calculus is a lazy two-watched literals scheme for first-order logic without equality. Its purpose is to efficiently detect propagations and all immediate conflicts with respect to a ground partial model M and a set of first-order clauses without equality N. The calculus works on a tuple $(M; \beta; O; F; D)$ called a *state* where M is a sequence of ground literals called a *trail*, β is the *limit* of the state, i.e., a ground literal limiting the considered ground literals and clause instances, O is a set of triples $(C; L_1; L_2)$ where C is a first-order clause and L_1 and L_2 are the two literals that are watched in C, F is a set of annotated literals $L^{C;K}$ that can be propagated with the clause instance C, and D is the conflict clause or $\top$. Annotations for literals in F mark the leftmost literal in the trail such that the respective clause propagates and $\top$ if the literal is propagated by a (implicit after factoring) unit clause. Annotations are only explicitly written where they are needed or changed.

The literals on the trail M are annotated either with a positive natural number or with a

clause instance. These annotations are just the standard annotations used in CDCL [1, 2, 6] and SCL [3, 4]. So a literal annotated with a positive natural number k means that this literal is a decision literal of level k and a literal annotated with a clause instance $C\sigma$ means that this literal is a propagated literal and that the clause instance $C\sigma$ propagates that literal. A literal L is called *assigned* with respect to a sequence M of ground literals, if either $L \in M$ or $\text{comp}(L) \in M$, i.e., it is either true or false, otherwise it is called *unassigned*. A ground clause $\{L_1, \ldots, L_n\}$ is called *assigned* with respect to a sequence M of ground literals, if either $\{L_1, \ldots, L_n\} \cap M \neq \emptyset$ or $\{\text{comp}(L_1), \ldots, \text{comp}(L_n)\} \subseteq M$, i.e., it is either true or false, otherwise it is called *unassigned*. A clause C can *propagate* the literal L with respect to a trail M if $C = C' \vee L \vee \ldots \vee L$ and L is unassigned with respect to M and $M \models \neg C'$. In the following the watched literals L_1 and L_2 are considered interchangeably. A literal $L_j\sigma$ is *fresh* out of $(C; L_1; L_2)$ if $L_j\sigma \in C\sigma$, $L_j\sigma \neq L_1\sigma$, $L_j\sigma \neq L_2\sigma$.

TWFO is called lazy because: (i) TWFO does not check literals for all its instances, i.e., a literal with variables may be watched although all its instances are false. (ii) TWFO does not propagate all propagatable literals it detects but only stores them in the *set of detected propagations* F; this is useful for first-order logic because there are infinitely many ground instances per predicate.

For a finite set of clauses N, also called the *initial clauses*, the *start state* for TWFO is $(\epsilon; \beta; O; F; \top)$ where (i) we choose the starting limit β such that $C \prec_B \beta$ for all clauses $C \in N$, (ii) the starting set O consists of exactly one instance $(C; L_1; L_2)$ for all starting clauses $C \in N$ such that $L_1 \in C$ and $L_2 \in C$ and $L_1 \neq L_2$ if possible, and (iii) the starting set F consists of all literals $L^{C,\top}$ we can propagate from starting units C in N, including units $C\sigma = L \vee \cdots \vee L$ (where $C\sigma \prec_B \beta$) that we can build by factoring a clause $C \in N$, e.g., if N contains the clause $P(x) \vee P(a)$, then $P(x) \vee P(a)$ can be factored to $P(a) \vee P(a)$ so the starting F must contain the literal $P(a)$. Note that we only allow non-empty clauses in N. The rules of the calculus are as follows:

CreateInstance $(M; \beta; O \uplus \{(C; L_1; L_2)\}; F; \top) \Rightarrow_{\text{TWFO}} (M; \beta; O \cup \{(C; L_1; L_2), (C\sigma; L_j\sigma; L_k\sigma)\}; F; \top)$

provided $\text{comp}(L_1\sigma) \in M$ for some minimal grounding substitution σ,
$C\sigma \prec_B \beta$, $L_j\sigma \in C\sigma$,
if possible $L_j\sigma \in M$ otherwise if possible $L_j\sigma$ unassigned otherwise $M = M_1' \,\text{comp}(L_j\sigma) M_1''$ and for all literals $L\sigma \in C\sigma$ holds $\text{comp}(L\sigma) \notin M_1''$,
if $\text{comp}(L_2\sigma) \notin M$ then $L_k = L_2$ else $L_k\sigma \in C\sigma$ and if possible $L_k\sigma \neq L_j\sigma$ and if possible $L_k\sigma \in M$ otherwise if possible $L_k\sigma$ unassigned otherwise $M = M_2' \,\text{comp}(L_k\sigma) M_2''$ and for all literals $L\sigma \in C\sigma \setminus \{L_j\sigma\}$ holds $\text{comp}(L\sigma) \notin M_2''$,
there exists no $K_1\sigma, K_2\sigma \in C\sigma$ such that $(C\sigma; K_1\sigma; K_2\sigma) \in O$.

CreateInstance creates an instance as long as a watched literal gets false by instantiation and the created instance is new.

UpdateWatched $(M; \beta; O \uplus \{(C; L_1; L_2)\}; F; \top) \Rightarrow_{\text{TWFO}} (M; \beta; O \cup \{(C; L_j; L_2)\}; F; \top)$

provided $\text{comp}(L_1) \in M$,

$L_2 \notin M$,
fresh $L_j \in C$ where $L_j \in M$ or if no such L_j exists L_j is unassigned.

UpdateWatched exchanges a false watched literal as long as the other watched is not true. It is replaced by a true or if such a literal does not exist, an unassigned literal.

FactorizeWatched $(M; \beta; O \uplus \{(C; L_1; L_2)\}; F; \top) \Rightarrow_{\text{TWFO}} (M; \beta; O' \cup \{(C; L_1; L_2), (C\sigma; L_j\sigma; L_k\sigma)\}; F; \top)$
provided $L_1\sigma = L_2\sigma$ for mgu σ,
$C\sigma \prec_B \beta$, fresh $L_j\sigma \in C\sigma$,
if possible $L_j\sigma \in M$ otherwise if possible $L_j\sigma$ unassigned otherwise $M = M_1' \operatorname{comp}(L_j\sigma) M_1''$ and for all literals $L\sigma \in C\sigma$ holds $\operatorname{comp}(L\sigma) \notin M_1''$,
if $\operatorname{comp}(L_2\sigma) \notin M$ then $L_k = L_2$ else $L_k\sigma \in C\sigma$ and if possible $L_k\sigma \neq L_j\sigma$ and if possible $L_k\sigma \in M$ otherwise if possible $L_k\sigma$ unassigned otherwise $M = M_2' \operatorname{comp}(L_k\sigma) M_2''$ and for all literals $L\sigma \in C\sigma \setminus \{L_j\sigma\}$ holds $\operatorname{comp}(L\sigma) \notin M_2''$,
there exists no $K_1\sigma, K_2\sigma \in C\sigma$ such that $(C\sigma; K_1\sigma; K_2\sigma) \in O$.

DetectPropLiteral $(M; \beta; O \uplus \{(C; L_1; L_2)\}; F; \top) \Rightarrow_{\text{TWFO}} (M; \beta; O \cup \{(C; L_1; L_2)\}; F \cup \{L_2^{C;K}\}; \top)$
provided $\operatorname{comp}(L_1) \in M$,
$C = C_0 \vee L_2 \vee \ldots \vee L_2$,
L_2 unassigned,
$M \models \neg C_0$,
$M = M'KM''$ and $\operatorname{comp}(K) \in C$, and for all $L \in C$ holds $\operatorname{comp}(L) \notin M''$,
there is no $H^{D;K'} \in F$ such that $H\tau = L_2$ for some τ with $D\tau \prec_B \beta$ and $K' \in M'K$.

So DetectPropLiteral factorizes implicitely on the unassigned literal and propagates as long as the literal is not known to be propagating. Note that we annotate the literal that we propagate with the last literal on the trail that falsifies a literal in the clause. As we will see below in a reasonable strategy this will be the last literal on the trail, that equals $\operatorname{comp}(L_1)$, except directly after backtracking. Also only in a reasonable strategy the condition that L_2 is assigned can only occur directly after backtracking, because DetectPropLiteral is complete for a reasonable strategy, as we will see.

Conflict $(M; \beta; O \uplus \{(C; L_1; L_2)\}; F; \top) \Rightarrow_{\text{TWFO}} (M; \beta; O \cup \{(C; L_1; L_2)\}; F; C)$
provided $\operatorname{comp}(L_1) \in M$ and $\operatorname{comp}(L_2) \in M$,
$M \models \neg C$.

ConflictF $(M; \beta; O; F \uplus \{L_1^{C_1}, L_2^{C_2}\}; \top) \Rightarrow_{\text{TWFO}} (ML_1\sigma^{C_1\sigma}; \beta; O; F \cup \{L_1^{C_1}, L_2^{C_2}\}; C_2\sigma)$
there is a grounding unifier σ such that $L_1\sigma = \operatorname{comp}(L_2\sigma)$,
$L_1\sigma \notin M$, $L_2\sigma \notin M$,
$C_1\sigma \prec_B \beta$ and $C_2\sigma \prec_B \beta$

Note that in the ConflictF rule we do not need the prerequisites that $\operatorname{comp}(L_1\sigma) \notin M$ and $\operatorname{comp}(L_2\sigma) \notin M$ hold because $\operatorname{comp}(L_1\sigma) = L_2\sigma \notin M$ and $\operatorname{comp}(L_2\sigma) = L_1\sigma \notin M$ hold.

DecLiteral(L^k) $(M;\beta;O;F;\top) \Rightarrow_{\text{TWFO}} (ML^k;\beta;O;F;\top)$
provided L ground and not defined in M,
there is no $L' \in F$ and grounding substitution σ such that $\text{comp}(L) = L'\sigma$ and $L\sigma \prec_B \beta$

PropLiteral($L\sigma^{C\sigma}$) $(M;\beta;O;F;\top) \Rightarrow_{\text{TWFO}} (ML\sigma^{C\sigma};\beta;O;F;\top)$
provided $L\sigma$ ground and not defined in M,
there is a literal $L^C \in F$, $C\sigma \prec_B \beta$

RemoveLiteral $(ML;\beta;O;F;D) \Rightarrow_{\text{TWFO}} (M;\beta;O;F';D)$
provided $F' \subseteq F$,
for all $L'^{C;K} \in F'$ holds that $L \neq K$,
for all $L'^{C;K} \in F \setminus F'$ holds that $L = K$,
$D \neq \top$

The clause D is the original conflict clause and the clause C in the rule Backtrack(C) below is the clause that we get from resolution in the SCL(T) calculus and learn as a new clause. The learned clause C in Backtrack(C) is a new clause and not an instance of an already existing clause, as we know from the SCL(T) calculus, and is therefore not already in O.

Backtrack(C) $(M;\beta;O;F;D) \Rightarrow_{\text{TWFO}} (M;\beta;O' \cup \{(C;L_1;L_2)\};F';\top)$
provided $D \neq \top$,
there is a grounding σ such that $C\sigma$ can propagate $L_1\sigma$, $C\sigma \prec_B \beta$
there is no τ and $M = M'M''$ where M'' is non-empty such that $C\tau$ can propagate with respect to M', $C\tau \prec_B \beta$
$L_1, L_2 \in C$,
$L_1 \neq L_2$ if C contains different literals,
L_1 unassigned,
if possible L_2 unassigned otherwise $M = M'\,\text{comp}(L_2)M''$ and for all literals $L \in C$ holds $\text{comp}(L) \notin M''$,
if there is a minimal substitution τ with $C\tau \prec_B \beta$ such that $C\tau = L\tau \vee \ldots \vee L\tau$ then $F' = F \cup \{L\tau^{C\tau;\top}\}$ else $F' = F$

The *learned clause* C in the Backtrack rule is a clause with a ground instance that propagates directly. So there are no true literals in C and at least one unassigned literal, the one that propagates. All other literals can be unassigned if not ground or false if ground. So we can watch one unassigned literal and the other watched literal can be unassigned or false.

Forget(V) $(M;\beta;O;F;\top) \Rightarrow_{\text{TWFO}} (\epsilon;\beta;O';F';\top)$
provided $O' = \{(C;K_1;K_2) \in O \mid C \notin V\}$ and $F' = \{L^{C\sigma;\top} \in F\}$

Note that the Forget rule only keeps clause instances annotated with $\top$. Also we only forget clauses when we restart and therefore the rule Forget(V) not only removes the given clauses V but also restarts with an empty trail and removes all clause instances that are not original instances. It also removes all literals in F that can not be directly

propagated from the kept clauses, i.e. literals that can be propagated from factorized unit instances.

Grow(β') $(\epsilon; \beta; O; F; \top) \Rightarrow_{\text{TWFO}} (\epsilon; \beta'; O; F; \top)$
provided $\beta \prec_B \beta'$.

The TWFO-calculus is part of a larger calculus to efficiently detect propagations and conflicts. So the rules DecLiteral(L^k), PropLiteral($L\sigma^{C\sigma}$), RemoveLiteral, Backtrack(C), Forget(V) and Grow(B') are rules that are used to update the TWFO-calculus such that it has the same trail, uninstantiated clauses and the same state of the conflict clause, if it is $\top$ or not, as the larger calculus. Therefore the rules DecLiteral(L^k), PropLiteral($L\sigma^{C\sigma}$), Backtrack(C), Forget(V) and Grow(B') have arguments.

CDCL-like calculi typically only forget clauses after a restart. Hence, Forget(V) not only removes the clauses V but also restarts with an empty trail and removes all clause instances that are not original instances. It also removes all literals in F that can not be directly propagated from the kept clauses, i.e. literals that can be propagated from factorized unit instances.

The TWFO-calculus is supposed to efficiently detect propagations and conflicts for a larger calculus. The intended larger calculus is the SCL calculus presented in [3], but there are also other calculi to which it could be adapted, e.g., the SCL(T) calculus [4]. The rules DecLiteral(L^k), PropLiteral($L\sigma^{C\sigma}$), RemoveLiteral, Backtrack(C), Forget(V), and Grow(β') are rules that are used to update the TWFO calculus such that it has the same trail, uninstantiated clauses and the same state of the conflict clause, if it is $\top$ or not, as the larger calculus. Therefore the rules DecLiteral(L^k), PropLiteral($L\sigma^{C\sigma}$), Backtrack(C), Forget(V), and Grow(β') have arguments. In order to keep TWFO in sync with the SCL calculus, it has to detect all propagations and conflicts before SCL can change the trail with the rules DecLiteral(L^k) and PropLiteral($L\sigma^{C\sigma}$). We ensure this with the help of a strategy:

Definition 1 (Reasonable Strategy)**.** *A strategy is called* reasonable *if the rules CreateInstance, UpdateWatched, FactorizeWatched, DetectPropLiteral, and Conflict are preferred over the rules DecLiteral(L^k) and PropLiteral($L\sigma^{C\sigma}$). To newly created instances by CreateInstance, UpdateWatched, FactorizeWatched, Backtrack(C) these rules are exhaustively applied before any other rule is applied. The rules ConflictF, RemoveLiteral, Backtrack(C) may be applied without further restrictions.*

Given this strategy TWFO actually finds all propagations and conflicts before SCL can change the trail. But in order to prove this, we first must ensure some invariants that each state reachable by TWFO from a starting state must fulfill:

Definition 2 (Consistent State)**.** *A state $(M; \beta; O; F; D)$ is called* consistent *if all of the following properties hold:*

1. *Every instance $(D; L_1; L_2) \in O$ is an instance of an initial clause or of a learned clause.*
2. *For every clause C there is at most one instance $(C; L_1; L_2) \in O$.*

3. *For every instance* $(C; L_1; L_2) \in O$ *it holds that* $L_1 \in C$ *and* $L_2 \in C$.
4. *For a clause instance* $(C; L_1; L_2) \in O$ *it holds that either all literals in* C *are equal or* $L_1 \neq L_2$.
5. *The trail* M *only contains ground literals and does define a literal at most once.*
6. *For all literals* $L^{C,K} \in F$ *with* $K \neq \top$ *it holds that* $M = M'KM''$, $C = C_0 \vee L \vee \cdots \vee L$, $M'K \models \neg C_0$, *and* L *is undefined in* $M'K$.
7. *For all literals* $L^{C,\top} \in F$ *it holds that* $C = L \vee \cdots \vee L$.
8. *For all literals* $L^{C,K} \in F$ *there exists an instance* $(D; L_1; L_2) \in O$, *a literal* $L' \in D$, *and a substitution* σ *such that* $C = D\sigma$ *and* $L = L'\sigma$.
9. *For all literals* $L_1^C \in M$ *it holds that there exists a literal* $L_2^{D,K} \in F$ *and a substitution* σ *such that* $L_1 = L_2\sigma$, $C = D\sigma$, $D = D_0 \vee L_2 \vee \cdots \vee L_2$, *and if* K *is a literal, then* $M = M_1 K M_2 L_1 M_3$.
10. *For every instance* $(C; L_1; L_2) \in O$ *for which there exists a substitution* σ *such that* $C\sigma = L \vee \cdots \vee L$ *and* $C\sigma \prec_B \beta$, *there also must exist a* $K^{D,\top} \in F$ *and a substitution* τ *such that* $L = K\tau$ *and* $D\tau = L \vee \cdots \vee L$.
11. *For every instance* $(D; L_1; L_2) \in O$, $D \prec_B \beta$.
12. *For all literals* $L_1^C \in M$ *it holds that there exists a clause instance* $(C; K_1; K_2) \in O$, *a literal* $L' \in C$, *and a substitution* σ *such that* $L_1 = L'\sigma$ *and* $L_1 \prec_B \beta$.
13. *For every literal* $L_2^{D,K} \in F$, $L_2 \prec_B \beta$.

Lemma 1 (Consistency of Starting States)**.** *A starting state* $(\epsilon; \beta; O; F; \top)$ *for clause set* N *is consistent.*

Lemma 2 (Preservation)**.** *Let* $(M; \beta; O; F; D)$ *be a consistent state. Then any state reachable from* $(M; \beta; O; F; D)$ *by a sequence of TWFO reasonable rule applications is consistent.*

Theorem 1 (Correctness)**.** *The following properties hold for a consistent state* $(M; \beta; O; F; D)$ *and every clause instance* $(C, L_1, L_2) \in O$*:*

1. *If* $D = \top$ *and there is a ground instance* $C\sigma \prec_B \beta$ *such that* $M \models \neg C\sigma$, *then Conflict or ConflictF is applicable to* $C\sigma$ *after a series of rule applications that consist only of the rules CreateInstance, UpdateWatched, and FactorizeWatched.*
2. *If* $D = \top$ *and there is a ground instance* $C\sigma \prec_B \beta$ *such that* $M \models \neg C\sigma$, *then any sequence of rule applications that follows a reasonable strategy will apply one of the rules Conflict, ConflictF, or Forget(*V*), before the rules DecLiteral(*L^k*) and PropLiteral(*$L\sigma^{C\sigma}$*) are applicable. This means if there is a clause in conflict with the trail, then the calculus will detect a conflict clause or forget/reset the trail, before it can propagate or decide new literals onto the trail.*
3. *If* $D = \top$ *and there is an instance* $C\sigma = C'\sigma \vee L\sigma \vee \cdots \vee L\sigma$ *where* $C\sigma \prec_B \beta$, $C'\sigma$ *is ground,* $M \models \neg C'\sigma$ *and* $L\sigma$ *is unassigned with respect to* M, *and there exists no* $H \in F$ *and substitution* τ *such that* $H\tau = L\sigma$, *then DetectPropLiteral is applicable to* $L\sigma$ *or a literal* H *that can be instantiated to* $L\sigma$ *after a series of rule applications that consist only of the rules CreateInstance, UpdateWatched, and FactorizeWatched.*

4. *If $D = \top$ and there is an instance $C\sigma = C'\sigma \vee L\sigma \vee \cdots \vee L\sigma$ where $C\sigma \prec_B \beta$, $C'\sigma$ is ground, $M \models \neg C'\sigma$ and $L\sigma$ is unassigned with respect to M, and there exists no $H \in F$ and substitution τ such that $H\tau = L\sigma$, then any sequence of rule applications that follows a reasonable strategy will apply one of the rules DetectPropLiteral, Conflict, ConflictF, or Forget(V), before the rules DecLiteral(L^k) and PropLiteral($L\sigma^{C\sigma}$) are applicable. This means if there exists a propagatable literal that has not been detected yet, then the calculus will detect a new propagatable literal or a conflict or forget/reset the trail, before it can propagate or decide new literals onto the trail.*
5. *If $D = \top$, then exhaustively applying CreateInstance, UpdateWatched, FactorizeWatched, and DetectPropLiteral will terminate after a finite number of rule applications.*
6. *If there is a transition $(M; \beta; O; F; \top) \Rightarrow_{TWFO}^{Conflict} (M; \beta; O; F; D\sigma)$ then there is a conflict $M \models \neg D\sigma$ and $D\sigma \prec_B \beta$ is a ground instance of an initial clause or of a learned clause.*
7. *If there is a transition $(M; \beta; O; F; \top) \Rightarrow_{TWFO}^{ConflictF} (ML; \beta; O; F; D\sigma)$ then there is a conflict $ML \models \neg D\sigma$ and $D\sigma \prec_B \beta$ is a ground instance of an initial clause or of a learned clause.*
8. *If there is a transition $(M; \beta; O; F; \top) \Rightarrow_{TWFO}^{DetectPropLiteral} (M; \beta; O; F \cup \{L^D\}; \top)$ then there is a propagation where $D = D' \vee L \vee \cdots \vee L$, $D \prec_B \beta$, and $M \models \neg D'$, D is an instance of an initial clause or of a learned clause, and D' is ground.*

Let $(M; \beta; O; F; \top)$ be a consistent state. Together the above properties show that our calculus is correct. That means in any reasonable run starting from $(M; \beta; O; F; \top)$, TWFO tries to exhaustively apply the rules CreateInstance, UpdateWatched, FactorizeWatched, and DetectPropLiteral. This can end in one of three ways (Theorem 1.1 and 1.3): (i) TWFO finds a conflict and interrupts the exhaustive exploration with Conflict or ConflictF, (ii) TWFO finds all literals that can be potentially propagated, and (iii) TWFO interrupts the exhaustive exploration by resetting the trail with rule Forget(V). Either way, the exhaustive exploration for trail M will terminate (Theorem 1.5). With regard to DecLiteral(L^k) and PropLiteral this means that TWFO will detect any conflicts and any literals that can be potentially propagated before the trail can be extended by DecLiteral(L^k) and PropLiteral (Theorem 1.2 and 1.3). Moreover, all detected conflicts and propagatable literals are sound (Theorem 1.6–8).

4. Examples

We now present some examples that intuitively show how each rule of the TWFO calculus is used. Note that the trails in the examples are not annotated with the clauses that propagated the literals to shorten the notation. Also instead of writing down the tuple $(C; L_1; L_2)$ for elements in the set O we will annotate watched literals with *.

Example 1 (Example run 1)**.** *Let $N = \{(1)\ P(x) \vee \neg Q(x) \vee R(x, y), (2)\ P(x) \vee Q(a), (3)\ P(a) \vee \neg R(x, b)\}$ be the set of starting clauses and let $\prec_B$ be an ordering and β be a ground literal such that $L \prec_B \beta$ iff the predicate of L is P, Q, or R and $\text{const}(L) \subseteq \{a, b\}$. Let*

O_0, O_1, and O_2 be sets of clause instances such that
$O_0 = \{(1)P(x)^* \vee \neg Q(x)^* \vee R(x,y), (2)P(x)^* \vee Q(a)^*, (3)P(a)^* \vee \neg R(x,b)^*\}$,
$O_1 = O_0 \cup \{(1.1)P(a) \vee \neg Q(a)^* \vee R(a,y)^*\}$,
$O_2 = O_1 \cup \{(4)P(a)^* \vee P(x)^*\}$
O_0 is our starting set of clause instances and the others will be reached during the run. Then the following is a run of the TWFO calculus:

$$
\begin{aligned}
&(\epsilon; \beta; O_0; \emptyset; \top)\\
\Rightarrow^{DecLiteral(\neg P(a)^1)}_{TWFO} \quad &(\neg P(a)^1; \beta; O_0; \emptyset; \top)\\
\Rightarrow^{CreateInstance(1)}_{TWFO} \quad &(\neg P(a)^1; \beta; O_1; \emptyset; \top)\\
\Rightarrow^{DetectPropLit(2)}_{TWFO} \quad &(\neg P(a)^1; \beta; O_1; \{Q(a)^{P(a)\vee Q(a);\neg P(a)}\}; \top)\\
\Rightarrow^{DetectPropLit(3)}_{TWFO} \quad &(\neg P(a)^1; \beta; O_1; \{Q(a)^{P(a)\vee Q(a);\neg P(a)}, \neg R(x,b)^{P(a)\vee \neg R(x,b);\neg P(a)}\}; \top)\\
\Rightarrow^{PropLiteral(Q(a))}_{TWFO} \quad &(\neg P(a)^1 Q(a); \beta; O_1; \{Q(a)^{P(a)\vee Q(a);\neg P(a)},\\
&\neg R(x,b)^{P(a)\vee \neg R(x,b);\neg P(a)}\}; \top)\\
\Rightarrow^{DetectPropLit(1.1)}_{TWFO} \quad &(\neg P(a)^1 Q(a); \beta; O_1; \{Q(a)^{P(a)\vee Q(a);\neg P(a)}, \neg R(x,b)^{P(a)\vee \neg R(x,b);\neg P(a)},\\
&R(a,y)^{P(a)\vee \neg Q(a)\vee R(a,y);Q(a)}\}; \top)\\
\Rightarrow^{PropLiteral(R(a,b))}_{TWFO} \quad &(\neg P(a)^1 Q(a) R(a,b); \beta; O_1; \{Q(a)^{P(a)\vee Q(a);\neg P(a)},\\
&\neg R(x,b)^{P(a)\vee \neg R(x,b);\neg P(a)}, R(a,y)^{P(a)\vee \neg Q(a)\vee R(a,y);Q(a)}\}; \top)\\
\Rightarrow^{Conflict(3)}_{TWFO} \quad &(\neg P(a)^1 Q(a) R(a,b); \beta; O_1; \{Q(a)^{P(a)\vee Q(a);\neg P(a)},\\
&\neg R(x,b)^{P(a)\vee \neg R(x,b);\neg P(a)}, R(a,y)^{P(a)\vee \neg Q(a)\vee R(a,y);Q(a)}\};\\
&P(a) \vee \neg R(a,b))\\
\Rightarrow^{RemoveLiteral}_{TWFO} \quad &(\neg P(a)^1 Q(a); \beta; O_1; \{Q(a)^{P(a)\vee Q(a);\neg P(a)}, \neg R(x,b)^{P(a)\vee \neg R(x,b);\neg P(a)},\\
&R(a,y)^{P(a)\vee \neg Q(a)\vee R(a,y);Q(a)}\}; P(a) \vee \neg R(a,b))\\
\Rightarrow^{RemoveLiteral}_{TWFO} \quad &(\neg P(a)^1; \beta; O_1; \{Q(a)^{P(a)\vee Q(a);\neg P(a)}, \neg R(x,b)^{P(a)\vee \neg R(x,b);\neg P(a)}\};\\
&P(a) \vee \neg R(a,b))\\
\Rightarrow^{RemoveLiteral}_{TWFO} \quad &(\epsilon; \beta; O_1; \emptyset; P(a) \vee \neg R(a,b))\\
\Rightarrow^{Backtrack(4)}_{TWFO} \quad &(\epsilon; \beta; O_2; \{P(a)^{P(a)\vee P(x);\top}\}; \top)
\end{aligned}
$$

In the last step the notation for the rule Backtrack($P(a) \vee P(x)$) was shortened to Backtrack(4). Note that in the last step where we use the rule Backtrack($P(a) \vee P(x)$) we got the clause $P(a) \vee P(x)$ just from resolving the conflict clause with the propagating clauses and fully factorizing the resulting clause. We use the original uninstantiated clauses to resolve.

Example 2 (Example run 2)**.** *Let the clause set $N = \{P(x) \vee Q(a) \vee Q(x), P(x) \vee \neg Q(x) \vee$*

$Q(b), \neg Q(a) \vee \neg Q(b)\}$ *be the set of starting clauses and and let* $\prec_B$ *be an ordering and* β *be a ground literal such that* $L \prec_B \beta$ *iff the predicate of* L *is* P *or* Q *and* $\text{const}(L) \subseteq \{a, b\}$. *Let* O_0, O_1, O_2, *and* O_3 *be sets of clause instances such that*
$O_0 = \{(1)P(x) \vee Q(a)^* \vee Q(x)^*, (2)P(x)^* \vee \neg Q(x) \vee Q(b)^*, (3)\neg Q(a)^* \vee \neg Q(b)^*\}$,
$O_1 = O_0 \cup \{(1.1)P(a)^* \vee Q(a)^* \vee Q(a)\}$,
$O_2 = O_1 \cup \{(2.1)P(a) \vee \neg Q(a)^* \vee Q(b)^*\}$,
$O_3 = O_2 \cup \{(4)P(a)^*\}$
O_0 *is our starting set of clause instances and the others will be reached during the run. Then the following is a run of the TWFO calculus:*

$$
\begin{aligned}
&(\epsilon; \beta; O_0; \emptyset; \top) \\
\Rightarrow^{FactorizeWatched(1)}_{TWFO} \quad &(\epsilon; \beta; O_1; \emptyset; \top) \\
\Rightarrow^{DecLiteral(\neg P(a)^1)}_{TWFO} \quad &(\neg P(a)^1; \beta; O_1; \emptyset; \top) \\
\Rightarrow^{CreateInstance(2)}_{TWFO} \quad &(\neg P(a)^1; \beta; O_2; \emptyset; \top) \\
\Rightarrow^{DetectPropLit(1.1)}_{TWFO} \quad &(\neg P(a)^1; \beta; O_2; \{Q(a)^{P(a)\vee Q(a)\vee Q(a);\neg P(a)}\}; \top) \\
\Rightarrow^{PropLiteral(Q(a))}_{TWFO} \quad &(\neg P(a)^1 Q(a); \beta; O_2; \{Q(a)^{P(a)\vee Q(a)\vee Q(a);\neg P(a)}\}; \top) \\
\Rightarrow^{DetectPropLit(2.1)}_{TWFO} \quad &(\neg P(a)^1 Q(a); \beta; O_2; \{Q(a)^{P(a)\vee Q(a)\vee Q(a);\neg P(a)}, \\
&\quad Q(b)^{P(a)\vee \neg Q(a)\vee Q(b);Q(a)}\}; \top) \\
\Rightarrow^{DetectPropLit(3)}_{TWFO} \quad &(\neg P(a)^1 Q(a); \beta; O_2; \{Q(a)^{P(a)\vee Q(a)\vee Q(a);\neg P(a)}, \\
&\quad Q(b)^{P(a)\vee \neg Q(a)\vee Q(b);Q(a)}, \neg Q(b)^{\neg Q(a)\vee \neg Q(b);Q(a)}\}; \top) \\
\Rightarrow^{ConflictF}_{TWFO} \quad &(\neg P(a)^1 Q(a) Q(b); \beta; O_2; \{Q(a)^{P(a)\vee Q(a)\vee Q(a);\neg P(a)}, \\
&\quad Q(b)^{P(a)\vee \neg Q(a)\vee Q(b);Q(a)}, \neg Q(b)^{\neg Q(a)\vee \neg Q(b);Q(a)}\}; \neg Q(a) \vee \neg Q(b)) \\
\Rightarrow^{RemoveLiteral}_{TWFO} \quad &(\neg P(a)^1 Q(a); \beta; O_2; \{Q(a)^{P(a)\vee Q(a)\vee Q(a);\neg P(a)}, \\
&\quad Q(b)^{P(a)\vee \neg Q(a)\vee Q(b);Q(a)}, \neg Q(b)^{\neg Q(a)\vee \neg Q(b);Q(a)}\}; \neg Q(a) \vee \neg Q(b)) \\
\Rightarrow^{RemoveLiteral}_{TWFO} \quad &(\neg P(a)^1; \beta; O_2; \{Q(a)^{P(a)\vee Q(a)\vee Q(a);\neg P(a)}\}; \neg Q(a) \vee \neg Q(b)) \\
\Rightarrow^{RemoveLiteral}_{TWFO} \quad &(\epsilon; \beta; O_2; \emptyset; \neg Q(a) \vee \neg Q(b)) \\
\Rightarrow^{Backtrack(4)}_{TWFO} \quad &(\epsilon; \beta; O_3; \{P(a)^{P(a),\top}\}; \top)
\end{aligned}
$$

Here the notation for the rule Backtrack$(P(x) \vee Q(a) \vee P(y) \vee Q(y))$ *was shortened to Backtrack(4). To get the clause for backtracking we resolve* $\neg Q(a) \vee \neg Q(b) \cdot \emptyset$ *with* $P(x) \vee \neg Q(x) \vee Q(b) \cdot \{x \mapsto a\}$ *resulting in* $P(x) \vee \neg Q(x) \vee \neg Q(a) \cdot \{x \mapsto a\}$ *and then we factorize this to* $P(a) \vee \neg Q(a) \cdot \emptyset$. *Finally we resolve with* $P(x) \vee Q(a) \vee Q(x) \cdot \{x \mapsto a\}$ *and factorize again.*

Example 3 (Example run 3)**.** *Let the clause set* $N = \{P(a) \vee P(b) \vee Q(a) \vee R(x), P(a) \vee \neg P(x), P(a) \vee \neg Q(a)\}$ *be the set of starting clauses and and let* $\prec_B$ *be an ordering*

and β be a ground literal such that $L \prec_B \beta$ iff the predicate of L is P, Q, or R and $\text{const}(L) \subseteq \{a, b\}$. *Let O_0, O_1, and O_2 be sets of clause instances such that*
$O_0 = \{(1)P(a)^* \vee P(b)^* \vee Q(a) \vee R(x), (2)P(a)^* \vee \neg P(x)^*, (3)P(a)^* \vee \neg Q(a)^*\}$,
$O_1 = \{(1)P(a) \vee P(b)^* \vee Q(a)^* \vee R(x), (2)P(a)^* \vee \neg P(x)^*, (3)P(a)^* \vee \neg Q(a)^*\}$,
$O_2 = \{(1)P(a) \vee P(b) \vee Q(a)^* \vee R(x)^*, (2)P(a)^* \vee \neg P(x)^*, (3)P(a)^* \vee \neg Q(a)^*\}$
O_0 is our starting set of clause instances and the others will be reached during the run. The only difference between the three sets are the selected watched literals. Then the following is a run of the TWFO calculus:

$$
\begin{aligned}
&(\epsilon; \beta; O_0; \emptyset; \top) \\
\Rightarrow^{DecLiteral(\neg P(a)^1)}_{TWFO} \quad & (\neg P(a)^1; \beta; O_0; \emptyset; \top) \\
\Rightarrow^{UpdateWatched(1)}_{TWFO} \quad & (\neg P(a)^1; \beta; O_1; \emptyset; \top) \\
\Rightarrow^{DetectPropLit(2)}_{TWFO} \quad & (\neg P(a)^1; \beta; O_1; \{\neg P(x)^{P(a) \vee \neg P(x); \neg P(a)}\}; \top) \\
\Rightarrow^{DetectPropLit(3)}_{TWFO} \quad & (\neg P(a)^1; \beta; O_1; \{\neg P(x)^{P(a) \vee \neg P(x); \neg P(a)}, \neg Q(a)^{P(a) \vee \neg Q(a); \neg P(x)}\}; \top) \\
\Rightarrow^{PropLiteral(\neg P(b))}_{TWFO} \quad & (\neg P(a)^1 \neg P(b); \beta; O_1; \{\neg P(x)^{P(a) \vee \neg P(x); \neg P(a)}, \\
& \neg Q(a)^{P(a) \vee \neg Q(a); \neg P(x)}\}; \top) \\
\Rightarrow^{UpdateWatched(1)}_{TWFO} \quad & (\neg P(a)^1 \neg P(b); \beta; O_2; \{\neg P(x)^{P(a) \vee \neg P(x); \neg P(a)}, \\
& \neg Q(a)^{P(a) \vee \neg Q(a); \neg P(x)}\}; \top) \\
\Rightarrow^{PropLiteral(\neg Q(a))}_{TWFO} \quad & (\neg P(a)^1 \neg P(b) \neg Q(a); \beta; O_2; \{\neg P(x)^{P(a) \vee \neg P(x); \neg P(a)}, \\
& \neg Q(a)^{P(a) \vee \neg Q(a); \neg P(x)}\}; \top) \\
\Rightarrow^{DetectPropLit(1)}_{TWFO} \quad & (\neg P(a)^1 \neg P(b) \neg Q(a); \beta; O_2; \{\neg P(x)^{P(a) \vee \neg P(x); \neg P(a)}, \\
& \neg Q(a)^{P(a) \vee \neg Q(a); \neg P(x)}, R(x)^{P(a) \vee P(b) \vee Q(a) \vee R(x); \neg Q(a)}\}; \top)
\end{aligned}
$$

Note that the trail at the end can be extended to a satisfiable valuation as DetectPropLiteral rule was applied to every clause instance and there are no contradicting literals in F.

5. Efficient Detection of Propagations and Conflicts

There are multiple reasons why the original two-watched literals scheme for SAT solving is so efficient. The most important reason is that it only has to check a clause for propagations and conflicts if one of its watched literals was assigned false. This means whenever a new literal L has been added to the trail, it only has to check clauses where $\text{comp}(L)$ is a watched literal. For our two-watched literals scheme for first-order logic, we can prove similar properties that we assume will lead to an efficient implementation as in the SAT case.

Lemma 3 (Single Check). *Let* $(M;\beta;O_0;F_0;\top) \Rightarrow_{TWFO} (M;\beta;O_1;F_1;\top) \Rightarrow_{TWFO} \ldots \Rightarrow_{TWFO} (M;\beta;O_n;F_n;\top)$ *be a reasonable run starting from a consistent state, where* n *may be* 0 *and Forget(*V*) was not applied.*

1. *Then if none of the rules CreateInstance, UpdateWatched, FactorizeWatched, DetectPropLiteral, and Conflict are applicable to clause instance* $(C, L_1, L_2) \in O_i \cap O_j$ *in state* $(M;\beta;O_i;F_i;\top)$ *then they are also not applicable to* (C, L_1, L_2) *in state* $(M;\beta;O_j;F_j;\top)$ *with* $0 \leq i < j \leq n$. *(Hence, TWFO does not have to check a clause instance again after it was fully processed and before the trail has changed.)*
2. *CreateInstance is applied at most twice to a clause instance* C *with* $(C, L_1, L_2) \in O_i$.
3. *UpdateWatched is applied at most twice to a clause instance* C *with* $(C, L_1, L_2) \in O_i$ *independently of which literals are watched.*
4. *FactorizedWatched and DetectPropLiteral are applied at most once to a clause instance* $(C, L_1, L_2) \in O_i$.

The above lemma guarantees that TWFO only has to check each clause instance once before it changes the trail again. Moreover, for a fixed trail M the maximum number of rule applications per clause instance is at most 6. Note that this does not include the rule applications to clause instances that were derived from this one.

Lemma 4 (Empty Trail Exploration). *Let* $(\epsilon;\beta;O_0;F;\top)$ *be a consistent state. Then CreateInstance, UpdateWatched, DetectPropLiteral and Conflict are not applicable. (However, all clauses have to be checked for potential applications of FactorizeWatched.)*

The above lemma guarantees that only FactorizeWatched can be applied on the empty trail.

Lemma 5 (New Literal Exploration). *Let* $(M;\beta;O_0;F_0;\top)$ *be a consistent state such that CreateInstance, UpdateWatched, FactorizeWatched, DetectPropLiteral, and Conflict are not applicable in state* $(M;\beta;O_0;F;\top)$. *Let* $(M;\beta;O_0;F_0;\top) \Rightarrow_{TWFO} (ML;\beta;O_0;F_0;\top) \Rightarrow_{TWFO} (ML;\beta;O_1;F_1;\top) \Rightarrow_{TWFO} \ldots \Rightarrow_{TWFO} (ML;\beta;O_n;F_n;\top)$ *be a reasonable run (where* n *may be* 0*):*

1. *Then in state* $(ML;\beta;O_n;F_n;\top)$ *UpdateWatched is only applicable to clause instances* $(C, L_1, L_2) \in O_n$ *where* $\mathrm{comp}(L) = L_1$.
2. *Then in state* $(ML;\beta;O_n;F_n;\top)$ *CreateInstance, DetectPropLiteral, and Conflict are only applicable to clause instances* $(C, L_1, L_2) \in O_n$ *if there exists a substitution* σ *such that* $\mathrm{comp}(L) = L_1\sigma$ *or if* $(C, L_1, L_2) \notin O_0$, *i.e., the instance was added or its watched literals were modified after the trail was extended.*
3. *Then in state* $(ML;\beta;O_n;F_n;\top)$ *FactorizeWatched is only applicable to clause instances* $(C, L_1, L_2) \in O_n \setminus O_0$, *i.e., only to those clause instances that where modified or added after the trail was extended.*

The above lemma guarantees that after TWFO adds a new literal L to the trail, we only have to check those clause instances (C, L_1, L_2) for rule applications, where $\mathrm{comp}(L)$ is a ground instance of one of its watched literals (i.e., there exists σ and $i \in 1, 2$ so $L_i\sigma = L$) and those clause instances that were added or modified after L was added to the trail.

Lemma 6 (Backtrack Exploration)**.** *Let* $(MM'L;\beta;O_0;F'_0;\top)$ *be a consistent state. Let* $(MM'L;\beta;O_0;F'_0;\top) \Rightarrow_{TWFO}^{Conflict/ConflictF} (MM'L;\beta;O_0;F'_0;D) \Rightarrow_{TWFO}^{RemoveLiteral*} (M;\beta;O_n;F'_n;D) \Rightarrow_{TWFO}^{Backtrack(D')} (M;\beta;O_1;F_1;\top)$ *be a reasonable run from one application of Conflict/ConflictF to the first subsequent application of Backtrack(*D'*). Let* $(M;\beta;O_1;F_0;\top) \Rightarrow_{TWFO} (M;\beta;O_2;F_2;\top) \Rightarrow_{TWFO} \dots \Rightarrow_{TWFO} (M;\beta;O_n;F_n;\top)$ *be a reasonable run (where* n *may be* 1*) and Forget was never applied. Then in state* $(ML;\beta;O_n;F_n;\top)$ *UpdateWatched, FactorizeWatched, CreateInstance, DetectPropLiteral, and Conflict are only applicable to clause instances* $(C,L_1,L_2) \in O_n$ *where* $(C,L_1,L_2) \notin O_0$*.*

The above lemma guarantees that after TWFO applies Backtrack(D'), we only have to check those clause instances (C,L_1,L_2) for applications of the rules that were (i) either added by the last Backtrack (i.e., $C = D'$) or (ii) derived after Backtrack by an application of the rules UpdateWatched, FactorizeWatched, and CreateInstance.

6. Interaction between TWFO and SCL

The TWFO calculus is a calculus to help detect propagations and conflicts efficiently for the SCL calculus. So we have to keep certain aspects of the state of both calculi equal, these are the trail, the clauses and the state of the conflict clause, that is if it is $\top$ or not. So for all rule applications of SCL we have to show that we can simulate them with rule applications of TWFO. We do this only for a regular run of SCL and a reasonable run of TWFO.

A SCL-state has the form $(M;N;U;\beta;k;D)$ and a TWFO-state has the form $(M';\beta;O;F;D')$. We have to show that we can keep M and M' equal as well as that $D = \top$ iff $D' = \top$ and that for every $C \in N \cup U$ there are L_1 and L_2 such that $(C;L_1;L_2) \in O$.

The start state of SCL is $(\epsilon;N;\emptyset;\beta;0;\top)$ and then the start state of TWFO is $(\epsilon;\beta;O;F;\top)$ such that for every $C \in N \cup U$ there are L_1 and L_2 such that $(C;L_1;L_2) \in O$ holds.

So now we show for every rule application in SCL how we can simulate them with TWFO. First we look at the rule propagate:

$$(M;N;U;\beta;k;\top) \Rightarrow_{\text{SCL}}^{\text{Propagate}} (ML\sigma^{C\delta\cdot\sigma};N;U;\beta;k;\top)$$

We know that $L\sigma$ is undefined in M. Because we can propagte $L\sigma$ in TWFO, either DetectPropLiteral was already applied or can be applied after applications of CreateInstance, UpdateWatched and FactorizeWatched. This holds because DetectPropLiteral is complete, Theorem 1.3, and F always contains all literals that we can propagate from (factorized) unit clauses, Def 2.10. So we can do the following rule applications in TWFO:

$$\begin{aligned}
&(M;\beta;O;F;\top)\\
&(\Rightarrow_{\text{TWFO}}^{\text{CreateInstance, UpdateWatched, FactorizeWatched},*} && (M;\beta;O';F;\top)\\
&\Rightarrow_{\text{TWFO}}^{\text{DetectPropLiteral}} && (M;\beta;O';F \cup \{L\sigma^{C\sigma;K}\};\top))\\
&\Rightarrow_{\text{TWFO}}^{\text{PropLiteral}(L\sigma)} && (ML\sigma^{C\sigma};\beta;O';F';\top)
\end{aligned}$$

Note that the rules CreateInstance, UpdateWatched and FactorizeWatched do not add a clause instance $(C; L_1; L_2)$. So the rule Propagate can be simulated. Moreover, TWFO never adds a literal to F that cannot be propagated by Propagate, Theorem 1.8.

The next rule we look at is the Decide rule:

$$(M; N; U; \beta; k; \top) \Rightarrow_{\text{SCL}}^{\text{Decide}} (ML\sigma^{k+1}; N; U; \beta; k+1; \top)$$

We know that $L\sigma$ is undefined in M.

For DecLiteral to be applicable we need that there is no contradicting literal $L' \in F$ such that there is a τ where $\text{comp}(L\sigma) = L'\tau$. To show this we need that the SCL run is regular. We assume that there is such an $L'^C \in F$. From Definition 2.6 we know that $C = C_0 \vee L' \vee \cdots \vee L'$ and $M \models \neg C_0$. Because the SCL run is regular we know that there is no conflict before the application of Decide because Conflict is preferred over every other rule. In a regular run an application of Decide does not create a conflict so there is also no conflict after the application of Decide. So especially $M \not\models \neg C\tau$ and $ML\sigma \not\models \neg C\tau$ for all grounding substitutions τ. But because $M \models \neg C_0$ we can follow that $ML\sigma \not\models \neg L'\tau$ for all grounding substitutions τ. Therefore there is no τ such that $\text{comp}(L\sigma) = L'\tau$ contradicting our assumption.

So we can do the following rule application in TWFO:

$$(M; \beta; O; F; \top) \Rightarrow_{\text{TWFO}}^{\text{DecLiteral}(L\sigma^{k+1})} (ML\sigma^{k+1}; \beta; O; F; \top)$$

So the rule Decide can be simulated.

Now we look at the rule Conflict:

$$(M; N; U; \beta; k; \top) \Rightarrow_{\text{SCL}}^{\text{Conflict}} (M; N; U; \beta; k; D \cdot \sigma)$$

We know that $M \models \neg D\sigma$ holds. For TWFO we know from the completeness of the Conflict rule, Theorem 1.1, that Conflict either is applicable or will be applicable after applications of CreateInstance, UpdateWatched and FactorizeWatched. So we can do the following rule applications in TWFO:

$$\begin{aligned}
&(M; \beta; O; F; \top) \\
&\Rightarrow_{\text{TWFO}}^{\text{CreateInstance, UpdateWatched, FactorizeWatched},*} && (M; \beta; O'; F; \top) \\
&\Rightarrow_{\text{TWFO}}^{\text{Conflict}} && (M; \beta; O'; F; D\sigma)
\end{aligned}$$

So the rule Conflict can be simulated.

The next rules are Resolve and Factorize. Both these rules only change the conflict clause but before the rule application it holds that the conflict clause is not $\top$ and this does not change after the rule application. So we do not have to change anything in the state of TWFO.

Then we look at the rule Skip:

$$(ML; N; U; \beta; k; D \cdot \sigma) \Rightarrow_{\text{SCL}}^{\text{Skip}} (M; N; U; \beta; l; D \cdot \sigma)$$

We can just remove a literal with the rule RemoveLiteral in TWFO:

$$(ML; \beta; O; F; D') \Rightarrow_{\text{TWFO}}^{\text{RemoveLiteral}} (M; \beta; O'; F'; D')$$

So we can simulate the rule Skip.

Now we look at the rule Backtrack:

$$(MK^{i+1}M'; N; U; \beta; k; (D \vee L) \cdot \sigma) \Rightarrow_{\text{SCL}}^{\text{Backtrack}} (ML\sigma^{C \cdot \sigma}; N; U \cup \{D \vee L\}; \beta; i; \top)$$

In TWFO we first have to remove the literals with RemoveLiteral until the trail is M and then use Backtrack($D \vee L$) to add the new clause to O. Finally we have to apply CreateInstance, UpdateWatched and FactorizeWatched to apply DetectPropLiteral and then we can apply PropLiteral($L\sigma^{C\sigma}$).

$$\begin{aligned}
&(MK^{i+1}M'; \beta; O; F; D') \\
&\Rightarrow_{\text{TWFO}}^{\text{RemoveLiteral},|M'|+1} && (M; \beta; O^1; F^1; D') \\
&\Rightarrow_{\text{TWFO}}^{\text{Backtrack}(D \vee L)} && (M; \beta; O^2; F^2; \top) \\
&\Rightarrow_{\text{TWFO}}^{\text{CreateInstance, UpdateWatched, FactorizeWatched},*} && (M; \beta; O^3; F^2; \top) \\
&\Rightarrow_{\text{TWFO}}^{\text{DetectPropLiteral}} && (M; \beta; O^3; F^3; \top) \\
&\Rightarrow_{\text{TWFO}}^{\text{PropLiteral}(L\sigma^{(D \vee L)\sigma})} && (ML\sigma^{(D \vee L)\sigma}; \beta; O^3; F^3; \top)
\end{aligned}$$

Note that after the application of Backtrack($D \vee L$) it holds again that for every $C \in N \cup U$ there are L_1 and L_2 such that $(C; L_1; L_2) \in O$. Note also that Backtrack checks whether the new clause can be factorized to a unit and adds the resulting literal to F. So Backtrack can also be simulated.

The last rule that we have to look at is the Grow rule:

$$(M; N; U; \beta; k; \top) \Rightarrow_{\text{SCL}}^{\text{Backtrack}} (\epsilon; N; U; \beta'; 0; \top)$$

To simulate this rule we have to use the rule Forget($\emptyset$) to restart and then use Grow(β').

$$\begin{aligned}
&(M; \beta; O; F; \top) \\
&\Rightarrow_{\text{TWFO}}^{\text{Forget}(\emptyset)} && (\emptyset; \beta; O'; F'; \top) \\
&\Rightarrow_{\text{TWFO}}^{\text{Grow}(\beta')} && (\emptyset; \beta'; O'; F'; \top)
\end{aligned}$$

So Grow can also be simulated.

So all rule applications in the SCL calculus can be simulated with rule applications in the TWFO calculus and therefore the states can stay similar.

We also want to show how we can simulate an application of the rule ConflictF with rules in SCL.

$$(M; \beta; O; F \uplus \{L_1^{C_1}, L_2^{C_2}\}; \top) \Rightarrow_{\text{TWFO}}^{\text{ConflictF}} (ML_1\sigma^{C_1\sigma}; \beta; O; F \cup \{L_1^{C_1}, L_2^{C_2}\}; C_2\sigma)$$

We know that $L_1\sigma = \text{comp}(L_2\sigma)$ and $L_1\sigma, L_2\sigma \notin M$ hold. From Definition 2.6 we know that $C_1 = C_1' \vee L_1 \vee \cdots \vee L_1$ and $M \models \neg C_1'$ as well as $C_2 = C_2' \vee L_2 \vee \cdots \vee L_2$ and $M \models \neg C_2'$ hold. So $L_1\sigma^{C_1\sigma}$ can be propagated on the trail and because $L_1\sigma = \text{comp}(L_2\sigma)$ holds that $ML_1\sigma^{C_1\sigma} \models \neg C_2\sigma$ and therefore Conflict in the SCL calculus is applicable.

$$\begin{aligned}
&(M; N; U; \beta; k; \top) \\
&\Rightarrow_{\text{SCL}}^{\text{Propagate}} && (ML_1\sigma^{C_1\sigma}; N; U; \beta; k; \top) \\
&\Rightarrow_{\text{SCL}}^{\text{Conflict}} && (ML_1\sigma^{C_1\sigma}; N; U; \beta; k; C_2\sigma)
\end{aligned}$$

So the ConflictF rule can be simulated with rules in SCL. Note that the only other rule

in TWFO that is no rule to synchronize TWFO with SCL and that does affect elements that we have to keep consistent between the two calculi is the Conflict rule. But this can just easily be simulated with an application of the Conflict rule in SCL is a direct implication of the Soundness of the Conflict rule, Theorem 1.6.

We also must guarantee that for a reasonable strategy an application of the rule DecLiteral(L^k) does not create a conflict.

Lemma 7 (DecLiteral does not enable Conflict)**.** *In a reasonable run starting from a consistent start state* $(\emptyset; \beta; O; F; \top)$ *an application of the rule DecLiteral(L^k) does not enable an application of the rule Conflict.*

7. Conclusion

We have generalized the two-watched literal principle from propositional logic to a finite domain first-order logic (fixed β). The lifting is involved and we managed to keep the main properties of the principle: first, only watched literals need to be considered for trail changes, and, second, after backtracking no updates are needed except for the newly learned clause. We are confident that these properties will guarantee that an implementation of TWFO for first-order logic will be efficient in practice.

There is still some future work left before we can implement an efficient version of TWFO. First and foremost, we need to develop efficient indexing structures for TWFO. For instance, in the propositional case, we put all clauses with watched literal L into a vector `watched`$[L]$. This allows us to efficiently iterate through all clauses that need to be checked when the negation of watched literal comp(L) is added to the trail. However, when comp(L) is added to the trail in the first-order case, it is not enough to just check all clause instances C where L is watched. Instead, we have to check all clause instances C with a watched literal L' that can be instantiated to L (i.e., there exists τ with $L'\tau = L$). So we need an efficient data structure that finds all watched literals that can be instantiated to L. Work on indexing in first-order logic can here be a good starting point [7].

Although model building through trails has been considered in first-order logic, e.g. [8, 9, 10], we are not aware of any work adapting the propositional two-watched literal principle to this setting. This might also be due to the fact that exhaustive propagation, as done in propositional logic, cannot be afforded in more expressive logics [3] and therefore the two-watched literal scheme needs an additional laziness component, as suggested in this work.

References

[1] J. P. M. Silva, K. A. Sakallah, Grasp - a new search algorithm for satisfiability, in: International Conference on Computer Aided Design, ICCAD, IEEE Computer Society Press, 1996, pp. 220–227.

[2] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient sat solver, in: Design Automation Conference, 2001. Proceedings, ACM, 2001, pp. 530–535.

[3] A. Fiori, C. Weidenbach, Scl clause learning from simple models, in: P. Fontaine (Ed.), 27th International Conference on Automated Deduction, CADE-27, volume 11716 of *LNAI*, Springer, 2019, pp. 233–249.

[4] M. Bromberger, A. Fiori, C. Weidenbach, Deciding the bernays-schoenfinkel fragment over bounded difference constraints by simple clause learning over theories, in: F. Henglein, S. Shoham, Y. Vizel (Eds.), Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings, volume 12597 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 511–533.

[5] D. E. Knuth, P. B. Bendix, Simple word problems in universal algebras, in: I. Leech (Ed.), Computational Problems in Abstract Algebra, Pergamon Press, 1970, pp. 263–297.

[6] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021.

[7] R. Nieuwenhuis, T. Hillenbrand, A. Riazanov, A. Voronkov, On the evaluation of indexing techniques for theorem proving, in: R. Goré, A. Leitsch, T. Nipkow (Eds.), Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings, volume 2083 of *LNCS*, Springer, 2001, pp. 257–271.

[8] P. Baumgartner, A. Fuchs, C. Tinelli, Lemma learning in the model evolution calculus, in: LPAR, volume 4246 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 572–586.

[9] K. Korovin, Inst-gen - A modular approach to instantiation-based automated reasoning, in: A. Voronkov, C. Weidenbach (Eds.), Programming Logics - Essays in Memory of Harald Ganzinger, volume 7797 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 239–270.

[10] M. P. Bonacina, U. Furbach, V. Sofronie-Stokkermans, On first-order model-based reasoning, in: N. Martí-Oliet, P. C. Ölveczky, C. L. Talcott (Eds.), Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday, volume 9200 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 181–204.