Quantifier-Free Equational Logic and Prime Implicate Generation - Proofs

Mnacho Echenim[1,2], Nicolas Peltier[1,4] and Sophie Tourret[1,3]

[1] Grenoble Informatics Laboratory
[2] Grenoble INP - Ensimag
[3] Université Grenoble 1
[4] CNRS

This technical report completes the article submitted to CADE with the proofs of all theorems and propositions.

1 Clauses with Uninterpreted Functions in Equational Logic

1.1 Basic Definitions

The theory of equational logic with uninterpreted functions will be denoted by EUF throughout the paper. Let Σ be a *signature* such that $\Sigma = \bigcup_{n=0}^{\infty} \Sigma_n$. The signature Σ_0 is the signature of constant symbols, usually denoted by a, b, and in general Σ_n is the signature of function symbols with arity n, usually denoted by f, g. The notation $\mathfrak{T}(\Sigma)$ stands for the set of well-formed ground terms over Σ, defined as usual, most often denoted by s, t, u, v, w. A well founded reduction order $\prec$ on $\mathfrak{T}(\Sigma)$ is assumed given (e.g. KBO [3]). For any term $s \in \mathfrak{T}(\Sigma)$, $\mathrm{Pos}(s)$ is the set of all positions in s, for example $\mathrm{Pos}(f(a, g(b))) = \{\epsilon, 1, 2, 2.1\}$. The expression $\mathrm{head}(s)$ represents the symbol of Σ appearing in s at position ϵ. Generally, any subterm of a term t at position p is accessed with $t_{|p}$.

A *literal*, usually denoted by l or m, is either an equation (or *atom*, or *positive* literal) $s \simeq t$, or an inequation $s \not\simeq t$ (or *negative* literal). The literal written $s \bowtie t$ can denote either the equation or the inequation between s and t. The literal l^c stands for $s \not\simeq t$ (resp. $s \simeq t$) when l is $s \simeq t$ (resp. $s \not\simeq t$). A literal of the form $s \not\simeq s$ is called a *contradictory* literal (or a contradiction) and a literal of the form $s \simeq s$ is a *tautological* literal (or a tautology).

We consider clauses as disjunctions (or multisets) of literals and *formulæ* as collections of clauses. If C is a clause and l a literal, $C \backslash l$ denotes the clause C where *all* occurrences of l have been removed (up to commutativity of equality). For any clause C, the notation C^+ (resp. C^-) denotes the set of positive (resp. negative) literals in C.

In Sect. 2, we also consider conjunctions of literals. If $\mathcal{X} = \bigwedge_{i=1}^n l_i$ then $\mathcal{X}^c$ denotes the clause $\bigvee_{i=1}^n l_i^c$. Similarly, if $C = \bigvee_{i=1}^n l_i$ then $\neg C \overset{\text{def}}{=} \bigwedge_{i=1}^n l_i^c$. We often identify sets of clauses with conjunctions, e.g., considering a set of clauses S, we may write $S \cup \bigwedge_{i=1}^n l_i$ for $S \cup \{l_i \mid i \in [1, n]\}$, and instead of $\{l_1, \ldots, l_n\} \subseteq \{l'_1, \ldots, l'_m\}$, we may write $\bigwedge_{i=1}^n l_i \subseteq \bigwedge_{i=1}^m l'_i$.

We define an *equational interpretation* $\mathcal{I}$ as a congruence relation on $\mathfrak{T}(\Sigma)$. In other words, $\mathcal{I}$ is an equivalence relation on $\mathfrak{T}(\Sigma)$ verifying the following property: $\forall f \in \Sigma_n, \big(\forall i \in \{1..n\}, s_i =_{\mathcal{I}} t_i \Rightarrow f(s_1,..,s_n) =_{\mathcal{I}} f(t_1,..,t_n)\big)$, where $s =_{\mathcal{I}} t$ means that the terms s and t belong to the same class in $\mathcal{I}$. A positive literal $l = s \simeq t$ is evaluated to $\top$ (true) in $\mathcal{I}$, written $\mathcal{I} \models l$, if $s =_{\mathcal{I}} t$; otherwise l is evaluated to $\bot$ (false). A negative literal $l = s \not\simeq t$ is evaluated to $\top$ in $\mathcal{I}$ if $s \neq_{\mathcal{I}} t$, and to $\bot$ otherwise. This evaluation is extended to clauses and sets of clauses in the usual way. An interpretation that evaluates C to $\top$ is a *model* of C (often written $\mathcal{M}$ in this paper). A *tautological clause* (or *tautology*) is a clause for which all equational interpretations are models and a *contradiction* is a clause that has no model.

1.2 Entailment of Clauses

Definition 1 Let C be a clause, we define for any term s the C-*equivalence class* of s as:

$$[s]_C = \{t \in \mathfrak{T}(\Sigma) \mid \neg C \models s \simeq t\}.$$

The corresponding equivalence relation is written $\equiv_C$. The C-*representative* of a term s, a literal l and a clause D are respectively defined by:

$$s_{\downharpoonleft C} \stackrel{\text{def}}{=} \min_{\prec}([s]_C), \quad l_{\downharpoonleft C} \stackrel{\text{def}}{=} s_{\downharpoonleft C} \bowtie t_{\downharpoonleft C}, \text{ for } l = s \bowtie t, \quad \text{and } D_{\downharpoonleft C} \stackrel{\text{def}}{=} \{l_{\downharpoonleft C} \mid l \in D\}$$

$\Diamond$

From the previous definition, the following results can be directly inferred:

Proposition 2 *Let s be a term, l be a literal and C and D be two clauses, then:*

$$\neg C \models s \simeq s_{\downharpoonleft C}, \quad \neg C \models l \Leftrightarrow l_{\downharpoonleft C}, \quad \textit{and } \neg C \models D \Leftrightarrow D_{\downharpoonleft C}.$$

where $\Leftrightarrow$ is the usual logical equivalence.

Proposition 3 *Let C be a non-tautological clause and l be a literal. Then $l \models C \vee l_{\downharpoonleft C}$.*

Proof. Let $\mathcal{M}$ be a model of l. If $\mathcal{M} \models C$ then the result clearly holds. Otherwise $\mathcal{M} \models \neg C$ and by Proposition 2, $\neg C \models l \Leftrightarrow l_{\downharpoonleft C}$, thus $\mathcal{M} \models l_{\downharpoonleft C}$. ∎

Definition 4 The two following orders on literals are used throughout the paper.

1. The total order $\prec$ on terms is extended in the usual way to literals and then to clauses, by considering that a negative literal $t \not\simeq s$ is a set $\{\{t, s\}\}$ and that a positive literal $s \simeq t$ is $\{\{t\}, \{s\}\}$.
2. The total order $<_\pi$ on literals is defined as follows:
   – the equations are all greater than the inequations;

– for l_1 and l_2 literals with the same polarity, $l_1 <_\pi l_2$ iff $l_1 \prec l_2$.

Both orders are relaxed (into $\preceq$ and $\leq_\pi$ resp.) by also accepting equal literals or clauses. $\diamondsuit$

Example 5 Let $C = g(a) \not\simeq b \vee c \simeq d$ and $D = a \not\simeq b \vee f(c) \simeq d$, with $a \prec b \prec c \prec d \prec f(c) \prec g(a)$. We have $D \prec C$ and $C <_\pi D$, because on the one hand $a \not\simeq b \prec c \simeq d \prec f(c) \simeq d \prec g(a) \not\simeq b$, and on the other hand $a \not\simeq b <_\pi g(a) \not\simeq b <_\pi c \simeq d <_\pi f(c) \simeq d$. $\clubsuit$

The order $\prec$ is used, as is usual, to determine which implicates are prime and which are redundant (cf. definition below). The order $<_\pi$ is necessary for handling clauses as presented in Sect. 3, but is not used outside this scope.

In propositional logic, testing entailment for redundancy detection amounts to a simple inclusion test [2] but things are more complex in EUF because the axioms of transitivity and substitutivity must be taken into account. For example, the clause $e \not\simeq b \vee b \not\simeq c \vee f(a) \simeq f(b)$ is redundant w.r.t. the clause $e \not\simeq c \vee a \simeq c$ because of these axioms.

The following proposition and theorem describe the *projection* method for testing entailment in a syntactic way.

Proposition 6 *Let C be a clause, and s, t be two terms. If there is no positive literal l in C such that $l_{\downharpoonleft C \vee s \not\simeq t}$ is tautological, then $C \vee s \not\simeq t$ is not a tautology.*

Proof. Let $D = C \vee t \not\simeq s$. Let $\mathcal{I} = \equiv_C$ and $\mathcal{J} = \equiv_D$. We show that $\mathcal{J}$ is a counter-model of D. Note that in particular, if $s =_\mathcal{I} t$, then $\mathcal{I} = \mathcal{J}$. $\mathcal{J} \not\models s \not\simeq t$ since $s =_\mathcal{J} t$ and for any negative literal $u \not\simeq v \in C$, by definition $u \not\simeq v \in D$, thus $u =_\mathcal{J} v$ and finally $\mathcal{J} \not\models u \not\simeq v$. For the positive literals of C, let $u \simeq v \in C$, and assume $\mathcal{J} \models u \simeq v$. Under this assumption, $u \not\simeq v \models D$, hence $u_{\downharpoonleft D} = v_{\downharpoonleft D}$, contradicting the hypothesis about the positive literals of C. Thus $\mathcal{J} \not\models u \simeq v$. $\blacksquare$

Theorem 7 *Let C and D be two non-tautological clauses. The relation $D \models C$ holds iff for every negative literal l in D, the literal $l_{\downharpoonleft C}$ is a contradiction and for every positive literal l in D, there exists a positive literal m in C such that $m_{\downharpoonleft C \vee l^c}$ is tautological.*

Proof. First assume that $D \models C$. Consider a negative literal $s \not\simeq t \in D$. Then we must have $s \not\simeq t \models C$, thus $t_{\downharpoonleft C} = s_{\downharpoonleft C}$, and the corresponding literal in $D_{\downharpoonleft C}$ is a contradiction. Now consider a positive literal $s \simeq t \in D$. If there is no positive literal m in C such that $m_{\downharpoonleft C \vee s \not\simeq t}$ is a tautology then $C \vee s \not\simeq t$ is not a tautology by Proposition 6. But $D \vee s \not\simeq t \models C \vee s \not\simeq t$, and $D \vee s \not\simeq t$ is a tautology, which yields a contradiction.

For the converse implication, we prove that every literal in D entails C. Let $s \not\simeq t \in D$, then by hypothesis we have $s_{\downharpoonleft C} = t_{\downharpoonleft C}$, thus $[s]_C = [t]_C$, and by definition $s \not\simeq t \models C$. Let $s \simeq t \in D$, then by hypothesis there is a literal $u \simeq v \in C$ such that $u_{\downharpoonleft C \vee s \not\simeq t} = v_{\downharpoonleft C \vee s \not\simeq t}$. It follows that $u \not\simeq v \models C \vee s \not\simeq t$, which is equivalent to $s \simeq t \models C \vee u \simeq v$. Since $u \simeq v \in C$, we conclude that $s \simeq t \models C$. $\blacksquare$

Example 8 Given the order $a \prec b \prec c \prec e \prec f(a) \prec f(b)$ on terms, let $C = e \not\simeq c \vee a \simeq c$ and $D = e \not\simeq b \vee b \not\simeq c \vee f(a) \simeq f(b)$ and let $l = e \not\simeq c$ and $m = a \simeq c$ be the literals of C. We have, $l_{\downharpoonright D} = b \not\simeq b$ because $[b]_D = \{b, c, e\}$ and $\min\limits^{\prec}([b]_D) = b$. Moreover the literal $f(a) \simeq f(b) \in D$ is such that $(f(a) \simeq f(b))_{\downharpoonright D \vee m^c} = f(a) \simeq f(a)$, which, by Th. 7, proves that D is redundant w.r.t. C. ♣

1.3 Clausal Normal Form

In order to avoid having to handle large numbers of equivalent clauses, we define a clausal normal form that is unique up to equivalence.

Definition 9 A non-tautological clause C is in *normal form* if:

1. every negative literal l in C is such that $l_{\downharpoonright C \setminus l} = l$.
2. every literal $t \simeq s \in C$ is such that $t = t_{\downharpoonright C}$ and $s = s_{\downharpoonright C}$;
3. there are no two distinct positive literals l, m in C such that $m_{\downharpoonright l^c \vee C^-}$ is a tautology;
4. C contains no literal of the form $t \not\simeq t$ or $t \simeq t$;
5. the literals in C occur exactly once in C;

The normal form equivalent to C is denoted by $C_{\downarrow}$. ◊

Remark 10 *In our last articles on prime implicate generation [4], the focus was on strictly flat clauses. For the sake of handling non-flat clauses, the clausal normal form (see e.g. [5], Def. 4) had to be extended. The differences lie with points 1 and 3 of Def. 9. They strengthen the requirements on negative and positive literals resp. to cover the non-flat ones.*

For example, using the same term ordering as in Ex. 8, the clause $c \not\simeq b \vee e \not\simeq b \vee f(b) \simeq f(a)$ is the normal form of the clauses $c \not\simeq b \vee e \not\simeq b \vee f(c) \simeq f(a)$, $c \not\simeq b \vee e \not\simeq b \vee f(e) \simeq f(a)$, $c \not\simeq e \vee e \not\simeq b \vee f(b) \simeq f(a)$, etc...

To prove the uniqueness of the clausal normal form, a rewriting system [1] is associated with each clause in normal form.

Definition 11 Let C be a clause in normal form. The rewriting system R_C associated to C is such that:

$$t \to s \in R_C \text{ iff } t \simeq s \in C \text{ with } t \succ s$$ ◊

Due to the definition of a clausal normal form, the associated rewriting systems are always convergent. We denote by $t_{\downharpoonright R_C}$ the term $t \in \mathfrak{T}(\Sigma)$ on which all possible rewriting rules from a rewrite system R_C have been applied. Note that $t_{\downharpoonright R_C} = t_{\downharpoonright C}$.

Proposition 12 *Let C be a clause in normal form and t be a term in $\mathfrak{T}(\Sigma)$. The term $t_{\downharpoonright R_C}$ is obtained from t by using only the rules of R_C that have a left-hand side smaller or equal to t.*

Proof. To compute $t_{\downharpoonright R_C}$, rewriting rules can be applied only on t or its sub-terms. All the sub-terms of t are smaller than t and by definition of R_C a rewriting always replaces a term by a smaller one. ■

Theorem 13 *The normal form of a non-tautological clause C is the smallest clause equivalent to C.*

Proof. We first prove the uniqueness of the normal form by considering the negative literals in a clause and then the positive ones. Consider two equivalent clauses C_1 and C_2 that are in normal form, so that $\equiv_{C_1} = \equiv_{C_2}$.

By Definition 11, C_1 and C_2 are associated to rewriting systems respectively denoted by R_1 and R_2. If $C_1^- \neq C_2^-$ then $R_1 \neq R_2$. Let us consider the smallest rule $t \to s$ with $t \succ s$ that does not appear in both rewriting systems, w.l.o.g. we assume $t \to s \in R_1$ and $t \to s \notin R_2$. Since C_1 and C_2 are equivalent and $t_{\downharpoonright R_1} = s$, we also have $t_{\downharpoonright R_2} = s$. By Proposition 12, the left-hand side of the rules used to rewrite t are smaller or equal to t. Note that since t is at least rewritten into s in R_1, there must be at least one rule in R_2 that can be applied to t.

– If in R_2, the rules applicable to t are all smaller than $t \to s$, then these rules also appear in R_1, making $t \to s$ redundant in R_1 and $t \not\simeq s$ redundant in C_1, thus contradicting the definition of the normal form (Def. 9, point 6).
– Otherwise there is a rule $t \to v \in R_2$, with $t \succ v \succ s$, and $t \to v \notin R_1$ by definition of the normal form. This same definition ensures that $v = s$, which is impossible under the current hypotheses.

We now consider the case of the positive literals. The equivalence of C_1 and C_2 allows us to invoke Theorem 7 in both directions. Let l_1 be a positive literal in C_1. There exists a literal l_2 in C_2 such that $l_{2\downharpoonright l_1^c \vee C_2^-}$ is a tautology, hence $l_1 \models C_2^- \vee l_2$. Similarly for l_2, there is a positive literal m_1 in C_1 such that $m_{1\downharpoonright l_2^c \vee C_1^-}$, so that $l_2 \models C_1^- \vee m_1$. By combining the two entailment relations, given that $C_1^- \equiv C_2^-$, we deduce that $l_1 \models C_1^- \vee m_1$, i.e. $m_1^c \models C_1^- \vee l_1^c$, thus $m_{1\downharpoonright C_1^- \vee l_1^c}$ is a tautology. This contradicts point 5 of the normal form definition, unless $m_1 = l_1$. Since this property is symmetric for C_1 and C_2, it follows that $C_1^- \vee l_1 \equiv C_2^- \vee l_2$, hence $l_{1\downharpoonright C_1} \equiv l_{2\downharpoonright C_2}$ and so $l_{1\downharpoonright C_1} = l_{2\downharpoonright C_2}$. In addition C_1 and C_2 are in normal form, thus $l_1 = l_2$ and by symmetry this result can be generalized to $C_1^+ = C_2^+$.

As for the proof of minimality, it stems directly from the definition of the normal form. Consider a clause C that is not in normal form. One of the points of the definition must be contradicted.

– If C contains a negative literal $t \not\simeq s$ with $s \prec t$ and such that $s \neq t_{\downharpoonright C}$, it is greater than the equivalent $s \not\simeq t_{\downharpoonright C} \vee t \not\simeq t_{\downharpoonright C}$ (which may not even appear in the normal form of C if it is implied by other negative literals of this clause).
– If C contains a positive literal $u \simeq v$ such that $u \neq u_{\downharpoonright C}$ or $v \neq v_{\downharpoonright C}$ (or both) then replacing this literal with $u_{\downharpoonright C} \simeq v_{\downharpoonright C}$ yields a smaller equivalent clause.
– No literal $t \simeq t$ can be present in a non-tautological clause and removing literals of the form $t \not\simeq t$ yields a smaller equivalent clause.

– The two last criteria guaranty the absence of redundant literals without which a smaller equivalent clause is also generated. ■

2 Implicate Generation

The results presented in this section are a direct extension of those of [4] (Sect. 2 & 3). Before introducing the calculus used for generating the implicates we present the notion of a constrained clause.

Definition 14 A *constraint* is a (possibly empty) conjunction (or set) of literals. A *constrained clause* (or *c-clause*) is a pair $[C \,|\, \mathcal{X}]$ where C is a clause and $\mathcal{X}$ is a constraint. $\Diamond$

Empty constraints are denoted by $\top$. $[C \mid \top]$ is often written simply as C and referred to as a standard clause. A constraint is *normalized*, or in *normal form*, if the clause $\mathcal{X}^c$ is in normal form. Note that only non-contradictory constraints can be normalized.

From a semantic point of view, a constrained clause $[C \,|\, \mathcal{X}]$ is equivalent to the standard clause $\mathcal{X}^c \vee C$. For example the c-clause $[c \simeq b \,|\, f(a) \simeq c \wedge c \not\simeq d]$ is equivalent to $c \simeq b \vee f(a) \not\simeq c \vee c \simeq d$. Intuitively, the intended meaning of a c-clause $[C \mid \mathcal{X}]$ is that the clause C can be inferred provided the literals in $\mathcal{X}$ are added as axioms to the considered clause set.

The usual notion of redundancy is extended to c-clauses.

Definition 15 A c-clause $[C \,|\, \mathcal{X}]$ is *redundant* w.r.t. a set of c-clauses S if either $\mathcal{X}$ is unsatisfiable or there exist c-clauses $[D_i \mid \mathcal{Y}_i] \in S$ $(1 \leq i \leq n)$ such that $\forall i \in \{1 \ldots n\}\, D_i \preceq C$ and $\mathcal{Y}_i \subseteq \mathcal{X}$, and $\mathcal{X}', D_1, \ldots, D_n \models C$, where $\mathcal{X}'$ denotes the set of literals in $\mathcal{X}$ that are smaller than C. $\Diamond$

It is now possible to extend the usual superposition calculus [7] to a constrained superposition calculus denoted c$\mathcal{SP}$, that is able to generate all prime implicates of a formula up to redundancy.

This calculus is composed of the standard superposition rules trivially extended to constrained clauses (Table 1) along with two assertion rules (Table 2). As usual the calculus is parametrized by the ordering $\succ$ on terms and a selection function sel, where s$el(C)$ contains all maximal literals in C or (at least) one negative literal. A literal is *selected* in C if it occurs in s$el(C)$. We assume that the clausal part of c-clauses is systematically normalized (which explains the absence of reflexivity from the standard rules of Table 1). Note, however, that the constraint part is *not* normalized.

The principle of c$\mathcal{SP}$ is to generate the implicates of a formula as constraints of the empty clause. The standard inference rules are used to refute the clausal part of c-clauses, while the assertion rules explore the possible implicates by making hypotheses about their literals, that are stored in the constraint part of the c-clauses. Since only the c-clauses with a refutable clausal part are of interest, the assertion rules apply only on the literals that render a new superposition

Superposition	$\dfrac{[r \simeq l \vee C \,\vert\, \mathcal{X}] \quad [u \bowtie v \vee D \,\vert\, \mathcal{Y}]}{[u[l] \bowtie v \vee C \vee D \,\vert\, \mathcal{X} \wedge \mathcal{Y}]}$	If $u_{\vert p} = r$, $r \succ l$, $u \succ v$, and $(r \simeq l)$ and $(u \bowtie v)$ are selected in $(r \simeq l \vee C)$ and $(u \bowtie v \vee D)$ respectively.
Factoring	$\dfrac{[t \simeq u \vee t \simeq v \vee C \,\vert\, \mathcal{X}]}{[t \simeq v \vee u \not\simeq v \vee C \,\vert\, \mathcal{X}]}$	If $t \succ u$, $t \succ v$ and $(t \simeq u)$ is selected in $t \simeq u \vee t \simeq v \vee C$.

Table 1: Standard Inference Rules

possible (into the clause to which the rule applies for the Pos. Assert. rule and into the asserted literal for the Neg. Assert. rule). In other words, these rules use the fact that $S \models C$ iff $S \wedge \neg C \models \Box$ to build implicates literal by literal.

Example 16 The following example shows how to derive the implicate $a \not\simeq d \vee f(c) \simeq f(b)$ from $\{a \simeq b, f(c) \simeq f(d)\}$, given the term ordering $a \prec b \prec c \prec d \prec f(a) \prec f(b) \prec f(c) \prec f(d)$.

1 $[f(c) \simeq f(d) \,\vert\, \top]$	(hyp)
2 $[f(c) \simeq f(a) \,\vert\, a \simeq d]$	(Pos. AR, 1)
3 $[f(a) \not\simeq f(b) \,\vert\, a \simeq d \wedge f(c) \not\simeq f(b)]$	(Neg. AR, 2)
4 $[a \simeq b \,\vert\, \top]$	(hyp)
5 $[f(a) \not\simeq f(a) \,\vert\, a \simeq d \wedge f(c) \not\simeq f(b)]$	(Sup. 3, 4)
6 $[\Box \,\vert\, a \simeq d \wedge f(c) \not\simeq f(b)]$	(Ref. 5)

The negation of $a \simeq d \wedge f(c) \not\simeq f(b)$ is the desired implicate. ♣

Positive Assertion	$\dfrac{[u \bowtie v \vee C \,\vert\, \mathcal{X}]}{[u[s] \bowtie v \vee C \,\vert\, \mathcal{X} \wedge t \simeq s]}$	If $u_{\vert p} = t$, $t \succ s$, $u \succ v$ and $(u \bowtie v)$ is selected in $(u \bowtie v \vee C)$.
Negative Assertion	$\dfrac{[t \simeq s \vee C \,\vert\, \mathcal{X}]}{[u[s] \bowtie v \vee C \,\vert\, \mathcal{X} \wedge u \bowtie v]}$	If $u_{\vert p} = t$, $t \succ s$, $u \succ v$, and $(t \simeq s)$ is selected in $(t \simeq s \vee C)$.

Table 2: Assertion Rules

Theorem 17 *$c\mathcal{SP}$ is sound and deductive complete.*

Lemma 18 *Let $[C \mid \mathcal{X}]$ be a c-clause derived in an arbitrary number of steps from n premises $[D_i \,\vert\, \mathcal{Y}_i]$ with $i \in \{1 \ldots n\}$. Then C is a logical consequence of $D_1, \ldots, D_n, \mathcal{X}$ and for all i, $\mathcal{Y}_i \subseteq \mathcal{X}$.*

Proof. It is easy to verify that this property holds for each inference rule, the result follows by a straightforward induction on the length of the derivation. ■

Lemma 18 permits to deduce the soundness of the calculus:

Corollary 19 *For any c-clause $[C \,|\, \mathcal{X}]$ deducible from a set of standard clauses S, C is a logical consequence of $S \cup \mathcal{X}$. In particular, if $C = \Box$ then $S \models \mathcal{X}^c$.*

We now prove that the calculus is deductive-complete, i.e., that it permits to generate every prime implicate of a given set of clauses.

Definition 20 For every set of c-clauses S and for every constraint $\mathcal{X}$, we denote by $S|_{\mathcal{X}}$ the set of standard clauses D such that $[D \,|\, \mathcal{Y}] \in S$ and $\mathcal{Y} \subseteq \mathcal{X}$. ♢

Proposition 21 *Let S be a set of c-clauses and let $\mathcal{X}$ be a satisfiable constraint. If a c-clause $[C \,|\, \mathcal{Y}]$ is redundant in S and $\mathcal{Y} \subseteq \mathcal{X}$, then C is redundant in $S|_{\mathcal{X}} \cup \mathcal{X}$.*

Proof. By definition of c-clause redundancy, there are two cases to consider.

- The first condition leading to redundancy is that $\mathcal{Y}$ is unsatisfiable. In this case, since $\mathcal{Y}$ is a conjunction of literals and $\mathcal{Y} \subseteq \mathcal{X}$, the constraint $\mathcal{X}$ is also unsatisfiable.
- In the second case, there exist n c-clauses $[D_i \mid \mathcal{Y}_i] \in S$ $(1 \leq i \leq n)$ such that $\forall i \in [1,n]\, C \succeq D_i$, $\forall i \in [1,n]\, \mathcal{Y}_i \subseteq \mathcal{Y}$ and $\mathcal{Y}', D_1, \dots, D_n \models C$, where $\mathcal{Y}'$ denotes the set of literals in $\mathcal{Y}$ that are smaller than C. Since $\mathcal{Y} \subseteq \mathcal{X}$ we deduce that $\forall i \in [1,n]\, \mathcal{Y}_i \subseteq \mathcal{X}$, hence $\forall i \in [1,n]\, D_i \in S|_{\mathcal{X}}$. Since $\mathcal{Y}', D_1, \dots, D_n \models C$, $\mathcal{X}', D_1, \dots, D_n \preceq C$ and $\mathcal{Y}' \cup \{D_1, \dots, D_n\} \subseteq S|_{\mathcal{X}} \cup \mathcal{X}$, C is redundant in $S|_{\mathcal{X}} \cup \mathcal{X}$. ■

Definition 22 A set of c-clauses S is *saturated w.r.t. a constraint* $\mathcal{X}$ if every c-clause $[C \,|\, \mathcal{Y}]$ such that $\mathcal{Y} \subseteq \mathcal{X}$ that is deducible from S by applying one of the inference rules once is redundant w.r.t. S. ♢

Theorem 23 *Let $\mathcal{X}$ be a normalized satisfiable constraint. Let S be a set of standard clauses and $S^\star$ be the set obtained by saturating S with $c\mathcal{SP}$. If $S^\star$ is saturated w.r.t. $\mathcal{X}$ and $S \models \mathcal{X}^c$, then there exists a constraint $\mathcal{Y} \subseteq \mathcal{X}$ such that $[\Box \,|\, \mathcal{Y}] \in S^\star$.*

Remark 24 *Note that considering only normalized constraints is not restrictive since any constraint is equivalent to a normalized one. Moreover our goal is to eventually generate implicates that are in normal form, thus all c-clauses whose constraints are not in normal form can be discarded (since these constraints occur in all the descendants). This strategy strongly restricts the search space. For instance no rule will apply on $[a \simeq b \,|\, c \simeq d]$ and $[anot \simeq b \,|\, c \simeq e]$ because the obtained constraint $c \simeq d \wedge c \simeq e$ is not in normal form. Note also that the handling of clauses and constraints differ: we* normalize *the clausal part of the c-clause, whereas we merely* check *that the constraint is in normal form.*

PROOF. Let $S' = S^\star|_{\mathcal{X}} \cup \mathcal{X}$. We first remark that S' is unsatisfiable. Indeed, $S^\star|_\top \models S$ since by Proposition 21 all the standard clauses that are removed from S during the saturation process must be redundant in $S^\star|_\top$; furthermore, $S^\star|_\top \subseteq S^\star|_{\mathcal{X}}$, so that $S' \models S^\star|_\top \cup \mathcal{X} \models S \cup \mathcal{X} \models \mathcal{X}^c \cup \mathcal{X}$. We now prove that S' is saturated (in the standard way [7]). We only consider the case where the Superposition rule is applied, the proof for the other rules is similar. Let $r \simeq l \vee P_1$ and $u \bowtie v \vee P_2$ be two clauses occurring in S', with $r = u_{|p}$, $l \prec r$, $v \prec u$ and assume that $r \simeq l$ and $u \bowtie v$ are selected in $r \simeq l \vee P_1$ and $u \bowtie v \vee P_2$ respectively. Let $u[l] \bowtie v \vee P_1 \vee P_2$ be the clause deduced by (standard) superposition from the two clauses introduced previously. Several cases are distinguished:

- If both $r \simeq l \vee P_1$ and $u \bowtie v \vee P_2$ occur in $S^\star|_{\mathcal{X}}$, then S contains two c-clauses of the form $[r \simeq l \vee P_1 \,|\, \mathcal{X}_1]$ and $[u \bowtie v \vee P_2 \,|\, \mathcal{X}_2]$ with $\mathcal{X}_1, \mathcal{X}_2 \subseteq \mathcal{X}$. It is clear that the superposition rule of c$\mathcal{SP}$ applies to these c-clauses, yielding $[u[l] \bowtie v \vee P_1 \vee P_2 \,|\, \mathcal{X}_1 \wedge \mathcal{X}_2]$. Since $S^\star$ is saturated w.r.t. $\mathcal{X}$, this c-clause is redundant w.r.t. $S^\star$, and since $\mathcal{X}_1 \wedge \mathcal{X}_2 \subseteq \mathcal{X}$, we deduce by Prop. 21 that $u[l] \bowtie v \vee P_1 \vee P_2$ is redundant w.r.t. S'.
- If $r \simeq l \vee P_1$ occurs in $S^\star|_{\mathcal{X}}$ and $u \bowtie v \vee P_2$ occurs in $\mathcal{X}$, then by definition, P_2 must be empty and $S^\star$ contains a c-clause of the form $[r \simeq l \vee P_1 \,|\, \mathcal{X}_1]$ with $\mathcal{X}_1 \subseteq \mathcal{X}$. Assume first that $\bowtie = \not\simeq$. Then the Neg. Assert. rule applies on the c-clause, yielding $[u[l] \not\simeq v \vee P_1 \,|\, \mathcal{X}_1 \wedge u \not\simeq v]$. Since $u \not\simeq v \in \mathcal{X}$ and $\mathcal{X}_1 \subseteq \mathcal{X}$, this c-clause is redundant in S, and Prop. 21 permits to deduce that $u[l] \not\simeq v \vee P_1$ is redundant in $S^\star|_{\mathcal{X}}$. If $\bowtie = \simeq$ then the Neg. Assert. rule applies on $[r \simeq l \vee P_1 \,|\, \mathcal{X}_1]$ yielding $[u[l] \simeq v \vee P_1 \,|\, \mathcal{X}_1 \wedge u \simeq v]$ and the result follows as in the case $\bowtie = \not\simeq$.
- If $r \simeq l \vee P_1$ occurs in $\mathcal{X}$ and $u \bowtie v \vee P_2$ occurs in $S^\star|_{\mathcal{X}}$ then the proof is similar to the previous case, replacing the use of the Neg. Assert. rule by the Pos. Assert. rule.
- If both $r \simeq l \vee P_1$ and $u \bowtie v \vee P_2$ occur in $\mathcal{X}$ then $\mathcal{X}$ is not in normal form since by Def. 1 $(u \bowtie v)^c{}_{\downarrow \mathcal{X}^c \setminus (u \bowtie v)^c} \preceq (u[l] \bowtie v)^c$ and $(u[l] \bowtie v)^c \neq (u \bowtie v)^c$, which contradicts point 6 of Def. 9 in the case $\bowtie = \simeq$ and point 2 if $\bowtie = \not\simeq$. This contradicts the hypotheses of the theorem.

Since S' is unsatisfiable and saturated, this set necessarily contains $\square$ by completeness of the standard superposition calculus, which entails that $\square \in S^\star|_{\mathcal{X}}$ (since the clauses in $\mathcal{X}$ are unit hence cannot be empty), hence the result. ■

A seemingly natural idea is to relax the condition of Def. 15 by testing logical entailment instead of set inclusion when comparing constraints (i.e., replacing $\mathcal{Y}_i \subseteq \mathcal{X}$ by $\mathcal{Y}_i \models \mathcal{X}$. However, this makes the calculus incomplete. More precisely, this relaxed notion of redundancy is not compatible with the previous restriction concerning the removal of clauses with non-normalized constraints. Experiments show that the restriction make the calculus more efficient, even with a more restrictive version of the redundancy elimination rule.

3 Storage and Manipulation of Clauses

To store the clauses generated by c$\mathcal{SP}$ and efficiently detect redundancies , a trie-like data structure, the *clausal tree*, is used.

3.1 Clausal Tree

Clausal trees allow one to store efficiently and concisely sets of clauses while taking into account equality axioms. Note that, since our goal is to generate all prime implicates of a formula, we only test one-to-one entailment between clauses (in contrast to the usual practice in automated deduction we cannot discard clauses that are redundant w.r.t. more than one clause since this clause may well be prime). For this reason, we define *e-subsumption*, and we assimilate it to redundancy in the rest of this article.

Definition 25 Let C and D be two clauses. The clause C *e-subsumes* the clause D, written $D \leq_r C$, iff $D \models C$ and $D \preceq C$. A c-clause $[C \mid \mathcal{X}]$ *c-subsumes* a clause $[D \,|\, \mathcal{Y}]$, written $[C \,|\, \mathcal{X}] \leq_r [D \,|\, \mathcal{Y}]$) iff $C \leq_r D$ and $\mathcal{X} \subseteq \mathcal{Y}$. $\lozenge$

Note that both parts of the c-clauses are handled in different ways: the inclusion relation $\subseteq$ used to compare constraints is clearly stronger than the c-subsumption relation $\leq_r$ used for clauses. For instance we have:

$$[a \not\simeq b \vee f(b) \simeq f(d) \,|\, \top] \leq_r [a \not\simeq c \vee b \not\simeq c \vee f(c) \simeq f(d) \,|\, \top], \text{ but}$$
$$[\square \,|\, a \simeq b \wedge f(b) \not\simeq f(d)] \not\leq_r [\square \,|\, a \simeq c \wedge b \simeq c \wedge f(c) \not\simeq f(d)].$$

Clausal trees are similar to the tries of propositional logic [6] that are trees where the edges are labeled with literals and where some additional ordering constraints ensure the sharing of literals. In such a tree, the clauses are the branches, that is the disjunction of the literals labeling the edges from root to leaf. Clausal trees for EUF are enriched with constraints that simplify the application of the redundancy detection algorithms described in parts 3.2 and 3.3 of this section.

Definition 26 A *clausal tree* is inductively defined as either $\square$, or a set of pairs of the form (l, T') where l is a literal and T' a clausal tree. In addition, a clausal tree T with $(l, T') \in T$ must respect the conditions:
- for all l' appearing in T', $l' <_\pi l$,
- there is no clausal tree T'' such that $(l, T'') \in T$.

The set of clauses represented by a clausal tree T is denoted by $\mathcal{C}(T)$ and defined inductively as follows:

$$\mathcal{C}(T) = \begin{cases} \{\square\} & \text{if } T = \square \\ \displaystyle\bigcup_{(l,T')\in T} \left(\bigcup_{D \in \mathcal{C}(T')} l \vee D \right) & \text{otherwise.} \end{cases}$$

$\lozenge$

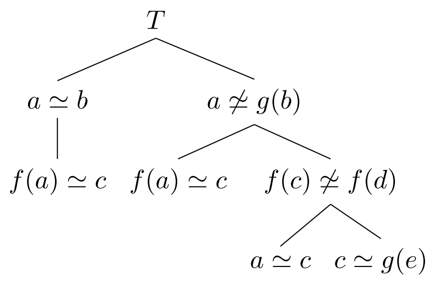
As the definition implies, leaves can be either $\Box$ or $\emptyset$, but in practice if a leaf is labeled with $\emptyset$ (a failure node) then the corresponding branch is irrelevant because a disjunction including the empty set has no meaning as a clause. The only exception is the empty tree, in which the root is labeled with $\emptyset$.

In our context, all the clauses stored in a clausal tree are in normal form. To emphasize this fact, we introduce a type of clausal tree specific to normal clauses.

Definition 27 A clausal tree is *normalized* if all the clauses in $\mathcal{C}(T)$ are in normal form. $\Diamond$

From this point on, any reference to a clausal tree is implicitly about a normalized clausal tree.

Example 28 The structure T below is a clausal tree with the term order $a \prec b \prec c \prec g(c) \prec g(e) \prec f(c) \prec f(d)$. There is no failure node, and for a better readability the labels are associated with the nodes rather than with the edges leading to them.



The clauses in $\mathcal{C}(T)$ are:

$$\begin{array}{l} a \simeq b \vee f(a) \simeq c \\ a \not\simeq g(b) \vee f(a) \simeq c \\ a \not\simeq g(b) \vee f(c) \not\simeq f(d) \vee a \simeq c \\ a \not\simeq g(b) \vee f(c) \not\simeq f(d) \vee c \simeq g(e) \end{array}$$

$\clubsuit$

Notation 29 Let C be a clause in normal form and T be a clausal tree such that $\forall D \in \mathcal{C}(T)$, $C \vee D$ is in normal form and $\forall l \in D$, $C <_\pi l$. In this case, $C.T$ denotes the clausal tree T' such that $\mathcal{C}(T') = \{C \vee D \mid D \in \mathcal{C}(T)\}$. $\Diamond$

The storage of constrained clauses is similar to that of standard clauses. A main clausal tree is used to store the clausal part of constrained clauses and at each leaf of this tree, a trie is appended to store the different constraints associated to the same clause. Note that, according to Def. 15, constraints are compared using set inclusion instead of logical entailment, thus the second tree must be a trie and *not* a clausal tree. In addition, all generated implicates (c-clauses with empty clausal part) should be stored into a clausal tree in order to remove redundancies.

The corresponding manipulation algorithms are neither theoretically nor technically challenging compared to the ones for standard clauses. Thus the choice was made to present these algorithms only for standard clauses.

The manipulation of clausal trees is decomposed into three operations. The first one consists in checking whether a new clause is redundant w.r.t. an existing one already stored in a clausal tree. The second one removes from a clausal tree all clauses that are redundant w.r.t. a given clause. The last one is the insertion of a new clause into a clausal tree. This last operation is straightforward and thus will not be described here. On the other hand, the first two operations are not trivial and are thus carefully described in the remaining parts of this section.

3.2 Testing the Entailment of a Clause

The algorithm isEntailed (Alg. 1) tests whether a clause C is redundant w.r.t. a clause in $\mathcal{C}(T)$, where T is a clausal tree. To do so, a call is made to $\textsc{isEntailed}(C, T, \square, \square)$ and in the recursive calls to $\textsc{isEntailed}(C', T', M, N)$, $M \vee C'$ is equal to C and N represents the path from the root of T to the subtree T'. The principle underlying these calls is to go through the input clause C and tree T while performing the operations necessary to test entailment with the projection method (Th. 7). Note that it is here that the use of the order $<_\pi$ is crucial. Intuitively, the need for this order stems from the fact that the negative literals of a clause C are the ones used to project C onto other clauses. In particular for the projection of positive literals, it is necessary to know of all the negative literals that belong to C, while the reverse does not hold.

Algorithm 1 $\textsc{isEntailed}(C,\ T,\ M,\ N)$

```
Require: $T$ is a clausal tree in normal form, $M \vee C$ and $N$ are clauses in normal form,
    $M$ is negative and $N \models M \vee C$
Ensure: $\textsc{isEntailed}(C,\ T,\ M,\ N) = \top$ iff $\exists D \in \mathcal{C}(T), D \vee N \leq_r M \vee C$
 1: if $T = \square$ then
 2:     return $N \preceq M \vee C$
 3: $T_1 \leftarrow \{(l, T') \in T \mid l_{\downharpoonright M}$ is a contradiction$\}$
 4: if $\bigvee_{(l,T') \in T_1} \textsc{isEntailed}(C,\ T',\ M,\ N \vee l)$ then
 5:     return $\top$
 6: if $C = \square$ then
 7:     return $\bot$
 8: $m_1 \leftarrow \min_{<_\pi} \{m \in C\}$
 9: if $m_1$ is of the form $u \not\simeq v$, with $u \succ v$ then
10:     $T_2 \leftarrow \{(l, T') \in T \mid l_{\downharpoonright M} \not<_\pi m_1$ and $\nexists w, (l_{\downharpoonright M} = u \not\simeq w$, with $u \succ w)\}$
11:     return $\bigvee_{(l,T') \in T_2} \textsc{isEntailed}(C \setminus m_1,\ l.T',\ M \vee m_1,\ N)$
12: else
13:     $T_3 \leftarrow \{(l, T') \in T \mid C_{\downharpoonright M \vee l^c}$ contains a tautological literal$\}$
14:     return $\bigvee_{(l,T') \in T_3} \textsc{isEntailed}(C,\ T',\ M,\ N \vee l)$
```

Proposition 30 *Let C be a non-empty clause such that $M \vee C$ is in normal form. Let l be a literal such that $l_{\downharpoonright M} \models M \vee C$. Let $m_1 = \min_{<_\pi} \{m \in C\}$. If l is a negative literal, then either $l_{\downharpoonright M}$ is a contradiction or $l_{\downharpoonright M} \not<_\pi m_1$.*

Proof. Let $l_{\downharpoonleft M} = u \not\simeq v$. Assuming that $l_{\downharpoonleft M} <_\pi m_1$, since m_1 is minimal in C, $v_{\downharpoonleft M\vee C} = v_{\downharpoonleft M} = v$ and by Theorem 7, $u_{\downharpoonleft M\vee C} = v_{\downharpoonleft M\vee C}$.

- If m_1 is of the form $u \not\simeq v'$ with $v' \prec u$, then $u_{\downharpoonleft C\vee M} = u_{\downharpoonleft C} = v'$, because $C \vee M$ is in normal form and $m_1 \in C$. But then $v' = v_{\downharpoonleft C}$, thus $v' \preceq v$ and $l_{\downharpoonleft M} \not<_\pi m_1$, contradicting our assumption.
- Otherwise $m_1 = s \not\simeq t$ with $s \succ u$ and thus $u_{\downharpoonleft C} = u$ (because the smallest term to be rewritten, namely s, is greater than u) thus $u = v$. ■

Proposition 31 *Let $M \vee C$ be a clause in normal form with M negative. Set $m_1 = \min\limits_{<_\pi}\{m \in C\}$ and let m_1 be a negative literal $u \not\simeq v$ with $u \succ v$. Let l be a literal such that $l_{\downharpoonleft M} \not<_\pi m_1$ and $l_{\downharpoonleft M} \models C \vee M$. In these conditions, the literal $l_{\downharpoonleft M}$ cannot be of the form $u \not\simeq w$ with $u \succ w \succ v$*

Proof. Assume that $l_{\downharpoonleft M} = u \not\simeq w$ with $u \succ w \succ v$. Then because $C \vee M$ is in normal form $u_{\downharpoonleft C\vee M} = v$. In addition, by Theorem 7, since $l_{\downharpoonleft M} \models C \vee M$, we know that $w_{\downharpoonleft C\vee M} = u_{\downharpoonleft C\vee M}$, thus $w_{\downharpoonleft C\vee M} = v$. The fact that $w \prec u$ entails $w_{\downharpoonleft C\vee M} = w_{\downharpoonleft M} = w$ (because the maximal terms in the disequations in C are all greater than w, hence w cannot be rewritten by C, cf. Prop. 12). Finally, we have $w = v$ which contradicts the hypothesis $w \succ v$. Hence, $l_{\downharpoonleft M} \neq u \not\simeq w$ with $u \succ w \succ v$, ■

Theorem 32 *If T is a clausal tree in normal form, $M \vee C$ and N are clauses in normal form, M is negative and $N \models M \vee C$ then the call* isEntailed(C, T, M, N) *terminates and* isEntailed$(C, T, M, N) = \top$ *iff $\exists D \in \mathcal{C}(T), D \vee N \leq_r M \vee C$.*

Proof. The termination proof is trivial, because for all recursive calls, the positive value $|C| + \mathrm{depth}(T)$ strictly decreases.

The correction proof requires two inductions, one for each implication. In the direct direction the proof consists in going through the different cases enumerated by the algorithm to verify that the requirements of the recursive calls are indeed met and that it is possible to derive the desired property from its inductive children. In the converse direction the different cases that validate the right-hand side of the equivalence are considered and matched with the different cases of the algorithm.

Direct implication: assuming isEntailed$(C, T, M, N) = \top$, then one of the "**return**" instructions (except that of line 7) has been triggered and returned true. Let us consider each of them in their order of appearance.

1. Line 2, $T = \Box$ and $N \preceq M \vee C$. In this case $\mathcal{C}(T) = \{\Box\}$. Given that $N \models M \vee C$ by hypothesis, the property $N \leq_r M \vee C$ is verified.
2. Line 4, $T \neq \Box$ and $\bigvee\limits_{(l,T')\in T_1}$ isEntailed$(C, T', M, N \vee l)$ returns true, with $T_1 = \{(l, T') \in T \mid l_{\downharpoonleft M} \text{ is a contradiction}\}$. Thus there exists a pair $(l, T') \in T_1$ such that isEntailed$(C, T', M, N \vee l)$ returns true and $l_{\downharpoonleft M}$ is a contradiction. By Theorem 7, $l \models M$, thus $l \models M \vee C$. Moreover by

hypothesis $N \models M \vee C$. These two properties ensure that the preconditions of the corresponding recursive call are met. By induction $\exists D \in \mathcal{C}(T')$ s.t. $D \vee l \vee N \leq_r M \vee C$. Considering that $l \vee D \in \mathcal{C}(T)$, the result is verified in this case.

3. Line 11, $T \neq \Box$, $C \neq \Box$ and $\bigvee_{(l,T') \in T_2} \textsc{isEntailed}(C \setminus m_1, l.T', M \vee m_1, N)$ with $T_2 = \{(l, T') \in T \mid l_{\downharpoonleft M} \not<_\pi m_1 \text{ and } \nexists w, (l_{\downharpoonleft M} = u \not\simeq w, \text{ with } u \succ w)\}$ and $m_1 (= \min_{<_\pi} \{m \in C\})$ is of the form $u \not\simeq v$, with $u \succ v$. Thus there is a pair $(l, T') \in T$ such that $l \neq u \not\simeq w$ with $u \succ w$, and $m_1 \leq_\pi l_{\downharpoonleft M}$ for which $\textsc{isEntailed}(C \setminus m_1, l.T', M \vee m_1, N)$ returns true. By hypothesis $N \models M \vee C$. Moreover, $M \vee C = M \vee m_1 \vee C \setminus m_1$ and $M \vee m_1$ is purely negative. Therefore the pre-conditions of this recursive call are verified. By induction, its returning true entails $\exists D \in \mathcal{C}(l.T')$ s.t. $D \vee N \leq_r M \vee m_1 \vee C \setminus m_1$. Since $\mathcal{C}(l.T') \subset \mathcal{C}(T)$, the clause D also belongs to $\mathcal{C}(T)$, whence the result in this case.
4. Line 14, $m_1 = u \simeq v$, $l = s \simeq t$ and $\bigvee_{(l,T') \in T_3} \textsc{isEntailed}(C, T', M, N \vee l)$ returns true, with $T_3 = \{(l, T') \in T \mid C_{\downharpoonleft M \vee l^c} \text{ contains a tautological literal}\}$. In this case there exists a pair $(l, T') \in T$ and $m_2 \in C$ s.t. $m^c_{2\downharpoonleft M \vee l^c}$ is a contradiction thus, by Theorem 7, $l_2^c \models M \vee l^c$ which is equivalent to $l \models M \vee l_2$. Since $N \models M \vee C$, the preconditions of the corresponding recursive call are met. Thus by induction $D \vee l \vee N \leq_r M \vee C$ with $D \in T'$. Since $l \vee D \in \mathcal{C}(T)$, we have the desired result for this case.

Converse implication: Assuming there exists a D in $\mathcal{C}(T)$ such that $D \vee N \leq_r M \vee C$ (with C, T, M and N respecting the algorithm's pre-conditions), there are several cases to consider.

- If $T = \Box$, then line 2 is reached and the test $N \preceq M \vee C$ returns true because $D = \Box$ and $N \leq_r M \vee C$ by hypothesis.
- Otherwise, $D = l \vee D'$ with $(l, T') \in T$ and $D' \in \mathcal{C}(T')$. Some sub-cases must be distinguished.
  - If $C = \Box$ then we have $l \vee D' \vee N \leq_r M$, thus also $l \vee D' \vee N \models M$. Since M is purely negative, Theorem 7 ensures that $l_{\downharpoonleft M}$ is a contradiction thus $(l, T') \in T_1$. Moreover $D' \vee l \vee N \leq_r M$ and the pre-conditions of $\textsc{isEntailed}(C, T', M, N \vee l)$, called line 4, are verified. Thus by induction this call returns true, ensuring that $\textsc{isEntailed}(C, T, M, N)$ also returns true line 5.
  - If $C = m_1 \vee C'$ with $m_1 = \min_{<_\pi} \{m \in C\}$ and if l is a negative literal, then by Proposition 30 either 1) $l_{\downharpoonleft M}$ is a contradiction or 2) $l_{\downharpoonleft M} \not<_\pi m_1$. Indeed, the conditions of application of this proposition are verified, since $l \models M \vee C$ and Proposition 2 ensures that $l_{\downharpoonleft M} \models M \vee C$.
    1. If $l_{\downharpoonleft M}$ is a contradiction then $(l, T') \in T_2$ and $N \vee l \models M \vee C$ is implied by the hypotheses, thus the recursive call $\textsc{isEntailed}(C \setminus m_1, l.T', M \vee m_1, N)$, line 11, returns true by induction.
    2. If $l_{\downharpoonleft M} \not<_\pi m_1$ then m_1 is a negative literal (since l is negative and by definition of $<_\pi$), and since $N \vee l \vee D' \leq_r M \vee C$, we have

$N \vee l \vee D' \leq_r M \vee m_1 \vee C \setminus m_1$. Here, by Proposition 31, $(l, T') \in T_2$ thus the recursive call isEntailed$(C \setminus m_1, l.T', M \vee m_1)$ reached line 11 returns true by induction. (The preconditions are respected and $D \leq_r M \vee C = M \vee m_1 \vee C \setminus m_1$.)

– The last case is when $C = m_1 \vee C'$ with $m_1 = \min_{<_\pi} \{m \in C\}$ and $l = s \simeq t$ is a positive literal. If m_1 is a negative literal then $(l, T') \in T_2$ and line 11 is reached as in the previous case. It returns true for the same reason. Otherwise by Theorem 7 there exists a positive literal l_2 in C such that $l_{2 \downarrow M \vee C \vee s \not\simeq t}$ is a tautology, thus $(l, T') \in T_3$. Since $N \vee l \vee D' \leq_r M \vee C$, the precondition $N \vee l \models M \vee C$ of the call isEntailed$(C, T', M, N \vee l)$ reached line 14 is verified and by induction it returns true. ∎

3.3 Pruning Entailed Clauses from a Clausal Tree

The algorithm pruneEntailed (Alg. 2) removes from the input tree T all the clauses redundant w.r.t. the input clause C. It proceeds by going through both objects, performing projections and storing the already considered literals resp. in parameters N and M. Once an entailment is established in this way, all that remains is to compare the selected clauses using the order $\prec$ to detect redundancies. This last part is done by the algorithm pruneInf (Alg. 3).

The first correction result is about pruneInf. It is necessary to guarantee the correction of pruneEntailed which, along with the terminations of both algorithms, is proved in Th. 34

Proposition 33 *Let C and N be clauses in normal form and T be a clausal tree in normal form verifying the preconditions of* pruneInf. *The output tree T_{out} = pruneInf(C, T, N) is such that $\mathcal{C}(T_{out}) = \{D_T \in \mathcal{C}(T) \mid C \not\leq_r D_T \vee N\}$.*

Proof. We proceed by induction. Let $D \in \mathcal{C}(T_{out})$. If $D = \Box$, then $T_{out} = \Box$ by definition of a clausal tree. In this case T_{out} is returned at line 2 and since $C \not\preceq N$, we have $C \not\leq_r N$, thus $D \in \{D_T \in \mathcal{C}(T) \mid C \not\leq_r D_T \vee N\}$. Otherwise, $D = l \vee D'$ with $(l, T'_{out}) \in T_{out}$ and $D' \in \mathcal{C}(T'_{out})$ such that T'_{out} = pruneInf$(C, T', N \vee l)$ with $(l, T') \in T$. By induction $\mathcal{C}(T'_{out}) = \{D_{T'} \in \mathcal{C}(T') \mid C \not\leq_r D_{T'} \vee N \vee l\}$ because the preconditions are verified (in particular $C \models N$ thus $C \models N \vee l$). Therefore $C \not\leq_r D \vee N$, validating the property in this case.

Set $D \in \mathcal{C}(T)$ such that $C \not\leq_r D \vee N$. We know that $C \models N \vee D$ since $C \models N$, hence $C \not\preceq N \vee D$. If $D = \Box$ then $T = \Box$ thus line 2 is reached and $D \in \mathcal{C}(T_{out})$. Otherwise $D = l \vee D'$ with $(l, T') \in T$ and $D' \in \mathcal{C}(T)$. The recursive call line 4 is reached because $C \not\preceq N$, and since $C \models N \vee l$ and $C \not\leq_r D' \vee l \vee N$, we have $D' \in \mathcal{C}(T'_{out})$ by induction (where T'_{out} = pruneInf$(C, T', N \vee l)$). Thus, $T'_{out} \neq \emptyset$ and $D \in \mathcal{C}(T_{out})$. ∎

Theorem 34 *Let $C \vee M$ and N be clauses in normal form and T be a clausal tree in normal form verifying the preconditions of* pruneEntailed. *Then the calls* pruneEntailed(C, T, M, N) *and* pruneInf(C, T, N) *always*

Algorithm 2 PRUNEENTAILED($C,\ T,\ M,\ N$)

```
Require: $T$ is a clausal-tree in normal form, $M \vee C$ and $N$ are clauses in normal form,
    $M \models N$ and ISENTAILED($C \vee M,\ N.T,\ \Box,\ \Box$) $= \bot$.
Ensure: $\mathcal{C}(T_{out}) = \{D \in \mathcal{C}(T) \mid C \vee M \not\leq_r D \vee N\}$, with
    $T_{out}$ = PRUNEENTAILED($C,\ T,\ M,\ N$).
 1: if $C = \Box$ then
 2:     return PRUNEINF($M,\ T,\ N$)
 3: select $m_1 \in C$ s.t. $m_{1\downharpoonleft N} = \min\limits_{<_\pi} \{m_{\downharpoonleft N} \mid m \in C\}$
 4: if $m_{1\downharpoonleft N}$ is a contradiction then
 5:     return PRUNEENTAILED($C \setminus m_1,\ T,\ M \vee m_1,\ N$)
 6: if $T = \Box$ then
 7:     return $T$
 8: $T_1 \leftarrow \{(l, T') \in T \mid l = u \not\simeq v \wedge m_{1\downharpoonleft N} \succeq l\}$
 9: $T_{out1} \leftarrow \{(l,$ PRUNEENTAILED($C,\ T',\ M,\ N \vee l$)$)|$
            $(l, T') \in T_1 \wedge$ PRUNEENTAILED($C,\ T',\ M,\ N \vee l$) $\neq \emptyset\}$
10: if $m_1$ is positive then
11:     $T_2 \leftarrow T \setminus T_1$
12:     $T_{out2} \leftarrow \{(l,$ PRUNEENTAILED($C \setminus L_l,\ T',\ M \vee L_l,\ N \vee l$)$)|$
                $(l, T') \in T_2 \wedge L_l = \{m \in C \mid l_{\downharpoonleft N \vee m}$ is tautological$\} \wedge$
                PRUNEENTAILED($C \setminus L_l,\ T',\ M \vee L_l,\ N \vee l$) $\neq \emptyset\}$
13:     return $T_{out1} \cup T_{out2}$
14: else
15:     return $T_{out1} \cup T \setminus T_1$
```

terminate and T_{out} = PRUNEENTAILED(C, T, M, N) *is such that* $\mathcal{C}(T_{out}) = \{D \in \mathcal{C}(T) \mid C \vee M \not\leq_r D \vee N\}$.

PROOF. The termination of both algorithms is ensured by the same argument: for all recursive calls, the value of $|C| + \text{depth}(T)$ strictly decreases.

The schema of the correction proof of PRUNEENTAILED is identical to that of Th. 32. Let D be a clause in $\mathcal{C}(T_{out})$.

- If $C = \Box$ then line 2 was reached to generate T_{out}. The preconditions of PRUNEINF are respected (in particular $M \models N$ by hypothesis). By Proposition 33, $D \in \{\mathcal{C}(T) \mid M \not\leq_r D \vee N\}$, whence the result in this case.
- Otherwise, C is of the form $m_1 \vee C'$, where m_1 is such that $m_{1\downharpoonleft N} = \min\limits_{<_\pi} \{m_{\downharpoonleft N} \mid m \in C\}$.
  - If $m_{1\downharpoonleft N}$ is a contradiction then T_{out} is returned at line 5 and $m_1 \models N$ by Theorem 7. By induction $C \setminus m_1 \vee m_1 \vee M \not\leq_r D \vee N$.
  - Otherwise if $T = \Box$ then $T_{out} = T$ and $m_1 \not\models N$ by Theorem 7. Hence $M \vee C \not\models N$.
  - Else $m_{1\downharpoonleft N}$ is not a contradiction and $T \neq \Box$ thus $D = l \vee D'$, where $(l, T'_{out}) \in T_{out}$ and $D' \in \mathcal{C}(T'_{out})$ such that one of the following holds:
    1. $(l, T'_{out}) \in T_{out1}$, in which case $l = u \not\simeq v$, $m_{1\downharpoonleft N} \succeq l$ and T'_{out} = PRUNEENTAILED$(C, T', M, N \vee l)$;

Algorithm 3 $\text{PRUNEINF}(C, T, N)$

```
Require: $T$ is a clausal-tree in normal form, $C$ in a clause in normal form, $N$ is a
    clause in normal form, $C \models N$.
Ensure: $\mathcal{C}(T_{out}) = \{D \in \mathcal{C}(T) | C \not\leq_r D \vee N\}$, with $T_{out} = \text{PRUNEINF}(C, T, N)$.
1: if $T = \Box$ and $C \not\preceq N$ then
2:     return $T$
3: if $C \not\preceq N$ then
4:     return $\{(l, \text{PRUNEINF}(C, T', N \vee l)) | (l, T') \in T \wedge$
               $\text{PRUNEINF}(C, T', N \vee l) \neq \emptyset\}$
5: return $\emptyset$
```

2. l and m_1 are positive literals and $(l, T'_{out}) \in T_{out2}$ in which case $T'_{out} = \text{PRUNEENTAILED}(C \setminus L_l, T', M \vee L_l, N \vee l)$ with $(l, T') \in T$ and $L_l \leftarrow \{l' \in C \mid l_{\downharpoonright N \vee l'} \text{ is tautological}\}$;
3. m_1 is negative and $m_{1\downharpoonright N} \prec l$ in which case $T'_{out} = T'$ with $(l, T') \in T \setminus T_1$.

In all cases $m_1 \not\models N$ by Theorem 7. In the first case, the preconditions of the recursive call of line 9 are respected ($M \models N \vee l$). Therefore $C \vee M \not\leq_r D' \vee l \vee N$. In the second case, Theorem 7 allows us to assert that $M \vee L_l \models N \vee l$. Thus, the preconditions of the recursive call line 12 (reached because m_1 is positive) are verified and by induction $M \vee L_l \vee C \setminus L_l \not\leq_r D' \vee l \vee N$. In the last case, by Proposition 12 and Theorem 7, $m_1 \not\models D$ thus $C \not\models N \vee D$.

For the second inclusion, let D be a clause in $\mathcal{C}(T)$ such that $C \vee M \not\leq_r D \vee N$.

- If $C = \Box$, given that $M \models N$, necessarily $M \not\preceq D \vee N$. Thus, T_{out} must be generated at line 2 and, by Proposition 33, $D \in \mathcal{C}(T_{out})$.
- Otherwise C is of the form $m_1 \vee C'$ where m_1 is such that $m_{1\downharpoonright N} = \min\limits_{<_\pi} \{m_{\downharpoonright N} \mid m \in C\}$.
  - If $m_{1\downharpoonright N}$ is a contradiction then $m_1 \models N$ by Theorem 7 and T_{out} is returned at line 5. Since $C \setminus m_1 \vee m_1 \vee M \not\leq_r D \vee N$, by induction $D \in \mathcal{C}(T_{out})$.
  - If $m_{1\downharpoonright N}$ is not a contradiction then either $T = \Box$, in which case $T_{out} = T$ and the result is straightforward, or the same three cases as in the other direction of the proof can be studied separately (with $D = l \vee D'$, $(l, T') \in T$ and $D' \in \mathcal{C}(T')$).
    1. If l is negative and $m_{1\downharpoonright N} \succeq l$, then $(l, T') \in T_1$. Set $T'_{out1} = \text{PRUNEENTAILED}(C, T', M, N \vee l)$, as in line 9. By induction $D' \in \mathcal{C}(T'_{out1})$ thus $D \in \mathcal{C}(T_{out1})$.
    2. If l and m_1 are both positive literals then the execution path goes through line 12. The clause $L_l = \{l' \in C \mid l_{\downharpoonright N \vee l'} \text{ is tautological}\}$ is such that $L_l \models N \vee l$ by Theorem 7, thus $M \vee L_l \models N \vee l$. Hence the preconditions of the recursive call $\text{PRUNEENTAILED}(C \setminus L_l, T', M \vee L_l, N \vee l) = T'_{out2}$ are respected. Since $C \setminus L_l \vee M \vee L_l \not\leq_r D' \vee l \vee N$, by induction $D' \in \mathcal{C}(T'_{out2})$ thus $D \in T_{out2}$.

3. Finally, if m_1 is negative and $m_{1\downharpoonright N} \prec l$ then $(l, T') \in T \setminus T_1$ and line 15 is triggered, thus $D \in \mathcal{C}(T_{out})$. ∎

References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998), `http://www4.informatik.tu-muenchen.de/\%003cnipkow/TRaAT/`
2. De Kleer, J.: An improved incremental algorithm for generating prime implicates. In: Proceedings of the National Conference on Artificial Intelligence. pp. 780–780. John Wiley & Sons ltd (1992)
3. Dershowitz, N.: Orderings for term-rewriting systems. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science. pp. 123–131. IEEE Computer Society, Washington, DC, USA (1979), `http://dl.acm.org/citation.cfm?id=1398508.1382612`
4. Echenim, M., Peltier, N., Tourret, S.: A deductive-complete constrained superposition calculus for ground flat equational clauses. In: 4th Workshop on Practical Aspects of Automated Reasoning. (2014)
5. Echenim, M., Peltier, N., Tourret, S.: A rewriting strategy to generate prime implicates in equational logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning, pp. 137–151. No. 8562 in Lecture Notes in Computer Science, Springer International Publishing (Jul 2014), `http://link.springer.com/chapter/10.1007/978-3-319-08587-6_10`
6. Fredkin, E.: Trie memory. Commun. ACM 3(9), 490–499 (1960)
7. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Handbook of Automated Reasoning, pp. 371–443 (2001)