A New Instantiation Scheme for Satisfiability Modulo Theories (Research Report)

Mnacho Echenim and Nicolas Peltier

December 20, 2009

Most formal verification tools rely on procedures that decide the validity or, dually, the satisfiability of logical formulas. In general, the validity of the formula (or set of clauses) needs only be tested *modulo* a background theory $\mathcal{T}$. In formal software verification for example, the background theory can define one or a combination of data structures such as arrays or lists. These problems are known as *$\mathcal{T}$-decision problems* or more comonly, *SMT problems*, and the tools capable of solving these problems are known as *$\mathcal{T}$-decision procedures*, or *SMT solvers*[1].

A lot of research has been devoted to the design of SMT solvers that are both efficient and scalable. Generally, state-of-the-art SMT solvers rely on algorithms based on the DPLL procedure [10] to deal with the boolean part of the SMT problems. These solves deal with the theory reasoning by applying methods ranging from the *eager approach*, which consists in applying sophisticated techniques to reduce the entire SMT problem to an equisatisfiable SAT problem, to the *lazy approach*, which consists in searching for a conjunction of literals satisfying the boolean part of the formula, and testing whether this conjunction of literals is satisfiable modulo the background theory. A survey on SMT solvers and the different approaches can be found in [5].

Quantifier reasoning is meant to tackle the issue of solving SMT problems with formulas involving quantifiers. Most approaches rely on the original work of [13] on the Simplify prover, in which heuristics for quantifier instantiation are devised. State-of-the-art techniques include [15, 12].

The rewrite-based approach to solving SMT problems, initiated by [2], consists in employing a theorem prover for first-order logic with equality to solve the SMT problems. This approach is appealing, since by feeding any finitely axiomatized theory along with a set of gound clauses, one obtains a system that is refutationally complete. If the theorem prover is guaranteed to terminate on the input set, then it acts as a decision procedure for the background theory. Thus, much research on this subject is devoted to determining results on termination for the theorem prover [1, 8, 21, 6]. The main issue with the rewrite-based approach is that theorem provers are not designed to handle the boolean part of a formula as efficiently as possible. Therefore, they do not perform well on SMT problems with a large boolean part. A solution to this problem consists in integrating the theorem prover with a DPLL-based tool, and although this raises new issues, such an integration was accomplished in, e.g., [23, 24, 11].

Another solution to this problem was invvestigated in [7], which consists in decomposing the SMT problem into a *definitional* part, made up of an conjunction of ground literals, and an *operational* part, containing the boolean structure of the problem. During the first stage, the theorem prover is fed the definitional part of the SMT problem along with the background theory, and the saturation process *compiles* the theory away. During the second stage, the saturated set generated by the theorem prover is fed to a DPLL-based tool, along with the operational part of the SMT problem. This approach allows to exploit the full power of the theorem prover and the DPLL-based tool, without requiring a tight integration between them. However, in the theory of arrays for example, the set of clauses obtained after compiling the theory away was not ground, and required an additional instantiation before being fed to the DPLL-based tool.

The goal of this paper is to investigate how the instantiation phases of the saturation process can be singled out in order to devise a generic instantiation scheme for solving SMT problems. Solving an SMT problem modulo a background theory for which the instantiation scheme is complete would thus reduce to testing the satisfiability of a ground formula in first-order logic with equality, with no mention to any background

[1] SMT stands for *Satisfiability Modulo Theories*

theory. This scheme is meant to be as efficient as possible, i.e., instantiate the non-ground clauses under consideration as little as possible. This approach is close to that of [9] for the theory of arrays. However, contrary to that of [9] which is model-theoretic, this one is proof-theoretic, and is not restricted to just one theory or its extensions. The efficiency requirement comes at the expense of completeness, and contrary to the schemes of [14, 22, 16], it is not always guaranteed that the original set of clauses and the instantiated one are equisatisfiable. However, the class of theories for which the scheme is complete is large enough to capture several theories of interest in the SMT community.

This paper presents the instantiation scheme that was devised, along with two sets syntactic criteria on clauses, that guarantee the instantiated set of clauses and the original one are equisatisfiable. The first set of criteria is simpler to implement, and the other is more general. The imposed conditions have a common characteristic: they are based on the arguments of function symbols. Depending on their positions, the former may be required to be ground, or to have a limited depth. These conditions come up quite naturally in our proof-theoretic setting, and are general enough for many theories of data structures from SMT problems to comply by them. These criteria have been implemented[2], and have allowed us to verify automatically that the instantiation scheme can be applied to several theories such as arrays, records or lists.

1 Preliminaries

1.1 Basic Definitions

In this section we briefly review some usual definitions and notations in Logic and Automated Theorem Proving. The results in this section are standard and their proofs are omitted; we refer the reader to, e.g., [3] for details.

We assume given a set of *variables* $\mathcal{V}$, a set of *function symbols* Σ (containing at least a constant symbol) and an *arity function* mapping each element of Σ to a natural number. Σ_n denotes the set of symbols of arity n in Σ. Throughout this paper, a, b, c always denote constant symbols, f, g, h denote function symbols and x, y, z denote variables (possibly with indices). The symbol `true` denotes a special constant symbol used to encode predicate symbols.

The set of *terms*, *atoms*, *literals* and *clauses* are defined as usual on Σ and $\mathcal{V}$, using the equality symbol $\simeq$. For every expression (term, clause etc.) e, we denote by $V(e)$ the set of variables occurring in e, and $\mathrm{head}(t)$ denotes the head symbol of the term t. A term is *linear* if it contains at most one occurrence of each variable. The notions of *interpretations*, *models* and *satisfiability* are defined as usual. Two sets of clauses S, S' are *equisatisfiable* (written $S \equiv_{sat} S'$) if either S, S' are both unsatisfiable or S, S' are both satisfiable.

A *substitution* is a function mapping every variable to a term. The set of variables x such that $x\sigma \neq x$ is called the *domain* of σ and denoted by $dom(\sigma)$. A substitution σ of domain $x_1, \ldots, x_n$ such that $x_i\sigma = t_i$ for $i = 1, \ldots, n$ is usually denoted by $\{x_i \mapsto t_i \mid i \in [1..n]\}$. As usual, a substitution can be extended into a homomorphism on terms, atoms, literals and clauses. The image of an expression e by a substitution σ will be denoted by $e\sigma$. If E is a set of expressions, then $E\sigma$ denotes the set $\{t\sigma \mid t \in E\}$. The composition of two substitutions σ and θ is denoted by $\sigma\theta$. A substitution σ is *more general* than θ if there exists a substitution η such that $\theta = \sigma\eta$. The substitution σ is a *renaming* if it is injective and $\forall x \in dom(\sigma), x\sigma \in \mathcal{V}$, and it is a *unifier* of two terms t, s if $t\sigma = s\sigma$. Any unifiable pair of terms (t, s) has a most general unifier, unique up to a renaming, and denoted by $mgu(t, s)$. A term or clause containing no variable is *ground*. A substitution σ is *ground* if $x\sigma$ is ground, for every variable x in its domain.

Lemma 1 *Let t, s be two unifiable terms and let $\sigma = mgu(t, s)$ be a flat substitution. Let θ be a flat and ground substitution such that $\forall x \in dom(\theta)$, $x\theta = x\sigma\theta$. Then $t\theta$ and $s\theta$ are unifiable. Moreover, if $\eta = mgu(t\theta, s\theta)$ then $dom(\eta) = dom(\sigma) \setminus dom(\theta)$ and for all $x \in dom(\eta)$, $x\eta = x\sigma\theta$.*

Proof. A *unification problem* is a conjunction of equations (or `false`). If ϕ, ψ are two unification problems, then we write $\phi \rightarrow_{unif} \psi$ if ψ is obtained from ϕ by applying the usual replacement or decomposition rules [17].

σ is of the form $\{x_i \mapsto u_i \mid i \in [1..m]\}$, where $x_1, \ldots, x_m$ are pairwise distinct variables not occurring in $u_1, \ldots, u_m$. Moreover, there exists a sequence of unification problems $(\phi_i)_{i \in [1..n]}$ such that $\phi_1 = (t =^? s)$,

[2] http://capp.imag.fr/fish.html

$\forall i \in [1..n-1], \phi_i \rightarrow_{unif} \phi_{i+1}$ and $\phi_n = \bigwedge_{i=1}^{m}(x_i =^? u_i)$. We assume that the unification rules are applied with the following priority: decomposition first, then replacement by terms w such that $w\theta \in T_0$ and finally the remaining replacements.

Let k be an index in $[1..n]$. We shall prove that $\phi_k\theta \rightarrow^*_{unif} \phi_{k+1}\theta$.

If ϕ_{k+1} is obtained from ϕ_k by decomposition, then ϕ_k, ϕ_{k+1} are respectively of the form $f(\vec{t}) =^? f(\vec{s}) \wedge \psi$ and $\vec{t} =^? \vec{s} \wedge \psi$. Then $\phi_k\theta$ is of the form $f(\vec{t})\theta =^? f(\vec{s})\theta \wedge \psi\theta$, thus the decomposition rule applies and deduces $\vec{t}\theta =^? \vec{s}\theta \wedge \psi\theta$ i.e. $\phi_{k+1}\theta$.

If ϕ_{k+1} is obtained from ϕ_k by replacement then ϕ_k, ϕ_{k+1} are respectively of the form $x =^? w \wedge \psi$ and $x =^? w \wedge \psi\{x \rightarrow w\}$ where x is a variable in $x_1, \ldots, x_m$ and w is a term not containing x. By definition, $x \in dom(\sigma)$. $\phi_k\theta$ is $x\theta =^? w\theta \wedge \psi\theta$.

If $x \notin dom(\theta)$ then $x\theta = x$. $x \notin V(w\theta)$ (since $x \notin V(w)$ and θ is ground). Thus the replacement rule applies and deduces $x =^? w\theta \wedge \psi\theta\{x \rightarrow w\theta\}$ i.e. $\phi_{k+1}\theta$.

Now assume that $x \in dom(\theta)$. We have, by definition, $x\sigma = w\sigma$, thus $x\sigma\theta = w\sigma\theta$ and by the above condition, since $x \in dom(\theta)$, $x\theta = w\sigma\theta$. If $w \in dom(\theta)$, or if $w \notin \mathcal{V}$ then we have $w\sigma\theta = w\theta$, thus $x\theta = w\theta$ and $\phi_k\theta = \phi_{k+1}\theta$. If $w \in \mathcal{V} \setminus dom(\theta)$ then we must have $x\sigma \neq w$ (otherwise we would have $w\theta = x\theta$ hence $w \in dom(\theta)$). This means that $w \in dom(\sigma)$ and w is replaced at some step. Thus there exists in the unification problem an equation of the form $y =^? x\sigma$, where $y \in \mathcal{V}$. But then, according to the above strategy, the replacement rule should have been applied first on this equation before $x =^? w$ (since $x\sigma\theta \in T_0$).

Therefore, we have $t\theta =^? s\theta \rightarrow^*_{unif} \bigwedge_{i=1}^{m}(x_i\theta =^? u_i\theta)$. By the above reasoning, for every $i, j \in [1..m]$, we have either $x_i\theta = x_i$ and $x_i \notin V(u_j\theta)$, or $x_i \in dom(\theta)$ and $x_i\theta = u_i\theta$. Thus $\{x_i \mapsto u_i\theta \mid i \in [1..m], x_i \notin dom(\theta)\} = mgu(t\theta, s\theta)$. ∎

A *position* is a sequence of natural numbers. ϵ denotes the empty sequence and $p.q$ denotes the concatenation of p and q. p is a *position in* t if either $p = \epsilon$ or $p = i.q$ and $t = f(t_1, \ldots, t_n)$ and q is a position in t_i. $t|_p$ and $t[s]_p$ respectively denote the subterm at position p in t and the term obtained by replacing the term at position p by s: $t|_\epsilon \stackrel{\text{def}}{=} t$, $f(t_1, \ldots, t_n)|_{i.q} \stackrel{\text{def}}{=} t_i|_q$, $t[s]_\epsilon \stackrel{\text{def}}{=} s$ and $f(t_1, \ldots, t_n)[s]_{i.q} \stackrel{\text{def}}{=} f(t_1, \ldots, t_{i-1}, t_i[s]_q, t_{i+1}, \ldots, t_n)$. These notions extend straightforwardly to atoms, literals or clauses.

Flatness

A term is *flat* if it is a variable or a constant symbol. The set of flat terms is denoted by T_0 ($T_0 \stackrel{\text{def}}{=} \mathcal{V} \cup \Sigma_0$). A non flat term is *complex*. A clause C is *flat* if for every literal $t \simeq s$ or $t \not\simeq s$ occurring in C, $t, s \in T_0$. A substitution σ is *flat* if $\forall x \in \mathcal{V}, x\sigma \in T_0$. A clause C is a *flat instance* of D if there exists a flat substitution σ such that $C = D\sigma$ (note that C is not necessarily flat). Flat substitutions are stable by composition:

Proposition 2 *Let σ and μ be flat substitutions. Then $\sigma\mu$ is also flat.*

Definition 3 Given a clause C, we denote by C° the disjunction of the literals in C that are both ground and flat and by $C^\bullet$ the disjunction of the literals in C that are either non flat or non ground ($C = C^\circ \vee C^\bullet$).◊

Any set of ground clauses can be *flattened*, by introducing fresh constants that serve as names for complex terms. This operation produces a set of ground clauses such that the only non-flat clauses are of the form $f(a_1, \ldots, a_n) \simeq b$, for constants $a_1, \ldots, a_n, b$. For example, $S = \{f(a) \not\simeq f(c) \vee a \simeq c, f(b) \not\simeq f(c) \vee b \simeq c\}$ is flattened by introducing the fresh constants a', b' and c', and replacing S by

$$\{f(a) \simeq a',\ f(b) \simeq b',\ f(c) \simeq c',\ a' \not\simeq c' \vee a \simeq c,\ b' \not\simeq c' \vee b \simeq c\}.$$

The original set of ground clauses and the flattened one are equisatisfiable.

Quasi-flat and Quasi-closed Terms

We characterize two syntactic classes of terms that will be useful in the following. These classes are defined by imposing that some of the arguments of function symbols are not variables, and that others have a limited depth.

Definition 4 We associate to every function symbol f of arity n two sets of indices $\mathrm{I}_0(f)$ and $\mathrm{I}_{\mathrm{nv}}(f)$ in $[1..n]$ such that $\mathrm{I}_0(f) \cup \mathrm{I}_{\mathrm{nv}}(f) = [1..n]$.

A term t is:

- *quasi-flat* if it is not a variable, and for every subterm $f(t_1, \ldots, t_n)$ of t and for every $i \in \mathrm{I}_0(f)$, $t_i \in T_0$.
- *quasi-closed* if it is not a variable, and for every subterm $f(t_1, \ldots, t_n)$ of t and for every $i \in \mathrm{I}_{\mathrm{nv}}(f)$, $t_i \notin \mathcal{V}$.

A set of clauses S is quasi-flat (resp. quasi-closed) if all terms occurring in S are quasi-flat (resp. quasi-closed). ◊

Informally (see Definition 4 for details), $\mathrm{I}_0(f)$ denotes the set of indices of the arguments of f that must be flat and $\mathrm{I}_{\mathrm{nv}}(f)$ denotes the set of indices that must be non variables.

Example 5 Let f be a function symbol of arity 3. Assume that $\mathrm{I}_0(f) = \{1, 2\}$ and that $\mathrm{I}_{\mathrm{nv}}(f) = \{2, 3\}$. Then:

- $f(x, a, b)$ and $f(a, b, f(a, b, c))$ are quasi-closed and quasi-flat,
- $f(x, x, a)$ is quasi-flat but not quasi-closed because of index 2,
- $f(f(a, b, c), a, b)$ is quasi-closed but not quasi-flat because of index 1,
- $f(f(a, x, b), a, b)$ is neither quasi-flat nor quasi-closed because of indices 1 and 2. ♣

1.2 Superposition

We review some basic notions about superposition-based theorem proving [4]. Our presentation is not exactly standard, since in general presentations, clauses are implicitly renamed in derivations. The proofs of completeness for our instantiation method, on the other hand, require keeping track of variables in derivations. In the standard setting, this would mean defining equivalence classes of variables, two variables being equivalent if one is an implicit renaming of the other. We did not find this approach to be practical, and chose instead to adopt a version of the superposition calculus in which variables are *rigid*, so that the same variable can appear throughout a derivation. Most results from standard definitions of the superposition calculus are straightforwardly transposed to our setting

Selection and Inference

Let $<$ denote a reduction ordering which is substitution-monotonic (i.e. $(t < s) \Rightarrow (t\sigma < s\sigma)$, for every σ). We assume that if a is a constant and t is a complex term, then $a < t$. This property on orderings is termed as the *goodness* property in [1]. The ordering $<$ is extended to atoms, literals and clauses using multiset extension. A literal L is *maximal* in a clause C if for every $L' \in C$, $L \not< L'$.

We consider a *selection function* sel which maps every clause C to a set of *selected literals* in C. A term t is *eligible* in a clause C if t is not a variable and it occurs in a term u, where $\mathrm{sel}(C)$ contains a literal of the form $u \simeq v$ or $u \not\simeq v$ with $u \not< v$.

Definition 6 A clause C is *variable-eligible* if $\mathrm{sel}(C)$ contains a literal of the form $x \simeq t$ (resp. $x \not\simeq t$), where $x \in \mathcal{V}$ and $x \not< t$. ◊

Proposition 7 *Every flat clause is either ground or variable-eligible.*

We consider the calculus (parameterized by $<$ and sel) of Figure 1.

Superposition calculus:

Superposition	$C \vee t \simeq s, D \vee u \simeq v \quad \rightarrow \quad (C \vee D \vee t[v]_p \simeq s)\sigma$
	if $\sigma = mgu(u, t\|_p)$, $u\sigma \not\leq v\sigma, t\sigma \not\leq s\sigma$, $t\|_p$ is not a variable,
	$(t \simeq s)\sigma \in \text{sel}([C \vee t \simeq s]\sigma)$, $(u \simeq v)\sigma \in \text{sel}([D \vee u \simeq v]\sigma)$.
Paramodulation	$C \vee t \not\simeq s, D \vee u \simeq v \quad \rightarrow \quad (C \vee D \vee t[v]_p \not\simeq s)\sigma$
	if $\sigma = mgu(u, t\|_p)$, $u\sigma \not\leq v\sigma, t\sigma \not\leq s\sigma$, $t\|_p$ is not a variable,
	$(t \not\simeq s)\sigma \in \text{sel}([C \vee t \not\simeq s]\sigma)$, $(u \simeq v)\sigma \in \text{sel}([D \vee u \simeq v]\sigma)$.
Reflection	$C \vee t \not\simeq s \quad \rightarrow \quad C\sigma$
	if $\sigma = mgu(t, s)$, $(t \not\simeq s)\sigma \in \text{sel}([C \vee t \not\simeq s]\sigma)$.
Eq. Factorisation	$C \vee t \simeq s \vee u \simeq v \quad \rightarrow \quad (C \vee s \not\simeq v \vee t \simeq s)\sigma$
	if $\sigma = mgu(t, u)$, $t\sigma \not\leq s\sigma$, $u\sigma \not\leq v\sigma$, $(t \simeq s)\sigma \in \text{sel}([C \vee t \simeq s \vee u \simeq v]\sigma)$.

Figure 1: The superposition calculus

Redundancy

A *tautology* is a clause containing two complementary literals or a literal of the form $t \simeq t$. A clause C is *subsumed* by a clause D if there exists a substitution σ such that $D\sigma \subseteq C$.

A ground clause C is *redundant* in S if there exists a set of clauses S' such that $S' \models C$, and for every $D \in S'$, D is an instance of a clause in S such that $D < C$. A non ground clause C is *redundant* if all its instances are redundant. In particular, every (strictly) subsumed clause and every tautological clause is redundant.

In practice, one has to use a decidable approximation of this notion of redundancy. We will assume that a clause C is redundant if it can be reduced by some demodulation steps (i.e. by superposition without instantiation of the variable in the "into" clause) to either a tautology or to a subsumed clause (it is obvious that all the clauses satisfying this property are redundant in the previous sense).

We have the following:

Proposition 8 *If a clause C is redundant w.r.t. a set of clauses $S \cup \{D\}$ and if D is redundant w.r.t. S, then C is redundant w.r.t. S.*

Derivation Relations and Saturated Sets

If S is a set of clauses, we write $S \rightarrow^{\text{sel}}_{<,\sigma} C$ when C can be deduced from S in one step by applying one of the rules of Figure 1, using the m.g.u. σ. We adopt the convention that S contains only the premises of the rule and the clauses in S are *not* renamed, for instance:

$$\{f(x) \simeq g(y), f(x) \simeq h(y)\} \rightarrow^{\text{sel}}_{<,id} (h(y) \simeq g(y))$$

and

$$\{f(x) \simeq a, f(g(x)) \simeq b\} \not\rightarrow^{\text{sel}}_{<,\sigma} (a \simeq b), \text{ for any substitution } \sigma.$$

We denote by $\mathcal{S}^{\text{sel}}_{<}(S)$ the set of clauses C such that $S' \rightarrow^{\text{sel}}_{<,\sigma} C$ where all the clauses in S' are pairwise variable-disjoint renamings of clauses in S. A set of clauses S is *saturated* if every clause $C \in \mathcal{S}^{\text{sel}}_{<}(S)$ is redundant in S. If S is saturated and if for every clause $C \in S$ $\text{sel}(C)$ either contains a negative literal, or contains all the maximal literals in C, then S is satisfiable if $\square \notin S$ (see [4] for details).

2 An Instantiation-Based Proof Procedure

A close inspection of the instantiation phases of the superposition calculus in the rewrite-based approach to SMT problems revealed that for the considered background theories, it is sufficient to instantiate the axioms

of the theory using only the ground terms appearing in the original problem. The main idea we exploit in our instantiation scheme is the fact that the flattening operation permits to view constants as names for complex terms. This is why we focus on instantiations based only on constants.

Let S be a set of clauses whose satisfiability we wish to test by considering a finite set of its ground instances. A first, intuitive way to instantiate the non-ground clauses in S consists in replacing all variables by the constants appearing in S *in every possible way*. A formal definition of the resulting set follows.

Definition 9 Given a set of clauses S, we let S_{c} denote the set

$$S_{\mathrm{c}} \quad = \quad \{C\theta \,|\, C \in S,\ \theta \text{ is flat and ground}\}. \qquad \Diamond$$

If S_{c} is unsatisfiable, then so is S; yet, it is clear that in general, S_{c} may be satisfiable although S is not. However, it is possible to determine a sufficient condition of the equisatisfiability of S_{c} and S, based on the inferences that derive the empty clause starting from S.

Definition 10 Given a set of clauses S, we denote by $\mathcal{SF}^{\mathrm{sel}}_{<}(S)$ the set of clauses $C \in \mathcal{S}^{\mathrm{sel}}_{<}(S)$ such that the mgu of the inference that generated C is flat.

A set of clauses S admits a *flat refutation* if there exists a sequence $S_0, \ldots, S_n$ such that:

- $S_0 = S$;
- for all $i = 1..n$, $S_i = S_{i-1} \cup \mathcal{SF}^{\mathrm{sel}}_{<}(S_{i-1})$;
- $\Box \in S_n$. $\Diamond$

The instantiation of the clauses with all the constants occurring in the original set of clauses is complete if the latter admits a flat refutation:

Theorem 11 *If S admits a flat refutation, then S_{c} is unsatisfiable.*

Proof. Let $S'_0, \ldots, S'_n$ denote the sequence such that:

- $S'_0 = S_{\mathrm{c}}$;
- for all $i = 1..n$, $S'_i = S'_{i-1} \cup \mathcal{S}^{\mathrm{sel}}_{<}(S'_{i-1})$.

We prove by induction on k that for any flat ground substitution θ and for any clause $C \in S_k$, the clause $C\theta$ is in S'_k. Thus, if $\Box \in S_n$, then also, $\Box \in S'_n$, which proves that S_{c} is unsatisfiable.

If $k = 0$ then the result is obvious: C is a clause in S, hence, by construction, $C\theta$ is a clause in S_{c}. Now assume the result is true for some $k \geq 0$, let $C \in S_{k+1}$, and consider a flat ground substitution θ. We assume that C is obtained by a paramodulation from $C_1 = u \simeq v \vee D$ into $C_2 = l[u'] \simeq r \vee D'$; the proof in all the other cases is similar. Here, $C = (l[v] \simeq r \vee D \vee D')\sigma$, and by hypothesis, σ is flat. Let $\mu = \sigma\theta$, then μ is also a flat ground substitution; therefore, by the induction hypothesis, both $C_1\mu$ and $C_2\mu$ are in S'_k. Since the paramodulation of $C_1\mu$ into $C_2\mu$ generates C, the latter is in S'_{k+1}, which concludes the proof. ∎

The main inconvenience is that ensuring a set of clauses admits a flat refutation is not decidable in general, and that even if this condition were guaranteed, S_{c} could be a very large set. We thus define an instantiation scheme which generates a set of ground clauses potentially much smaller than S_{c}, while still restricting ourselves to instantiations based on constants.

Definition of the Instantiation Scheme

The basic idea of our instantiation scheme is close to the one of existing instantiation-based methods (see for instance [20, 14]): namely, to use unification in order to generate relevant instances. For example, from the clauses $f(x) \simeq a, f(b) \not\simeq a$, we shall generate, by unifying the terms $f(x)$ and $f(b)$, the instance $f(b) \simeq a$ of the first clause, yielding the ground unsatisfiable set $\{f(b) \simeq a, f(b) \not\simeq a\}$. More generally, if C is a clause containing a (non variable) term t and if s is a term occurring in the clause set, we shall derive the instance $C\sigma$ such that σ is a unifier of t and s.

The difference between our approach and existing ones is twofolds. First we restrict ourselves to *flat* unifiers, simply by discarding complex subterms. This is possible since the clause sets we consider will always admit flat refutations. Second, we use a *relaxed* (pseudo-)unification algorithm, in which decomposition is only applied at the root position. More precisely, two (proper) subterms of t, s are always taken to be identical, except if one of them is a variable x and the other is a constant symbol c, in which case we add the binding $x \mapsto c$ into the substitution σ. In particular if two such bindings occur, namely $x \mapsto a, x \mapsto b$, where $a \neq b$, then the unification algorithm still succeeds, and by convention, the smallest constant symbol among a, b according to the ordering $<$ is chosen as the value of x. Bindings of the form $x \mapsto f(t_1, \ldots, t_n)$ where $n > 0$ are simply ignored.

The underlying idea is that we do not want to compute only the unifiers of t and s, but rather all the unifiers of the terms t', s' that can (at least potentially) be obtained from t and s after some superposition steps behind the root. For instance the clauses $f(a, x) \simeq a$ and $f(c, b) \simeq c$ would yield the instance $f(a, b)$, although the terms $f(a, x)$ and $f(c, b)$ are not unifiable, because they may become so later in the derivation, e.g. if a is replaced by c by superposition. Similarly the clauses $f(x, x) \simeq a$ and $f(b, c) \simeq c$ should yield the instance $f(b, b) \simeq a$ (or $f(c, c) \simeq a$). On the other hand $f(x)$ and $g(y)$ are still not unifiable, but $f(f(x))$ and $f(g(y))$ are, with an empty unifier, since they can be reduced to the same term by superposition, e.g. $f(a) \simeq a, g(b) \simeq a$. Similarly $f(x, g(x))$ and $f(g(y), y)$ have an empty unifier.

The reason for this rather unusual decision is that we cannot use the superposition calculus to compute an "exact" set of substitutions (as is done in [14]). Indeed, this calculus does not terminate in general on the classes we consider. We need to reason only on the set of terms already occurring in the original clause set, using an over-approximation of the set of (flat) unifiers, which as we shall see is sufficient in our context. The fact that clauses are only instantiated with flat substitutions is not such a limitation since in practice, fresh constants that serve as names for ground terms can be introduced during the flattening operation. Thus, in a sense, the only limitation of this scheme is that instantiations only involve ground terms occurring in the original set of clauses.

Definition 12 Let $t = f(t_1, \ldots, t_n)$ and $s = f(s_1, \ldots, s_n)$ be two terms. We denote by $\sim_{(t,s)}$ the smallest equivalence relation on T_0 for which $t_i \sim_{(t,s)} s_i$, for all $i \in [1..n]$ such that $t_i, s_i \in T_0$.

The *pseudo-unifier* of two terms t, s with the same head symbols is the substitution σ defined as follows: $dom(\sigma) \stackrel{\text{def}}{=} \{x \in \mathcal{V} \mid \exists c \in \Sigma_0, x \sim_{(t,s)} c\}$, and for all $x \in dom(\sigma)$,

$$x\sigma \stackrel{\text{def}}{=} \min_{<}\{c \mid c \in \Sigma_0, x \sim_{(t,s)} c\}.$$ ◊

Example 13 Consider the complex terms $t = f(g(x), a, x, b, x, z, z')$ and $s = f(h(y), y, c, d, b, z', z)$ and the ordering $a \prec b \prec c \prec d \prec x \prec y \prec z' \prec z$. Then there are three equivalence classes: $a \sim y, x \sim c \sim b \sim d$, and $z \sim z'$. Thus, the pseudo-unifier of t and s is $\{x \mapsto b, y \mapsto a\}$. ♣

The relation $\sim_{(t,s)}$ is preserved by a particular class of substitutions:

Proposition 14 *Let t, s be two terms and σ be a substitution that maps every variable in its domain to another variable. If $u \sim_{(t,s)} v$ then $u\sigma \sim_{(t\sigma, s\sigma)} v\sigma$.*

Proof. Immediate. ∎

Complex terms	Involved clauses	Instantiated clause
$\mathtt{car}(x), \mathtt{car}(a)$	(1), (4)	$\mathtt{cons}(\mathtt{car}(a), \mathtt{cdr}(a)) \simeq a$
$\mathtt{cdr}(x), \mathtt{cdr}(b)$	(1), (6)	$\mathtt{cons}(\mathtt{car}(b), \mathtt{cdr}(b)) \simeq b$
$\mathtt{cons}(x, y), \mathtt{cons}(a, c)$	(2), (5)	$\mathtt{car}(\mathtt{cons}(a, c)) \simeq a$
$\mathtt{cons}(x, y), \mathtt{cons}(a, c)$	(3), (5)	$\mathtt{cdr}(\mathtt{cons}(a, c)) \simeq d$

Figure 2: Instantiated clauses of Example 16.

Definition 15 The **Instantiation Rule (I)** is defined as follows:

$$S \to S \cup \{C\sigma\}$$

if the following holds:

- C is a clause in S.
- D is a renaming of a clause in S (possibly C), sharing no variable with C.
- σ is the pseudo-unifier of two terms $t = f(\vec{t})$ and $s = f(\vec{s})$ occurring in C, D respectively.

We denote by $\hat{S}$ the set of clauses that can be generated from S using the instantiation rule above. $\Diamond$

Example 16 Let S denote the set of clauses containing the clauses

$$\begin{array}{llllll} 1: & \mathtt{cons}(\mathtt{car}(x), \mathtt{cdr}(x)) \simeq x, & 2: & \mathtt{car}(\mathtt{cons}(x, y)) \simeq x, & 3: & \mathtt{cdr}(\mathtt{cons}(x, y)) \simeq y, \\ 4: & \mathtt{car}(a) \simeq b, & 5: & \mathtt{cons}(a, c) \simeq d, & 6: & \mathtt{car}(\mathtt{cdr}(b)) \simeq c. \end{array}$$

The clauses other than those in S that are generated by the instantiation scheme are represented in Figure 2. The first column of the table contains the complex terms that are considered for the psuedo-unification, the second column contains the numbers of the clauses in which these terms occur, and the third column contains the instantiated clause. ♣

In general the set $\hat{S}$ is not ground, since it contains S. We assume that Σ contains a special constant symbol λ, not occurring in S; all the variables occurring in $\hat{S}$ are instantiated with this constant.

Definition 17 Given a set of clauses S, we denote by S_λ the set of clauses obtained by replacing all variables in S by λ. $\Diamond$

If C is a clause in S containing n distinct variables and there are m constant symbols occurring in S, then there are at most m^n pseudo-unifiers that can be applied to C. This entails the following result:

Theorem 18 *Given a set of clauses S, let n denote the maximal number of distinct variables appearing in a clause in S, and m denote the number of constants occurring in S. Then the maximal number of clauses that can be generated by the instantiation rule is $O(|S|m^n)$.*

This result is important from a complexity point of view, when considering $\mathcal{T}$-satisfiability problems. Indeed, suppose theory $\mathcal{T}$ is fixed and it is guaranteed that the instantiation scheme is correct for any set of clauses of the form $\mathcal{T} \cup S$, where S is a set of ground unit clauses. Then the number of distinct variables appearing in a clause is a constant, which means that a *polynomial* set of ground clauses is generated. If $\mathcal{T}$ is Horn, then the generated set of ground clauses is also Horn, and its satisfiability can be tested in polynomial time. This result shows an advantage of the instantiation scheme compared to the rewrite-based approach:

contrary to the latter, this scheme avoids the generation of exponentially many clauses in, e.g., the theory of arrays (see [1]). For the theory of records with extensionality [1], it generates a polynomial set of ground Horn clauses, whose satisfiability can be tested in polynomial time, whereas the rewrite-based approach is an exponential satisfiability procedure for this theory.

Before proving correctness results on the instantiation scheme, we provide (without proof) a list of theories on which this scheme can be safely applied (see Section 8 for details): Natural numbers, Integer Offset, Lists, Records, Arrays, Encryption.

The following section is devoted to the definition and properties of various classes of *derivation trees*, which will be used in Section 4 to prove the completeness of our instantiation scheme on particular sets of clauses.

3 Derivation Trees

The main topic of this section is the presentation of so-called *derivation trees* and their properties. These derivation trees allow us to keep track of the variables involved in derivations in a simple way, and by constraining the properties they have to satisfy, we shall be able to devise a sufficient condition guaranteeing the completeness of our instantiation scheme.

3.1 Basic Definitions

Throughout this section, we shall fix the reduction ordering and the selection function that are employed by the superposition calculus.

Definition 19 We denote by $\rightarrow_\sigma$ the relation $\rightarrow^{sel}_{<,\sigma}$, where $<$ is the strongest reduction ordering[3] and *sel* denotes the selection function defined as follows:

- $sel(C) = C$ if C is flat.
- Otherwise, $sel(C)$ is the set of literals in C that are either not flat or not ground[4]. $\Diamond$

This relation corresponds to a (partially) unrestricted version of the calculus (in the sense that the considered ordering is as strong as possible), which is inefficient, but correct.

Definition 20 (Derivation tree) The class of *derivation trees* for a set of clauses S is the smallest set of expressions of the form $\tau = [C, \mathcal{T}, \sigma]$, such that:

- C is a clause, called the *root* of the tree and denoted by $root(\tau)$;
- $\mathcal{T}$ is a (possibly empty) set of pairwise variable-disjoint derivation trees for S;
- σ is a substitution;
- if $\mathcal{T} = \emptyset$ then C is of the form $D\sigma$, where $D \in S$ and σ is a renaming; otherwise $root(\mathcal{T}) \rightarrow_\sigma C$.

The notation $root(\mathcal{T})$ is defined in a standard way as follows:

$$root(\mathcal{T}) \stackrel{\text{def}}{=} \{root(\tau) \mid \tau \in \mathcal{T}\}.$$

A derivation tree $[C, \mathcal{T}, \sigma]$ is *flat* if σ is flat and every derivation tree $\tau \in \mathcal{T}$ is flat. A *refutation tree* is a derivation tree of root $\Box$. $\Diamond$

The set of clauses S may remain implicit, in which case we will simply talk about derivation trees.

[3] i.e., all symbols in Σ are mutually incomparable

[4] Obviously $sel(C)$ contains all the maximal literals in C.

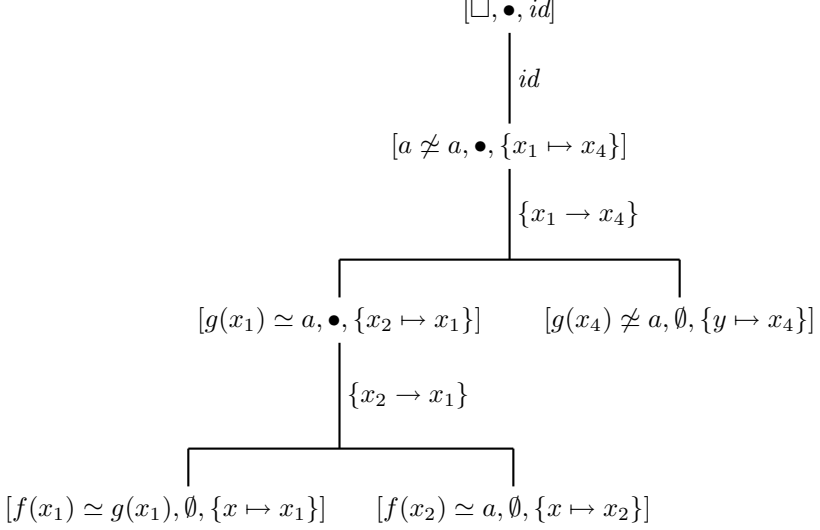

Figure 3: Derivation tree of Example 21

Example 21 Let $S = \{f(x) \simeq g(x),\ f(x) \simeq a,\ g(y) \not\simeq a\}$. Then starting with the derivation trees

$$\tau_1 \ = \ [f(x_1) \simeq g(x_1), \emptyset, \{x \mapsto x_1\}] \ \text{ and } \ \tau_2 = [f(x_2) \simeq a, \emptyset, \{x \mapsto x_2\}],$$

we may construct the derivation tree $\tau_3 = [g(x_1) \simeq a, \{\tau_1, \tau_2\}, \sigma]$, where $\sigma = \{x_2 \to x_1\}$.

Consider $\tau_4 = [g(x_4) \not\simeq a, \emptyset, \{y \mapsto x_4\}]$ and $\sigma' = \{x_1 \to x_4\}$, then $\tau = [a \not\simeq a, \{\tau_3, \tau_4\}, \sigma']$ and $\tau' = [\Box, \{\tau\}, id]$ are derivation trees for S, which is unsatisfiable.

A graphical representation of τ' is provided in Figure 3. The black dots denote the derivation trees immediately below. ♣

When clear from the context, we will keep the substitutions implicit in the graphical representations.

Definition 22 The *depth* of a derivation tree is inductively defined as follows:

$$\mathit{depth}([C, \emptyset, \sigma]) \stackrel{\text{def}}{=} 0, \quad \text{and} \quad \mathit{depth}([C, \mathcal{T}, \sigma]) \stackrel{\text{def}}{=} 1 + \max_{\tau \in \mathcal{T}} \mathit{depth}(\tau).$$

The set of variables occurring in a derivation tree τ, denoted by $V(\tau)$, is the set of variables defined as follows:

$$V([C, \mathcal{T}, \sigma]) \stackrel{\text{def}}{=} V(C) \cup \bigcup_{\tau' \in \mathcal{T}} V(\tau').$$

◊

Example 23 In Example 21, the derivation tree τ' is of depth 3, and the set of variables occuring in τ' is $V(\tau') = \{x_1, x_2, x_4\}$. ♣

We also define the composition of the unifiers over a derivation tree.

Definition 24 Given a derivation tree τ, the substitution μ_τ denotes the composition of the unifiers in τ, which is formally defined as follows:

$$\begin{aligned} \mu_{[C,\emptyset,\sigma]} &\stackrel{\text{def}}{=} id, \\ \mu_{[C,\mathcal{T},\sigma]} &\stackrel{\text{def}}{=} (\bigcup_{\tau \in \mathcal{T}} \mu_\tau)\sigma. \end{aligned}$$

◊

This notation is well-defined since the trees in $\mathcal{T}$ are mutually variable-disjoint. Note that the initial renamings of clauses are not taken into account in the construction of μ_τ. Thus, these substitutions only make sense when applied to the clauses of S that have already been renamed.

Example 25 In Example 21, we have $\sigma = \{x_2 \to x_1\}$ and $\sigma' = \{x_1 \to x_4\}$, hence $\mu_\tau = \sigma\sigma' = \{x_2 \to x_4, x_1 \to x_4\}$, and $\mu_{\tau'} = \mu_\tau$. ♣

We obtain a first result for keeping track of a variable in a derivation tree.

Proposition 26 *Let $\tau = [C, \mathcal{T}, \sigma]$ be a derivation tree. If $x \in V(C)$ then $x \notin dom(\mu_\tau)$.*

Proof. By an immediate induction on the depth of the tree. The variable x cannot occur in $dom(\sigma)$, since by definition of the calculus, σ is applied to the clauses in $\mathcal{T}$ to generated C, and is idempotent. ■

We define the notions of instantiated and uninstantiated hypotheses in a derivation tree. These sets respectively represent the clauses occurring in the original set of clauses, in their instantiated and uninstantiated versions.

Definition 27 Given a derivation tree τ, we denote by $hyp(\tau)$ the set of *uninstantiated hypotheses of* τ, and by $hyp^{\text{inst}}(\tau)$ the set of *instantiated hypotheses of* τ. Formally:

$$\begin{aligned} hyp([C, \emptyset, \sigma]) &\stackrel{\text{def}}{=} hyp^{\text{inst}}([C, \emptyset, \sigma]) \stackrel{\text{def}}{=} \{C\sigma\}, \\ hyp([C, \mathcal{T}, \sigma]) &\stackrel{\text{def}}{=} \bigcup_{\tau \in \mathcal{T}} hyp(\tau), \\ hyp^{\text{inst}}([C, \mathcal{T}, \sigma]) &\stackrel{\text{def}}{=} \bigcup_{\tau \in \mathcal{T}} hyp^{\text{inst}}(\tau)\sigma. \end{aligned}$$

A clause C is a *main hypothesis* of a flat tree τ if $C \in hyp(\tau)$ and if for any variable $x \in V(C)$ and for any unifier σ occuring in τ, we have $x\sigma \in V(C) \cup \Sigma_0$. ♢

Example 28 Suppose $\tau = [\Box, \mathcal{T}, \{x \to a\}]$, where $\mathcal{T} = \{[p(x,b), \emptyset, id], [\neg p(a,y), \emptyset, id]\}$. Then $hyp(\tau) = \{p(x,b), \neg p(a,y)\}$ and $hyp^{\text{inst}}(\tau) = \{p(a,b), \neg p(a,b)\}$. ♣

3.2 Restricted Classes of Derivation Trees

Since the set $\hat{S}$ that is obtained by our instantiation scheme only contains flat instances of S, it is obviously possible for S to be unsatisfiable whereas $\hat{S}$ is satisfiable. For example, this is the case if no refutation tree for S is flat. Even if S admits a flat refutation tree, $\hat{S}_\lambda$ may be satisfiable, as evidenced by the set of clauses $S = \{f(g(x)) \simeq a, g(y) \simeq b, f(z) \not\simeq a\}$. Here, $\hat{S} = S$, and $\hat{S}_\lambda = \{f(g(\lambda)) \simeq a, g(\lambda) \simeq b, f(\lambda) \not\simeq a\}$ is satisfiable. But S admits a flat refutation tree:

$f(b) \simeq a$	(superposition, $y \mapsto x$)
$a \not\simeq a$	(paramodulation, $z \mapsto b$)
$\Box$	(reflexivity)

Such a behaviour may occur even if S contains no function symbol, as evidenced by the set of clauses $S = \{x \not\simeq a, b \simeq a\}$.

In this section, we introduce a semantic criterion on derivation trees that guarantees equisatisfiability, as well as other restrictions that will facilitate the transition from a refutation tree for S to one for $\hat{S}$.

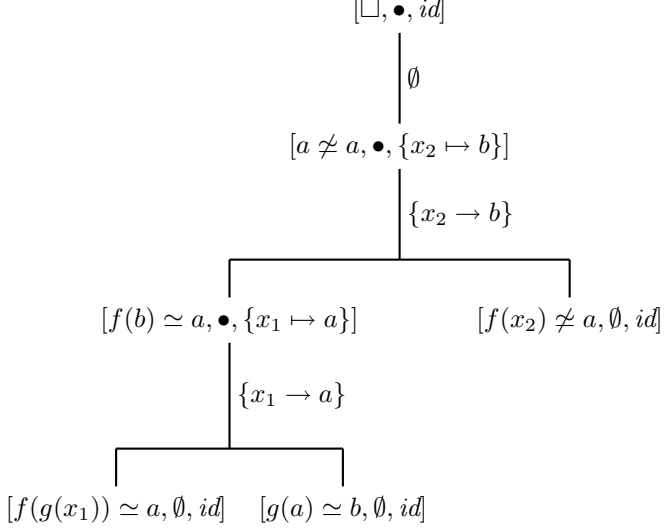

Figure 4: A non-simple derivation tree of root $\Box$.

Definition 29 (Simple Derivation Tree) Let S be a set of clauses. A derivation tree $\tau = [C, \mathcal{T}, \sigma]$ of S, where σ is an mgu of two terms t, s, is *simple* if it satisfies the following conditions:

1. σ is flat.
2. C is quasi-flat (see Section 1.1), and not variable-eligible (see Section 1.2).
3. If there is a position p such that $t|_p = f(t_1, \ldots, t_n)$ (resp. $s|_p = f(t_1, \ldots, t_n)$) and $t_i \in \mathcal{V}$ for some $i \in \mathrm{I_{nv}}(f)$, then either $s|_p$ (resp. $t|_p$) occurs in S or $\sigma = id$.
4. If C is obtained using superposition/paramodulation by replacing a term $u\sigma$ by $v\sigma$, then $v\sigma \notin \mathcal{V}$.
5. Every derivation tree in $\mathcal{T}$ is simple.

A set of clauses S is *simply provable* if for all clauses C, if S admits a derivation tree of root C then S also admits a simple derivation tree of root C. A class of clause sets $\mathfrak{S}$ is *simply provable* if every set of clauses in $\mathfrak{S}$ is simply provable. $\Diamond$

Note that every simple derivation tree is flat, but the converse does not hold. Condition 4 of the definition prevents replacement of a term by a variable which guarantees that inferences preserve Condition 2. Condition 3 is the less natural one. Intuitively it ensures that a variable is never unified with a constant symbol that has previously replaced a complex term by superposition. This allows us to discard derivation trees such as the one in Figure 4. In Figure 4, the term $f(x_2)$ is unified with $f(b)$ but since the latter does not occur in the original set of clauses, we cannot compute the substitution $x \mapsto b$ using the Instantiation rule only. For this purpose, we ensure that the subterms that are not quasi-closed ($f(x_2)$ in this example) are unified only with terms occurring in the original set of clauses.

The main result we shall prove is that if S is simply provable and unsatisfiable, then $\hat{S}_\lambda$ is unsatisfiable. We define a subclass of simple derivation trees that will serve as a link between a derivation tree for S and one for $\hat{S}_\lambda$.

Definition 30 (Pure Derivation Trees) A substitution σ is *pure* if for every variable x, $x\sigma \in \mathcal{V}$. A derivation tree τ is *pure* if it is simple and if μ_τ is pure (this implies that every subtree of τ is pure). $\Diamond$

It will sometimes be useful to replace a subtree in a simple or pure derivation tree by another one. This operation is harmless. Consider for example a derivation tree $\tau = [C, \mathcal{T}, \sigma]$ that is simple (resp. pure),

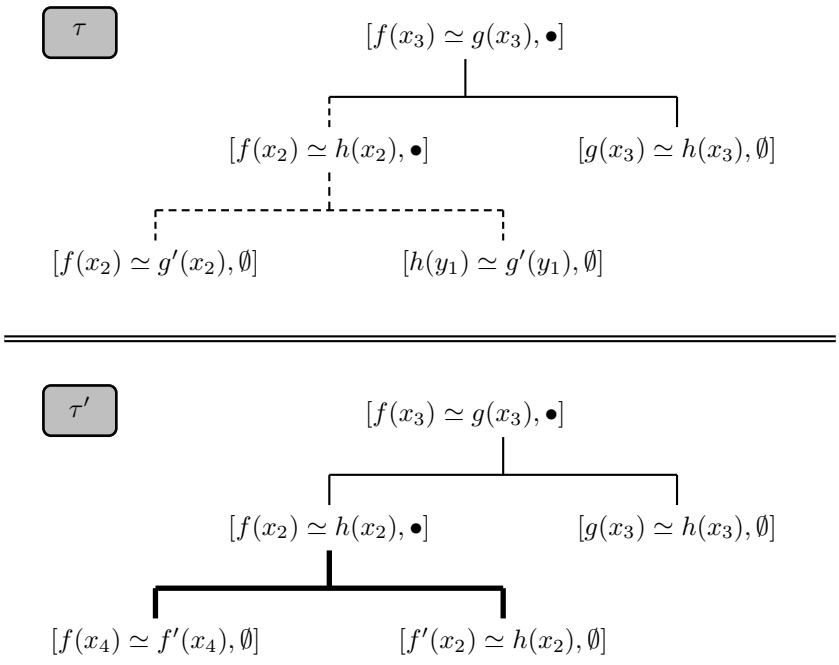

Figure 5: Derivation trees of Example 32 (the substitutions are omitted for the sake of readability)

and let $\tau' \in \mathcal{T}$. Let τ'' be a simple (resp. pure) derivation tree with the same root as τ', and such that the only variables τ and τ'' have in common are those occurring in root(τ''). Then $[C, \mathcal{T}', \sigma]$, where $\mathcal{T}' = (\mathcal{T} \setminus \{\tau'\}) \cup \{\tau''\}$, is a simple (resp. pure) derivation tree with the same root as τ. This result can be generalized to any subtree of τ:

Proposition 31 *Let τ be a simple (resp. pure) derivation tree for a set of clauses S, and $\tau' = [C, \mathcal{T}, \sigma]$ be a subtree of τ. Let τ'' be a simple (resp. pure) derivation tree for S such that:*

- *τ' and τ'' have the same root,*
- *the only variables τ and τ'' have in common are those occurring in C.*

Then the derivation tree obtained by replacing τ' by τ'' in τ is also a simple (resp. pure) derivation tree for S, with the same root as τ.

If τ and τ'' have more variables in common than those in C, then the other variables can of course be renamed, thus yielding a derivation tree that satisfies all the conditions of Proposition 31.

Example 32 Consider the set of clauses

$$S = \{f(x) \simeq f'(x),\ f(x) \simeq g'(x),\ f'(x) \simeq h(x),\ h(x) \simeq g'(x),\ g(x) \simeq h(x)\}.$$

The derivation trees τ (top) and τ' (bottom) of Figure 5 have the same root and are both pure; τ' was obtained from τ by replacing the subtree of root $f(x_2) \simeq h(x_2)$, with the dashed lines, by the subtree with the bold lines. Note that these subtrees only have x_2 as a common variable. ♣

We now define a last class of derivation trees, the most constrained one, which permits only basic inferences to be carried out on a set of clauses. This class will allow us to discard the basic inferences in the construction of a derivation tree.

Definition 33 (Elementary Derivation Tree) A derivation tree $[C, \mathcal{T}, \sigma]$ is *elementary* if $\mathcal{T} = \emptyset$ or $\mathcal{T} = \{\tau, \tau'\}$, where τ is elementary and τ' is pure and has a ground root. $\Diamond$

Note that every elementary tree is pure. We introduce the following measure on derivation trees:

Definition 34 Given a derivation tree τ, $\delta(\tau)$ is defined inductively by:

- $\delta(\tau) \stackrel{\text{def}}{=} 0$ if τ is elementary.
- $\delta([C, \mathcal{T}, \sigma]) \stackrel{\text{def}}{=} 1 + \max_{\tau \in \mathcal{T}} \delta(\tau)$ otherwise. $\Diamond$

In particular, this measure is unaffected by the adjunction of a flat ground clause to a hypothesis in a pure derivation trees:

Proposition 35 *Let $\tau = [C, \mathcal{T}, \sigma]$ be a pure derivation tree for a set of clauses S, $D \in hyp(\tau)$, and consider a flat, ground clause E. Then there exists a pure derivation tree τ' of root $C \vee E$ for $S \cup \{D \vee E\}$, such that $\delta(\tau') = \delta(\tau)$.*

Proof. We prove the result by induction on the depth of τ. If $\mathcal{T} = \emptyset$, then the result is obvious, since necessarily $C = D$. Now assume $\mathcal{T} = \{\tau_1, \tau_2\}$, where D_1, D_2 are the respective roots of τ_1, τ_2, and w.l.o.g., suppose $D \in hyp(\tau_1)$. Then by definition $\{D_1, D_2\} \rightarrow_\sigma C$, and by the induction hypothesis, there is a pure derivation tree τ_1' of root $D_1 \vee E$ for $S \cup \{D \vee E\}$. Since no ordering conditions are considered in the calculus, the literals in E have no influence on the derivation, and it is clear that $\{D_1 \vee E, D_2\} \rightarrow_\sigma C \vee E$. Thus, $\tau' = [C \vee E, \{\tau_1', \tau_2\}, \sigma]$ is a pure derivation tree for $S \cup \{D \vee E\}$, and obviously, $\delta(\tau') = \delta(\tau)$. ∎

3.3 Swapping Variables in Pure Derivation Trees.

Given a pure derivation tree, we will sometimes construct a new derivation tree with the same properties as the original one, by swapping some of its variables, as in the following example.

Example 36 Let $S = \{f(x, y) \simeq g(x, y), f(y, y) \simeq h(y)\}$, and consider the clauses

$$\begin{array}{rcccl} C_1 & = & f(x_1, x_2) & \simeq & g(x_1, x_2), \\ C_2 & = & f(y_1, y_1) & \simeq & h(y_1), \\ C & = & g(y_1, y_1) & \simeq & h(y_1), \end{array}$$

and the substitutions

$$\begin{array}{rcl} \sigma_1 & = & \{x \mapsto x_1,\ y \mapsto x_2\}, \\ \sigma_2 & = & \{y \mapsto y_1\}. \end{array}$$

Let $\tau_1 = [C_1, \emptyset, \sigma_1]$, $\tau_2 = [C_2, \emptyset, \sigma_2]$, and $\sigma = \{x_1 \rightarrow y_1, x_2 \rightarrow y_1\}$; then $\tau = [C, \{\tau_1, \tau_2\}, \sigma]$ is a pure derivation tree for S.

Another pure derivation tree for S with root C can also be constructed by swapping x_1 and y_1. This tree is obtained by taking the clauses

$$\begin{array}{rcccl} C_1' & = & f(y_1, x_2) & \simeq & g(y_1, x_2), \\ C_2' & = & f(x_1, x_1) & \simeq & h(x_1), \end{array}$$

the substitutions

$$\begin{array}{rcl} \sigma_1' & = & \{x \mapsto y_1, y \mapsto x_2\}, \\ \sigma_2' & = & \{y \mapsto x_1\}, \end{array}$$

and the derivation trees $\tau_1' = [C_1', \emptyset, \sigma_1']$ and $\tau_2' = [C_2', \emptyset, \sigma_2']$. Then $\tau' = [C, \{\tau_1', \tau_2'\}, \sigma]$ is also a pure derivation tree for S, and has the same root as τ. ♣

To formalize the intuition of this example, we define the renaming of a derivation tree.

Definition 37 If $\tau = [C, \mathcal{T}, \sigma]$ is a derivation tree and π is a renaming, then $\tau\pi$ denotes the derivation tree defined as follows: $\tau\pi \stackrel{\text{def}}{=} [C\pi, \mathcal{T}\pi, \pi^{-1}\sigma\pi]$. $\Diamond$

The next lemma shows that $\tau\pi$ is indeed a derivation tree with the same properties as τ. We prove the result in the case where τ is a pure derivation tree, along with some additional properties on its structure.

Lemma 38 *Let $\tau = [C, \mathcal{T}, \sigma]$ be a pure derivation tree for S and π be a renaming. Then $\tau\pi$ is a pure derivation tree τ' for S such that $root(\tau') = C\pi$, $\mu_{\tau'} = \pi^{-1}\mu_\tau\pi$, $hyp(\tau') = hyp(\tau)\pi$ and $\delta(\tau') = \delta(\tau)$.*

PROOF. The result is proven by induction on the depth of τ. If $\tau = [C, \emptyset, \sigma]$, then the result is obvious. Now assume $\tau = [C, \mathcal{T}, \sigma]$. Let $\mathcal{T} = \{\tau_1, \tau_2\}$ (we may have $\tau_1 = \tau_2$) and let $\tau'_i \stackrel{\text{def}}{=} \tau_i\pi$ $(i = 1, 2)$. By the induction hypothesis, τ'_1 and τ'_2 are derivation trees, and we have $root(\tau_i) = C_i\pi$, $\mu_{\tau'_i} = \pi^{-1}\mu_{\tau_i}\pi$, $hyp(\tau'_i) = hyp(\tau_i)$, $\delta(\tau'_i) = \delta(\tau_i)$ (for $i = 1, 2$). Then by definition, σ is the mgu of two terms s and t, and $\sigma' = \pi^{-1}\sigma\pi$ is an mgu of $s\pi$ and $t\pi$. Thus, if $\mathcal{T}' = \{\tau'_1, \tau'_2\}$, then $\tau' \stackrel{\text{def}}{=} [C\pi, \mathcal{T}', \sigma']$ is a pure derivation tree for S (since π is a renaming, τ'_1 and τ'_2 do not share any variables). Furthermore, by definition,

$$\begin{aligned}
\mu_{\tau'} &= (\bigcup_{\tau'_i \in \mathcal{T}'} \mu_{\tau'_i})\sigma' \\
&= (\bigcup_{\tau'_i \in \mathcal{T}'} \mu_{\tau'_i})\sigma' \\
&= (\bigcup_{\tau_i \in \mathcal{T}} \pi^{-1}\mu_{\tau_i}\pi)\sigma' \text{ (by the induction hypothesis)} \\
&= \pi^{-1}(\bigcup_{\tau_i \in \mathcal{T}} \mu_{\tau_i})\pi\sigma' \\
&= \pi^{-1}(\bigcup_{\tau_i \in \mathcal{T}} \mu_{\tau_i})\sigma\pi \text{ (because } \sigma' = \pi^{-1}\sigma\pi) \\
&= \pi^{-1}\mu_\tau\pi
\end{aligned}$$

It is simple to verify that $\delta(\tau') = \delta(\tau)$. ∎

The previous lemma shows how pure derivation trees are preserved by renamings. In the sequel, we will sometimes want to rename variables in a pure derivation tree, without modifying the root of this derivation tree. The following lemma provides a sufficient condition for safely performing such an operation.

Lemma 39 *Let $\tau = [C, \mathcal{T}, \sigma]$ be a pure derivation tree for a set of clauses S, let $x \in dom(\sigma)$ and consider $\pi = \{x \to x\sigma, x\sigma \to x\}$. Then $\tau' = [C, \mathcal{T}\pi, \pi^{-1}\sigma]$ is a pure derivation tree for S such that $root(\tau') = C$, $\mu_{\tau'} = \pi^{-1}\mu_\tau$, $hyp(\tau') = hyp(\tau)\pi$ and $\delta(\tau') = \delta(\tau)$.*

PROOF. The proof is immediate if $\mathcal{T} = \emptyset$. By Lemma 38, $\tau'' = [C\pi, \mathcal{T}\pi, \pi^{-1}\sigma\pi]$ is a pure derivation tree for S of root $C\pi$, such that $hyp(\tau'') = hyp(\tau)\pi$, $\mu_{\tau''} = \pi^{-1}\mu_\tau\pi$, $\delta(\tau'') = \delta(\tau)$, where $\pi^{-1}\sigma\pi$ is an mgu of two terms $s\pi$ and $t\pi$. But since π is a renaming, $(\pi^{-1}\sigma\pi)\pi^{-1} = \pi^{-1}\sigma$ is also an mgu of $s\pi$ and $t\pi$, and the clause generated with this mgu is $(C\pi)\pi^{-1} = C$. Thus $\tau' = [C, \mathcal{T}\pi, \pi^{-1}\sigma]$ is a pure derivation tree for S. We have $\mu_{\tau'} = \mu_{\tau''}\pi^{-1} = \pi^{-1}\mu_\tau$. ∎

In Example 36, C_1 and C_2 are hypotheses of τ, but none of them is a main hypothesis. But by swapping variables x_1 and y_1, the new derivation tree we obtain is such that C'_1 is a main hypothesis for this tree. We show that it is possible to generalize this example.

Lemma 40 *Let τ be a pure derivation tree for a set of clauses S, let $D \in hyp(\tau)$ and let u denote a term appearing in D. If θ is a ground substitution such that $dom(\theta) \subseteq V(u)$, then there exists a pure derivation tree τ' for S with the same root as τ, a substitution η, a clause $D' = D\eta \in hyp(\tau')$ and a term u' occurring in D' such that:*

- *$D'\mu_{\tau'} = D\mu_\tau$ and $u'\mu_{\tau'} = u\mu_\tau$,*
- *D' a main hypothesis,*
- *the substitution $\theta' = \eta^{-1}\theta\eta$ corresponding to θ is such that $dom(\theta')\mu_{\tau'} \subseteq dom(\theta')$.*

PROOF. Assume that τ, D, u, θ do not satisfy the property above. In particular, $\eta \stackrel{\text{def}}{=} id$, $D' \stackrel{\text{def}}{=} D$ and $u' \stackrel{\text{def}}{=} u$ cannot be a solution, thus there must exist a variable x such that one of the following conditions holds:

- either $x \in V(D)$ and $x\mu_\tau \notin V(D)$ (i.e. D is not a main hypothesis of τ),
- or $x \in dom(\theta)$ and $x\mu_\tau \notin dom(\theta)$.

We denote by $E(\tau, D, u, \theta)$ the set of variables x satisfying one of these properties. By definition, for every $x \in E(\tau, D, u, \theta)$, we have $x \in dom(\mu_\tau)$, thus there exists a (unique) subtree $\tau_x = [C, \mathcal{T}, \sigma]$ of τ such that $x \in dom(\sigma)$. Let $m(\tau, D, u, \theta) = \{\delta(\tau) - \delta(\tau_x) \mid x \in E(\tau, D, u, \theta)\}$. This measure $m(\tau, D, u, \theta)$ is clearly well-founded, thus we may assume w.l.o.g. that the tuple (τ, D, u, θ) is the minimal one (according to m) such that the above property does not hold.

Let $x \in E(\tau, D, u, \theta)$. Let $\tau_x = [C, \mathcal{T}, \sigma]$ and $x' = x\sigma$. We can safely replace τ_x by the tree obtained as in Lemma 39. Let τ' be the pure derivation tree obtained by replacing τ_x with this new derivation tree. Let $\pi = \{x \mapsto x', x' \mapsto x\}$ and $\theta' \stackrel{\text{def}}{=} \pi^{-1}\theta\pi$. By Lemma 39, $hyp(\tau')$ contains the clause $D' \stackrel{\text{def}}{=} D\pi$. Let $u' \stackrel{\text{def}}{=} u\pi$. We have $D'\mu_{\tau'} = D\pi\mu_{\tau'}$. By Lemma 39, $\mu_{\tau'} = \pi^{-1}\mu_\tau$ thus $D'\mu_{\tau'} = D\mu_\tau$ and $u\mu_\tau = u'\mu_{\tau'}$.

By definition, we have $x \notin E(\tau', D', u', \theta')$ (since x does not occur in D'). Let y be a variable occurring in $E(\tau', D', u', \theta')$ and distinct from x'. We show that $y \in E(\tau, D, u, \theta)$. Since τ' is obtained from τ by swapping x and x' in some part of the tree and since $x\sigma = x'\sigma = x'$, we have $y\mu_{\tau'} = y\mu_\tau$. We assume that $y \notin E(\tau, D, u, \theta)$ to derive a contradiction. We distinguish two possibilities:

- If $y \in V(D')$ and $y\mu_{\tau'} \notin V(D')$, then since $y \neq x, x'$ we have $y \in V(D)$ and $y\mu_\tau \notin V(D')$. If $y\mu_\tau \in V(D)$ this implies that $y\mu_\tau = x$, which is impossible since $x\sigma = x'$.
- If $y \in dom(\theta')$ and $y\mu_{\tau'} \notin dom(\theta')$, then we have $y \in dom(\theta)$ (since $y \neq x, x'$) and $y\mu_\tau \notin dom(\theta')$. If $y\mu_\tau \in dom(\theta)$, then this implies that $y\mu_\tau = x'$, thus $x' \notin dom(\theta')$, i.e. $x \notin dom(\theta)$. This means that both x and x' occur in D (x occurs in D since it is in $E(\tau, D, u, \theta)$ by definition and $x' \in dom(\theta) \subseteq V(D)$). Moreover since $y\mu_\tau = x'$ we have $x'\mu_\tau = x'$ and $x\mu_\tau = x'$. Consequently, x and $x\mu_\tau$ both occur in D and $x\mu_\tau \in dom(\theta)$. This contradicts the fact that $x \in E(\tau, D, u, \theta)$. ∎

Consequently, the only variable that may occur in $E(\tau', D', u', \theta')$ but not in $E(\tau, D, u, \theta)$ is x'. If $x' \in E(\tau', D', u', \theta')$ then $\delta(\tau'_{x'}) > \delta(\tau_x)$. Thus $m(\tau', D', u', \theta') < m(\tau, D, u, \theta)$. Since τ is minimal, there exists a tree τ'' a clause D'' and a term u'' in D'' satisfying the desired properties for τ', D', u'. By transitivity, this also holds for τ, D, u.

3.4 Relations Induced by Derivation trees

We introduce several relations on terms, literals and clauses, along with the properties they satisfy, individually or combined. We start by characterizing the sets of clauses that are invariant by inferences involving empty mgus.

Definition 41 A set of clauses S is Σ_0-*stable* if for every subset $S' \in S$ and for any clause C such that $S' \rightarrow_{id} C$, we have $C \in S$. ◊

Definition 42 Let S be a set of clauses and C be a flat, ground clause. We denote by $\equiv_C^S$ the smallest reflexive relation on constant symbols such that $a \equiv_C^S b$ if there exists a clause $(a \simeq b) \vee D \in S$ where $D \subseteq C$. This relation is extended to every expression (term, atoms or clause) as follows: $f(t_1, \ldots, t_n) \equiv_C^S f(s_1, \ldots, s_n)$ if for every $i \in [1..n]$, $t_i \equiv_C^S s_i$ (f denotes either a function symbol or a logical symbol).

Given two terms $t = f(t_1, \ldots, t_n)$ and $s = f(s_1, \ldots, s_n)$, we write $t \succeq_C^S s$ if for all $i \in [1..n]$, if $t_i \in T_0$ or $s_i \in \mathcal{V}$, then $t_i \equiv_C^S s_i$. $\Diamond$

Intuitively, if e and e' are two ground expressions (terms, literals or clauses) such that $e \equiv_C^S e'$, then every interpretation that satisfies S and falsifies C must interpret e and e' in exactly the same way.

Lemma 43 *Let S be a set of clause and C be a flat ground clause. If S is Σ_0-stable then $\equiv_C^S$ is an equivalence relation.*

Proof. $\equiv_C^S$ is reflexive by definition. Moreover it is symmetric by the commutativity of $\simeq$. We only have to show that it is transitive. Assume that $a \equiv_D^S c$ and $c \equiv_D^S b$, for some terms a, b, c. We show that $a \equiv_D^S b$. The proof is by induction on the depth of the terms. If $a = c$ or $c = b$ the proof is trivial. If $a = f(a_1, \ldots, a_n)$ is complex, then by definition c must be of the form $f(c_1, \ldots, c_n)$ where $\forall i \in [1..n].a_i \equiv_D^S c_i$. Similarly, we have $b = f(b_1, \ldots, b_n)$ and $\forall i \in [1..n].c_i \equiv_D^S b_i$. By the induction hypothesis, we have $\forall i \in [1..n].a_i \equiv_D^S b_i$ hence $a \equiv_D^S b$. If a is flat, then we must have $a, b, c \in \Sigma_0$ and S contains two clauses $(a \simeq c) \vee D_1$ and $(c \simeq b) \vee D_2$ where $D_1, D_2 \subseteq D$. By ground superposition (since the constant are unordered) we can derive $(a \simeq b) \vee D_1 \vee D_2$ from $(a \simeq c) \vee D_1$ and $(c \simeq b) \vee D_2$. Since S is Σ_0-stable, this clause must occur in S. Therefore, $a \equiv_C^S b$. ∎

We prove some simple results on the relation $\succeq_C^S$: this relation is stable by instantiation and inclusion, and it is transitive.

Proposition 44 *Let t, s be terms, C be a flat ground clause, S be a set of clauses and σ be a flat substitution. If $t \succeq_C^S s$, then $t\sigma \succeq_C^S s\sigma$.*

Proof. By definition, we have $t = f(t_1, \ldots, t_n), s = f(s_1, \ldots, s_n)$. Let $i \in [1..n]$. By definition of $\succeq_C^S$, one of the two following conditions holds:

- $t_i \equiv_C^S s_i$. By definition of $\equiv_C^S$, we have either $t_i = s_i$ and in this case $t_i\sigma = s_i\sigma$ or $t_i, s_i \in \Sigma_0$ hence $t_i\sigma = t_i, s_i\sigma = s_i$ and $t_i\sigma \equiv_C^S s_i\sigma$.
- $t_i \notin T_0$ and $s_i \notin \mathcal{V}$. Obviously $t_i\sigma$ cannot be flat and $s_i\sigma$ cannot be a variable. ∎

Proposition 45 *Let t, s be terms, C, D be two ground flat clauses, S be a set of clauses. If $t \succeq_C^S s$ and $C \subseteq D$, then $t \succeq_D^S s$.*

Proof. It suffices to remark that by definition of $\equiv_C^S$, we have: $u \equiv_C^S v \Rightarrow u \equiv_D^S v$. Then the result follows immediately by definition of $\succeq_C^S$. ∎

Proposition 46 *Let C be a flat ground clause and S be a Σ_0-stable set of clauses. Then $\succeq_C^S$ is transitive.*

PROOF. Assume that $t \succeq_C^S s \succeq_C^S u$. By definition of $\succeq_C^S$ we have $t = f(t_1, \ldots, t_n), s = f(s_1, \ldots, s_n), u = f(u_1, \ldots, u_n)$. Let $i \in [1..n]$. If $t_i \equiv_C^S s_i$ and $s_i \equiv_C^S u_i$ then $t_i \equiv_C^S u_i$ by transitivity of $\equiv_C^S$. If $t_i \equiv_C^S s_i$ and $s_i \notin T_0$ then by definition of $\equiv_C^S$ we must have $t_i = s_i$ thus $t_i \notin T_0$. If $t_i \notin T_0$ and $s_i \notin \mathcal{V}$ and $s_i \equiv_C^S u_i$ then we have either $s_i = u_i$ and the proof is obvious, of $s_i, u_i \in \Sigma_0$ thus $u_i \notin \mathcal{V}$. ∎

The relation $\succeq_C^S$ provides a link between any complex term occurring in the root of a pure derivation tree, and the complex terms occurring in the original set of clauses.

Lemma 47 *Let S be a set of clauses and S' be the set of clauses that can be deduced from S using a pure derivation tree. If τ is a pure derivation tree of root C for S and t is a complex term appearing in C, then there exists a clause $C' \in hyp^{\text{inst}}(\tau)$ containing a term s such that $s \succeq_{C^\circ}^{S'} t$.*

PROOF. The proof is by induction on the depth of τ. If $\tau = [C, \emptyset, id]$, then the result is obvious (by taking $C' = C$ and $s = t$). Now assume that $\tau = [C, \mathcal{T}, \sigma]$ and that $t = f(t_1, \ldots, t_n)$ occurs in C. Note that since τ is pure, σ must be pure. By definition of the calculus, one of the following condition holds:

- either $t = t'\sigma$, where t' occurs in a clause $D \in root(\mathcal{T})$ (paramodulation "outside" t' or replacement by t' or equational factoring or reflexivity rule),
- or $t = t'[v]_p\sigma$, where $p \neq \epsilon$ and t' occurs in a clause $D \in root(\mathcal{T})$ (paramodulation "inside" t').

In both cases, since σ is pure, t' must be of the form $f(t'_1, \ldots, t'_n)$. Consider the derivation tree $\tau' = [D, \mathcal{T}', \mu] \in \mathcal{T}$; since τ' is pure, by the induction hypothesis, $hyp^{\text{inst}}(\tau')$ contains a term $s = f(s_1, \ldots, s_n)$ such that $s \succeq_{D^\circ}^{S'} t'$. By definition, $s\sigma \in hyp^{\text{inst}}(\tau)$, and $s\sigma \succeq_{D^\circ}^{S'} t'\sigma$ by Proposition 44. The clause D cannot be flat since it contains s, thus $\text{sel}(D) \cap D^\circ = \emptyset$ and the literals in D° are not affected by the inference step. This implies that $D^\circ \subseteq C^\circ$, and by Proposition 45, $s\sigma \succeq_{C^\circ}^{S'} t'\sigma$.

In Case 1, $t = t'\sigma$ and the result is immediate. In Case 2, if position p is of length strictly greater than 1, then the flat arguments of $t'\sigma$ are not affected by the inference step and it is simple to check that $t'\sigma \succeq_{C^\circ}^{S'} t$; by Proposition 46, $s\sigma \succeq_{C^\circ}^{S'} t$ (since S' is obviously Σ_0-stable). Otherwise, $p = i$ for some $i \in [1..n]$, $root(\tau)$ contains a clause $(u \simeq v) \vee E$, and t is obtained from $f(t'_1, \ldots, t'_n)\sigma$ by replacing $t'_i\sigma = u\sigma$ by $v\sigma$. By Condition 4 in Definition 29, $v\sigma$ cannot be a variable.

- If $u\sigma$ is a complex term, then t'_i must also be a complex term since σ is flat. Since $t'_i \notin T_0$, s_i must also be complex, and the proof is complete.
- Otherwise, since u is eligible, u and v must be constant symbols, and E must be flat. Since $(u \simeq v) \vee E$ is not variable eligible, E must be ground. By definition of the calculus, $E \subseteq C^\circ$, hence $u \equiv_{C^\circ}^{S'} v$ and $s\sigma \succeq_{C^\circ}^{S'} t$. ∎

Given a simple derivation tree τ of root C and a substitution θ, we provide a characterization based on the main hypotheses of τ that guarantees the existence of another simple derivation tree whose root is $C\theta$.

Lemma 48 *Let S be a set of clauses and $\tau = [C, \mathcal{T}, \sigma]$ be a simple derivation tree for S of root C. Consider a clause D that is a main hypothesis of τ and a flat and ground substitution θ such that $dom(\theta) \subseteq V(D)$ and for all variables $x \in dom(\theta)$, $x\theta = x\mu_\tau\theta$. Assume further that there exists a variable $x_0 \in V(D)$ such that $x_0\mu_\tau \in V(C)$. Then there exists a simple derivation tree τ' for $S \cup \{D\theta\}$, of root $C\theta$, such that $\delta(\tau') \leq \delta(\tau)$.*

PROOF. Let $S' = S \cup \{D\theta\}$, we will prove the result by induction on the depth of τ.

If $\mathcal{T} = \emptyset$, necessarily, $C = D$, and $C\theta \in S'$. Thus $\tau' \stackrel{\text{def}}{=} [C\theta, \emptyset, id]$ satisfies the requirements. Now assume that $\mathcal{T} = \{\tau_1, \tau_2\}$, where $root(\tau_1) = C_1 = (t_1 \simeq v) \vee C'_1$ and $root(\tau_2) = C_2[t_2]_p$, and suppose that C is generated by superposition or paramodulation from C_1 into C_2, i.e., $\sigma = mgu(t_1, t_2)$ and $C = (C'_1 \vee C_2[v]_p)\sigma$.

By definition, there exists an $i \in \{1,2\}$ such that $D \in hyp(\tau_i)$. Moreover, $x_0\mu_{\tau_i}$ must occur in the root of τ_i (since $x_0\mu_\tau = x_0\mu_{\tau_i}\sigma \in V(C)$ by hypothesis, and $V(C) \subseteq V(C_1\sigma) \uplus V(C_2\sigma)$).

Let θ' be the restriction of $\sigma\theta$ to the variables in D; note that θ and θ' coincide on all the variables not occurring in $dom(\sigma)$. We check that we can apply the induction hypothesis on the derivation tree τ_i and the substitution θ'. First, it is clear that D is a main hypothesis of τ_i, since τ_i is a subtree of τ. Now let x be a variable occurring in $dom(\theta')$, we show that $x\theta' = x\mu_{\tau_i}\theta'$. If $x\mu_{\tau_i} = x$, then the proof is obvious. If $x\mu_{\tau_i}$ is a constant, then x cannot belong to $dom(\sigma)$ and of course, $x\mu_{\tau_i}\theta' = x\mu_{\tau_i}$. Hence $x \in dom(\theta)$, and by hypothesis, $x\theta = x\mu_\tau\theta$, which entails that

$$x\theta' \;=\; x\sigma\theta \;=\; x\theta \;=\; x\mu_\tau\theta \;=\; x\mu_{\tau_i}\sigma\theta \;=\; x\mu_{\tau_i}.$$

Now assume $x\mu_{\tau_i}$ is a variable other than x, and let $y = x\mu_{\tau_i} \neq x$. Note that $y \in V(D)$ since D is a main hypothesis of τ. Since $y \neq x$, necessarily, $x \notin dom(\sigma)$ (by Proposition 26, because $x \in dom(\mu_{\tau_i})$). Furthermore, by definition, $x\mu_\tau = y\sigma$, thus:

$$x\theta' \;=\; x\sigma\theta \;=\; x\theta \;=\; x\mu_\tau\theta \;=\; y\sigma\theta \;=\; y\theta' \;=\; x\mu_{\tau_i}\theta'.$$

Therefore, we may apply the induction hypothesis on τ_i and θ': there exists a simple derivation tree τ_i' for S', of root $C_i\theta'$, where $\delta(\tau_i') \leq \delta(\tau_i)$.

Let $j = 3 - i$, and let $\tau_j' = \tau_j\eta$ be a renaming of τ_j that contains no variable occurring in τ_i or τ_i'. Then of course, $C_j' = C_j\eta$ is the root of τ_j'. Since σ is a unifier of t_1, t_2 which are variable-disjoint, the substitution σ' such that $\forall x \in V(C_i), x\sigma' \stackrel{\text{def}}{=} x\sigma$ and $\forall x \in V(C_j'), x\sigma' \stackrel{\text{def}}{=} x\eta^{-1}\sigma$ is well-defined, flat, and is an mgu of $t_i, t_j\eta$. We now prove that $t_i\theta$ and $t_j\eta\theta$ are unifiable by verifying that the application conditions of Lemma 1 are satisfied for $t_i, t_j\eta$ and θ. By hypothesis, θ is flat and ground, and $\forall x \in dom(\theta), x\theta = x\mu_\tau\theta$. Furthermore, for all $x \in V(t_i)$, we have $x\mu_\tau = x\sigma$, and since σ and σ' are identical on $V(t_i)$, $x\mu_\tau = x\sigma'$. Thus, for all $x \in dom(\theta)$, $x\theta = x\sigma'\theta$. Consequently, we can apply Lemma 1: $t_i\theta$ and $t_j\eta\theta$ are unifiable, and have an m.g.u. γ such that $dom(\gamma) = dom(\sigma') \setminus dom(\theta)$ and $x\gamma = x\sigma'\theta$ for all $x \in dom(\gamma)$.

Since $dom(\theta) \subseteq V(D)$ and D is a main hypothesis, for all $x \in V(t_i) \cap dom(\theta)$, $x\sigma \in V(D)$ and $x\sigma\theta = x\theta'$. Hence,

$$\begin{aligned} t_j\eta\theta &= t_j\eta, \text{ and} \\ t_i\theta &= t_i\sigma\theta \;=\; t_i\theta'. \end{aligned}$$

Therefore, γ is a flat, ground mgu of $t_i\theta'$ and $t_j\eta$, and the paramodulation rule is applicable on the clauses $C_i\theta'$ and $C_j\eta$.

We now prove that the generated clause is $C\theta$. To this aim, it suffices to show that $C_i\theta'\gamma = C_i\sigma\theta$ and $C_j\eta\gamma = C_j\sigma\theta$ (since we use the unordered version of the calculus the inference step are stable by instantiation).

Let $x \in V(C_i)$. If $x \in V(D)$ then $x\theta' = x\sigma\theta$. By definition of γ, this implies that $x\theta'\gamma = x\sigma\theta$. If $x \notin V(D)$, then $x\theta' = x\theta = x$, hence $x\theta'\gamma = x\gamma = x\sigma'\theta'$. By definition of σ', we have $x\sigma' = x\sigma$. Furthemore, $x\sigma\theta' = x\sigma\theta$.

Now, let $x \in V(C_j)$. By definition $x \notin dom(\theta)$, hence $x\theta = x$. If $x\eta \notin dom(\sigma')$, then $x\eta\gamma = x$. Moreover, this implies that $x \notin dom(\sigma)$ thus $x\sigma\theta = x\theta = x$, and the proof is completed. Otherwise, by definition of γ, $x\eta\gamma = x\sigma\theta$.

Thus $\{C_i\theta', C_j\eta\} \rightarrow_\gamma C\theta$ and the derivation tree $\tau' = [C\theta, \{\tau_1', \tau_2'\}, \gamma]$ satisfies the desired result. This derivation tree is simple since it is obtained by instantiating simple derivation trees by a flat substitution (it is easy to verify that all the conditions in Definition 29 are satisfied).

If τ is nonelementary then $\delta(\tau) = 1 + \max(\delta(\tau_1), \delta(\tau_2)) \geq 1 + \max(\delta(\tau_1'), \delta(\tau_2')) = \delta(\tau')$. Otherwise, since the root of τ_i cannot be ground (since it contains the variable $x_0\mu_{\tau_i}$), τ_i must be elementary. Since $\delta(\tau_i') \leq \delta(\tau_i)$, necessarily, τ_i' is elementary. Furthermore, τ_j must be pure and have a ground root, hence, τ_j' is also pure and has a ground root. Therefore, τ' is elementary and $\delta(\tau) = \delta(\tau') = 0$.

The proof is similar if C is deduced using a unary inference step (reflection or equational factorisation).■

Lemma 49 provides a link between the relations $\equiv_C^S$, $\succeq_C^S$, and the relation employed to define pseudo-unifiers (see Definition 12).

Lemma 49 *Let S be a Σ_0-stable set of clauses, C be a ground clause, and consider the terms s, t, u and v such that:*

- *$u \succeq^S_C t$ and $v \succeq^S_C s$,*
- *t and s are unifiable, with flat unifier σ.*

For all terms $s', t' \in T_0$ such that $s' \sim_{(u,v)} t'$, we have $s'\sigma \equiv^S_C t'\sigma$.

PROOF. By definition of $\succeq^S_C$, s, t, u and v are respectively of the form $f(s_1, \ldots, s_n), f(t_1, \ldots, t_n), f(u_1, \ldots, u_n)$ and $f(v_1, \ldots, v_n)$. We prove the result on induction on $\sim_{(u,v)}$.

- Suppose $t' = u_j$ and $s' = v_j$ for some $j \in [1..n]$. Then $u_j, v_j \in T_0$, and since σ is flat, $u_j\sigma, v_j\sigma \in T_0$. Since $u \succeq^S_C$ and $v \succeq^S_C$, by Proposition 44, $u\sigma \succeq^S_C t\sigma$ and $v\sigma \succeq^S_C s\sigma$, hence $u_j\sigma \equiv^S_C t_j\sigma$ and $v_j\sigma \equiv^S_C s_j$. Since $t_j\sigma = s_j\sigma$, we have the result by transitivity (Lemma 43).
- Suppose $t' \sim_{(u,v)} t''$ and $t'' \sim_{(u,v)}$ for some $t'' \in T_0$. Then by the induction hypothesis, $t'\sigma \equiv^S_C t''\sigma$ and $t''\sigma \equiv^S_C s'\sigma$; by transitivity, $t'\sigma \equiv^S_C s'\sigma$. ∎

4 Completeness of the Instantiation Scheme

This short section is devoted to the proof that if S is simply provable, then S and $\hat{S}_\lambda$ are equisatisfiable. It is clear that if $\hat{S}_\lambda$ is unsatisfiable, then so is S, thus all we need to prove is that the other implication holds too. We prove a stronger result for simple derivation trees:

Theorem 50 *If τ is a simple derivation tree for a set of clauses S, then there exists a pure derivation tree for $\hat{S}$ with the same root as τ.*

PROOF. Let $\tau = [C, \mathcal{T}, \sigma]$ be a simple derivation tree for S. We shall prove by induction on $\delta(\tau)$ that there exists a pure derivation tree τ' for $\hat{S}$, with root C, and such that $\delta(\tau') \leq \delta(\tau)$. If τ is elementary then it is also pure, and there is nothing to prove. Otherwise, C must be deduced from (at most) two clauses D_1, D_2 (we may take $D_1 = D_2$ in case the rule is unary) and we have $\{D_1, D_2\} \to_\sigma C$. Moreover the two simple derivation trees τ_1, τ_2 for S, of respective roots D_1, D_2 are such that $\delta(\tau_i) < \delta(\tau)$ $(i = 1, 2)$.

By the induction hypothesis, there exist two pure derivation trees τ'_1 and τ'_2 of respective roots D_1 and D_2, such that $\delta(\tau'_i) \leq \delta(\tau_i)$ $(i = 1, 2)$. In particular, $D_1, D_2 \in S'$. By definition of the calculus, σ is the mgu of two terms s, t occurring in D_1, D_2 respectively. If σ is pure then the proof is obvious, since $[C, \{\tau'_1, \tau'_2\}, \sigma]$ is a pure derivation tree for $\hat{S}$ and $\delta([C, \{\tau'_1, \tau'_2\}, \sigma]) \leq \delta(\tau)$. We now assume that σ is not pure, which implies that there exists a position p such that exactly one of the two terms $t|_p, s|_p$ is a variable. We consider the case where $t|_p$ is a variable, the other case is symmetrical. W.l.o.g. we suppose that D_1, D_2 are the clauses with a minimal number of variables such that $\{D_1, D_2\} \to_\sigma C$ and there exist pure derivation trees τ'_1, τ'_2 for $\hat{S}$ of roots D_1, D_2 with $\delta(\tau'_i) \leq \delta(\tau_i)$ $(i = 1, 2)$.

The proof proceeds as follows. We show that p is of the form $q.i$, and by applying Lemma 47, we identify terms u', v' appearing in S such that $u'\mu_{\tau'_1} \succeq^{S'}_{C^\circ} t|_q$ and $v'\mu_{\tau'_2} \succeq^{S'}_{C^\circ} s|_q$. Then we consider a substitution θ' based on the pseudo-unifier of u' and v', and apply Lemma 48 and the induction hypothesis to determine a pure derivation tree for $\hat{S}$ of root $D_1\theta' \vee E$, where $E \subseteq C^\circ$. Finally, we will see that $\{D_1\theta' \vee E, D_2\} \to_\sigma C$, thus exhibiting a contradiction with the fact that the number of variables in D_1 is minimal.

Determination of u' and v'. Since σ is flat, $s|_p$ must be a constant symbol. Furthermore, t, s cannot be variables, because D_1, D_2 are not variable-eligible (by Point 2 in Definition 29), and paramodulation into variables is forbidden by definition of the calculus. Thus $p = q.i$, and the terms $t|_q$ and $s|_q$ are of the form $f(t_1, \ldots, t_m)$ and $f(s_1, \ldots, s_m)$ respectively, with $t_i \in \mathcal{V}$ and $s_i \in \Sigma_0$.

By Lemma 47, $hyp^{\text{inst}}(\tau_1')$ and $hyp^{\text{inst}}(\tau_2')$ contain two clauses C_1, C_2 respectively containing terms of the form $u = f(u_1, \ldots, u_m)$ and $v = f(v_1, \ldots, v_m)$ such that $u \succeq^{S'}_{D_1^\circ} f(t_1, \ldots, t_m)$ and $v \succeq^{S'}_{D_2^\circ} f(s_1, \ldots, s_m)$. In particular, note that $u_i = t_i$. By definition of the selection function sel, since D_1, D_2 are not flat, $D_1^\circ \cap sel(D_1) = D_2^\circ \cap sel(D_2) = \emptyset$, and the literals in D_1° and D_2° are not affected by the inference step yielding C. Thus, $D_1^\circ, D_2^\circ \subseteq C^\circ$, and by Proposition 45, we conclude that $u \succeq^{S'}_{C^\circ} f(t_1, \ldots, t_m)$ and $v \succeq^{S'}_{C^\circ} f(s_1, \ldots, s_m)$.

Since τ_1' and τ_2' are pure, there exist clauses C_1' and C_2' in $\hat{S}$ such that for $i \in \{1, 2\}, C_i'\mu_i = C_i$, where $\mu_i = \mu_{\tau_i'}$ is a pure substitution; let $u' = f(u_1', \ldots, u_m')$ and $v' = f(v_1', \ldots, v_m')$ be the terms in C_1', C_2' such that $u'\mu_1 = u$ and $v'\mu_2 = v$, then $u'\mu_1 \succeq^{S'}_{C^\circ} t|_q$ and $v'\mu_2 \succeq^{S'}_{C^\circ} s|_q$. Let θ be the restriction of the pseudo unifier of u' and v' to the variables in u', and θ' be the restriction of $\mu\sigma$ to $dom(\theta)$. By Lemma 40, up to swapping some variables in τ_1', we may assume that C_1' is a main hypothesis of τ_1', and that $dom(\theta')\mu_{\tau_1'} \subseteq dom(\theta')$.

A pure derivation tree of root $D_1\theta'$. We assume that C_1', C_2' share no variable and denote by μ the substitution $\mu = \mu_{\tau_1'} \cup \mu_{\tau_2'}$. If $x \in dom(\theta)$, then $x\theta$ is a constant such that $x \sim_{(u',v')} x\theta$. Thus by Proposition 14, $x\mu \sim_{(u,v)} x\theta$ because μ is pure, and since S' is Σ_0-stable, by Lemma 49,

$$x\mu\sigma \equiv^{S'}_{C^\circ} x\theta. \qquad (\star)$$

By $(\star)$, $C_1'\theta'$ is obtained from $C_1'\theta$ by replacing some constant symbols $c_1, \ldots, c_l$ by constant symbols $c_1', \ldots, c_l'$ such that $\forall j \in [1..l], c_j \equiv^{S'}_{C^\circ} c_j'$. By definition of $\equiv^{S'}_{C^\circ}$, for all $j \in [1..l]$, S' contains a clause of the form $c_j \simeq c_j' \vee E_j$, for some $E_j \subseteq C^\circ$. Clearly, there exists a flat ground clause $E \subseteq C^\circ$ such that the clause $C_1'\theta' \vee E$ is obtained by l applications of the (propositional, unordered) paramodulation rule into $C_1'\theta$ from $c_j \simeq c_j' \vee E_j$ $(1 \leq j \leq l)$. By definition of $\hat{S}$, since $C_1', C_2' \in \hat{S}$ and $\hat{S}$ is closed for the Instantiation rule, $C_1'\theta \in \hat{S}$. Thus S' also contains a clause $C_1'\theta' \vee E$.

Since τ_1' is a pure derivation tree for S' of root D_1 and $C_1' \in hyp(\tau_1')$, by Proposition 35, there also exists a pure derivation tree for $S' \cup \{C_1' \vee E\}$ of root $D_1 \vee E$, with the same measure $\delta(\tau_1')$. Furthermore, since we assumed that C_1' is a main hypothesis of τ_1', we conclude that $C_1' \vee E$ is a main hypothesis of this new tree. We now check that the application conditions of Lemma 48 are satisfied for $D_1 \vee E$, with substitution θ' and main hypothesis $C_1' \vee E$. By definition, θ' is flat and ground, and $dom(\theta') \subseteq dom(\theta) \subseteq V(C_1' \vee E)$. Moreover, $C_1' \vee E$ contains the variable u_i', which is such that $u_i'\mu = u_i = t_i$ occurs in D_1. Now, let $x \in dom(\theta')$, we show that $x\mu_{\tau_1'}\theta' = x\theta'$. We assumed that $dom(\theta')\mu_{\tau_1'} \subseteq dom(\theta')$, which entails that $x\mu_{\tau_1'} = x\mu$ is also in $dom(\theta')$, thus, $x\mu_{\tau_1'}\theta' = x\mu\theta' = x\mu\mu\sigma = x\mu\sigma = x\theta'$ (because μ is idempotent).

Therefore, we may apply Lemma 48: there exists a simple derivation tree τ_1'' for $S' \cup \{C_1'\theta' \vee E\} = S'$ of root $D_1\theta' \vee E$, and such that $\delta(\tau_1'') \leq \delta(\tau_1') < \delta(\tau)$. By the induction hypothesis, $D_1\theta' \vee E$ admits a pure derivation tree τ_p for $\hat{S}$ such that $\delta(\tau_p) \leq \delta(\tau_1'') < \delta(\tau)$.

Exhibiting the contradiction. Let x be a variable in D_1, then by Proposition 26, $x \notin dom(\mu)$ (since $\mu = \mu_{\tau_1} \cup \mu_{\tau_2}$). By definition, θ' is the restriction of $\mu\sigma$ to $dom(\theta)$; hence, if $x \in dom(\theta)$ then $x\theta' = x\mu\sigma = x\sigma$, and $x\theta'\sigma = x\sigma\sigma = x\sigma$, since σ is idempotent. If $x \notin dom(\theta)$ then $x\theta' = x$, and $x\theta'\sigma = x\sigma$. Thus $\{D_1\theta' \vee E, D_2\} \to_\sigma C$ (since $E \subseteq C^\circ$).

Since $v \succeq^{S'}_{C^\circ} f(s_1, \ldots, s_m)$, by definition, if $v_i \in T_0$, then $v_i \equiv^{S'}_{C^\circ} s_i$. Thus, if $v_i \in T_0$, then it must be a constant, since s_i is a constant. If $v_i \notin T_0$, then, since every clause occurring in τ is quasi-flat by Condition 2 of Definition 29, index i cannot be in $\mathrm{I}_0(f)$. Thus, this index is necessarily in $\mathrm{I}_{\mathrm{nv}}(f)$. Since $t_i \in \mathcal{V}$ and $\sigma \neq id$, by Condition 3 of Definition 29, s must occur in the initial clause set S, and we can safely replace C_2' with the clause containing s. Hence, we may assume w.l.o.g. that v_i is a constant, and therefore that v_i', which is such that $v_i = v_i'\mu_{\tau_2'}$, is also a constant.

Since $t_i \in \mathcal{V}$ and $u \succeq^{S'}_{C^\circ} f(t_1, \ldots, t_m)$, we have $u_i = t_i$, i.e., $u_i'\mu = t_i$. In particular, u_i' is a variable, and since v_i' is a constant, by definition of a pseudo-unifier, $u_i'\theta \in \Sigma_0$. Now, by $(\star)$, $u_i'\mu\sigma \equiv^{S'}_{C^\circ} u_i'\theta$,

which means that $u'_i\mu\sigma = u'_i\theta'$ is a constant. Since μ is idempotent, $u'_i\theta' = u'_i\mu\sigma = u'_i\mu\mu\sigma = t_i\theta'$, hence $t_i\theta' \in \Sigma_0$. Consequently, $D_1\theta'$ is a strict instance of D_1, which contradicts the fact that the number of variables in D_1 is minimal. ■

We may therefore prove the completeness of the instantiation scheme for the class of simply provable sets of clauses:

Corollary 51 *If S is a simply provable and unsatisfiable set of clauses then $\hat{S}_\lambda$ is unsatisfiable.*

PROOF. This is a simple consequence of Theorem 50: since S is simply provable and unsatisfiable, it admits a simple derivation tree τ with root $\square$. By Theorem 50, $\hat{S}$ admits a pure derivation tree τ' with the same root, and by instantiating all the variables in τ' by λ, we obtain a refutation tree for the set $\hat{S}_\lambda$ which is therefore unsatisfiable. ■

Although we have proved the completeness of the scheme for the class of simply provable sets of clauses, there remains the issue of being able to detect such sets of clauses, since this condition is semantic. The following two sections are devoted to the description of syntactic conditions that will guarantee this condition.

5 Syntactic Characterization of Simply Provable Clause Sets

The goal of this section is to introduce a first syntactic characterization of simply provable clause sets. This characterization is obtained by defining the class of variable-preserving sets of clauses, that cannot generate any variable-eligible clause, and the subclass of controlled sets of clauses, that are always simply provable.

In order to determine syntactic conditions that are general enough to be satisfied by several theories of interest, we need to restrain the potential inferences that can take place in a derivation tree. This is done by considering a selection function that is more restricted than that of Definition 19.

In this section and the following one, we consider a selection function satisfying the following properties:

Definition 52 We consider a selection fuction sel such that:

1. For every clause C and every literal L in C, $L \in \text{sel}(C)$ if there exists a ground substitution σ of the variables in C such that $L\sigma \in \text{sel}(C\sigma)$.
2. If L is a positive literal in C and if L' is a literal in C such that $L < L'$ then L is not in $\text{sel}(C)$.
3. If L is flat and ground and C is not flat and ground, then $L \notin \text{sel}(C)$.
4. I will need the selection function (or the ordering?) to guarantee that if the maximal terms in the literals are constants, then the clauses are flat and ground. ◊

Note that this selection function is not intended to be employed in practice: in this context, it is used to show the *existence* of simple derivation trees

5.1 Preservation of Quasi-flatness and Quasi-closedness

Recall from Definition 4 that some of the arguments of a quasi-flat term must have a limited depth, and that some of the arguments of a quasi-closed term cannot be variables. We prove that quasi-flatness and quasi-closedness are preserved by flat substitutions and by replacements.

Lemma 53 *Let σ be a flat substitution. If t is a quasi-flat (resp. quasi-closed) term then $t\sigma$ is quasi-flat (resp. quasi-closed).*

PROOF. Since σ is flat, any complex subterm in $t\sigma$ is of the form $f(t_1, \ldots, t_n)\sigma$, where $f(t_1, \ldots, t_n)$ occurs in t. Moreover, we have obviously $t_i \in T_0 \Rightarrow t_i\sigma \in T_0$ (since σ is flat) and $t_i \notin \mathcal{V} \Rightarrow t_i\sigma \notin \mathcal{V}$. Thus the conditions of Definition 4 are preserved by substitution. ∎

Lemma 54 *Let t, s be two non variable terms and let p be a position in t such that $t|_p \in \Sigma_0 \Rightarrow s \in \Sigma_0$. If t, s are quasi-flat (resp. quasi-closed) then $t[s]_p$ is quasi-flat (resp. quasi-closed).*

PROOF. We prove the result by induction on the length of p. If $p = \varepsilon$, then the result is obvious, since $t[s]_p = s$. Otherwise, $p = i.q$, which means that $t = f(t_1, \ldots, t_n)$ and $t[s]_p = f(t_1, \ldots, t_{i-1}, t_i[s]_q, t_{i+1}, \ldots, t_n)$. Since t is quasi-flat (resp. quasi-closed), so is t_i; furthermore, $t|_p \in \Sigma_0 \Rightarrow t_i|_q \in \Sigma_O$. We can therefore apply the induction hypothesis: $t_i[s]_q$ is quasi-flat (resp. quasi-closed), and so is $t[s]_p$. ∎

The next lemma states that the m.g.u. of two quasi-closed and quasi-flat terms is necessarily flat.

Lemma 55 *Let s and t be two quasi-flat and quasi-closed terms that are unifiable, and let σ be an m.g.u. of s and t. Then σ is flat.*

PROOF. By definition, neither s nor t can be a variable, and if they are both constants then the result is obvious. Now assume that $s = f(s_1, \ldots, s_n)$, and $t = f(t_1, \ldots, t_n)$, we prove the result by induction on the size of s.

Let $\theta_0 = \emptyset$, and for $i \in \{1, \ldots, n\}$, let μ_i be an m.g.u. of $\{s_i\theta_{i-1} =^? t_i\theta_{i-1}\}$ and $\theta_i = \theta_{i-1}\mu_i$. Obviously, θ_n is an m.g.u. of $\{s =^? t\}$. We prove that for all $i \in \{1, \ldots, n\}$, μ_i is flat. Since θ_0 is trivially flat, a simple induction using Proposition 2 shows that every θ_i is also quasi-flat. Suppose θ_{i-1} is flat. Then by Lemma 53 $s\theta_{i-1}$ and $t\theta_{i-1}$ are both quasi-flat and quasi-closed, and there are two cases to consider.

- If $s_i\theta_{i-1}$ is of the form $g(u_1, \ldots, u_m)$, where $m \geq 1$, then i cannot be in $\mathrm{I}_0(f)$ since $s\theta_{i-1}$ is quasi-flat. Thus $i \in \mathrm{I}_{\mathrm{nv}}(f)$, since $\mathrm{I}_0(f) \cup \mathrm{I}_{\mathrm{nv}}(f) = [1..n]$ (see Definition 4). Since $t\theta_{i-1}$ is quasi-closed, this implies that $t_i\theta_{i-1}$ is not a variable, thus it is also of the form $g(v_1, \ldots, v_m)$. Since the size of $s_i\theta_{i-1}$ is strictly smaller than that of s, by the induction hypothesis, μ_i is quasi-flat.
- If $s_i\theta_{i-1}$ is a variable, then $i \in \mathrm{I}_0(f)$, since $s\theta_{i-1}$ is quasi-closed. Since $t\theta_{i-1}$ is quasi-flat $t_i\theta_{i-1}$ is either a variable or a constant. Thus, $\mu_i = \{s_i\theta_{i-1} \mapsto t_i\theta_{i-1}\}$ is flat. The case where $t_i\theta_{i-1}$ is a variable is similar. Finally, if $t_i\theta_{i-1}$ and $s_i\theta_{i-1}$ are both constant symbols then $\mu_i = id$ and $\theta_i = \theta_{i-1}$. ∎

5.2 Variable-Preserving Clause Sets

In the previous section, we assumed that no clause occurring in a derivation tree was variable-eligible. We introduce a set of syntactic criteria that ensure this property is satisfied. Other approaches ensuring the absence of variable-eligible clauses in derivations include [1, 19, 21]. Our approach consists in defining syntactic conditions that are tested on the original set of clauses, to guarantee the required property. This is done by defining variables that are *instantiable* in a clause, a condition that depends on the positions of their occurrences in the clause. Intuitively, no matter the inference, an instantiable variable cannot cause the generated clause to be variable-eligible.

Definition 56 We associate to each function symbol f of arity n a set of indices $\mathrm{Inst}(f)$ in $[1..n]$. A variable x is *instantiable* in a term t if t is of the form $f(t_1, \ldots, t_n)$ and there exists an $i \in [1..n]$ such that:

- either $i \in \mathrm{Inst}(f)$ and $t_i = x$,
- or x is instantiable in t_i.

A variable x is *instantiable in a literal* L if L is of the form $t \simeq s$ or $t \not\simeq s$ and x is instantiable in t or in s. A variable is *instantiable in a clause* C if it is instantiable in at least one literal in C. If e is an expression (term, literal, or clause), we denote by $I_{\mathcal{V}}(e)$ the set of variables that are instantiable in e. ◊

Example 57 Let $\Sigma = \{f, g\}$, where f is of arity 2 and g of arity 1, and suppose $\mathrm{Inst}(f) = \mathrm{Inst}(g) = \{1\}$. If $C = x \simeq y \vee f(g(x), y) \simeq f(g(x), x)$, then x is instantiable in C and y is not. ♣

In particular, if for all $n \in \mathbb{N}$ and for all $f \in \Sigma_n$ we have $\mathrm{Inst}(f) = [1..n]$, and if e is a nonvariable expression, then $I_{\mathcal{V}}(e) = V(e)$. If $\mathrm{Inst}(f) = \emptyset$ for all $f \in \Sigma$, then $I_{\mathcal{V}}(e) = \emptyset$.

Proposition 58 *If s is a subterm of t, then $I_{\mathcal{V}}(s) \subseteq I_{\mathcal{V}}(t)$.*

Lemma 59 *Let t be a term and σ be a flat substitution. Then $I_{\mathcal{V}}(t\sigma) = I_{\mathcal{V}}(t)\sigma \cap \mathcal{V}$.*

Proof. We prove the result by induction on the depth of t. Note that t and $t\sigma$ must have the same depth, since σ is flat. If t is a variable or a constant, then by Definition 56, $I_{\mathcal{V}}(t) = \emptyset$. Moreover $t\sigma$ is also flat, thus $I_{\mathcal{V}}(t\sigma) = \emptyset$.

If $t = f(t_1, \ldots, t_n)$ then $I_{\mathcal{V}}(t) \stackrel{\text{def}}{=} \{t_i \mid i \in \mathrm{Inst}(f), t_i \in \mathcal{V}\} \cup \bigcup_{i=1}^{n} I_{\mathcal{V}}(t_i)$, and $I_{\mathcal{V}}(t\sigma) \stackrel{\text{def}}{=} \{t_i\sigma \mid i \in \mathrm{Inst}(f), t_i\sigma \in \mathcal{V}\} \cup \bigcup_{i=1}^{n} I_{\mathcal{V}}(t_i\sigma)$. If $t_i\sigma$ is a variable, then so is t_i, thus $\{t_i\sigma \mid i \in \mathrm{Inst}(f), t_i\sigma \in \mathcal{V}\} = \{t_i \mid i \in \mathrm{Inst}(f), t_i \in \mathcal{V}\}\sigma \cap \mathcal{V}$. Also, by the induction hypothesis, for $i \in [1..n]$, $I_{\mathcal{V}}(t_i\sigma) = I_{\mathcal{V}}(t_i)\sigma \cap \mathcal{V}$. Consequently $I_{\mathcal{V}}(t\sigma) = I_{\mathcal{V}}(t)\sigma \cap \mathcal{V}$. ■

We obtain as a simple consequence:

Corollary 60 *Let t, s be two terms such that $I_{\mathcal{V}}(t) = I_{\mathcal{V}}(s)$ and let σ be a flat substitution. Then $I_{\mathcal{V}}(t\sigma) = I_{\mathcal{V}}(s\sigma)$.*

The sets of variable-preserving literals and clauses are defined by imposing constraints that ensure instantiable variables remain so.

Definition 61 A literal L is *variable-preserving* in a clause $C = L \vee D$ if:

1. L is a negative literal $t \not\simeq s$, and one of the following conditions is satisfied:
   (a) s (resp. t) is a variable occurring in $I_{\mathcal{V}}(t)$ (resp. $I_{\mathcal{V}}(s)$);
   (b) $I_{\mathcal{V}}(t) = \emptyset$ or $I_{\mathcal{V}}(s) = \emptyset$;
   (c) $I_{\mathcal{V}}(t) \cup I_{\mathcal{V}}(s) \subseteq I_{\mathcal{V}}(D)$;
2. L is a positive literal $t \simeq s$, and one of the following conditions is satisfied:
   (a) $t, s \notin \mathcal{V}$ and $I_{\mathcal{V}}(t) = I_{\mathcal{V}}(s)$;
   (b) $\{t, s\} \subseteq T_0$ and $\{t, s\} \cap \mathcal{V} \subseteq I_{\mathcal{V}}(D)$.

A clause C is *variable-preserving* if every literal $L \in C$ is variable-preserving in C, and a set of clauses S is *variable-preserving* if every clause in S is variable-preserving. ◊

This notion is close to that of a variable-inactive clause, which is defined in [1], since it is meant to prevent clauses from being variable-eligible. However, it is more general than the latter, as evidenced by the following example inspired from [21]:

Example 62 Consider the clause $C = f(x) \simeq g(x) \vee x \simeq y \vee f(y) \simeq g(y)$. This clause is not variable-inactive because of the literal $x \simeq y$. Let $\mathrm{Inst}(f) = \mathrm{Inst}(g) = \{1\}$, then $I_{\mathcal{V}}(f(x)) = I_{\mathcal{V}}(g(x)) = \{x\}$ and $I_{\mathcal{V}}(f(y)) = I_{\mathcal{V}}(g(y)) = \{y\}$. The first and third literal of C satisfy Condition (2a), and the second literal satisfies Condition (2b). This clause is therefore variable-preserving. ♣

The main property of interest satisfied by variable-preserving clauses is the following:

Proposition 63 *All variable-preserving clauses are non-variable-eligible.*

Proof. Let C be a variable-preserving clause. Assume that C is of the form $x \simeq t \vee D$, where x is a variable such that $x \not\leq t$, and assume that $x \simeq t$ is an eligible literal. Then by Definition 61, one of Conditions (2a) or (2b) must hold. However, x is a variable, hence Condition (2a) cannot hold. Thus, Condition (2b) holds, $t \in T_0$ and $x \in I_{\mathcal{V}}(D)$. Furthermore, if t is a variable, then $t \in I_{\mathcal{V}}(D)$.

Since $x \simeq t \in \mathrm{sel}(D)$, there exists a ground substitution σ such that $\forall L \in D$, $x\sigma \simeq t\sigma \not\leq L\sigma$ by Condition 1 of the selection function, see Definition 52). Now since $x \in I_{\mathcal{V}}(D)$, there exists a term of the form $f(t_1, \ldots, t_n)$ occurring in D such that $t_i = x$. Thus, $x\sigma < f(t_1, \ldots, t_n)\sigma$ and necessarily, $t\sigma \not\leq f(t_1, \ldots, t_n)\sigma$ (and $x\sigma < t\sigma$). This implies that $t\sigma$ cannot be a constant, and since $t \in T_0$, we must have $t \in \mathcal{V}$. But in this case, t must also be in $I_{\mathcal{V}}(D)$, which means that t occurs at a non-root position in a literal $L' \in D$. But then $(x \simeq t)\sigma < L'\sigma$, which is impossible. ■

The following lemmata provide exhibit conditions that guarantee variable-perservtion is maintained by operations such as disjunctions, instantiations and replacements.

Proposition 64 *Let C, D be two variable-preserving clauses. Then $C \vee D$ is variable-preserving.*

Definition 65 A literal L is *dominated* in a clause $L \vee C$ if $I_{\mathcal{V}}(L) \subseteq I_{\mathcal{V}}(C)$. ◊

Lemma 66 *If L is dominated in a clause $L \vee C$ and $L \vee C$ is variable-preserving, then C is variable-preserving.*

Proof. Assume that C is not variable-preserving. Then there exists a literal $L' \in C$ that is not variable-preserving in C. Since L' is variable-preserving in $L \vee C$, this implies by Definition 61 that one of Conditions (1c) or (2b) does not hold for the clause C. Hence, there exists a variable x occurring in $I_{\mathcal{V}}(L)$ but not in $I_{\mathcal{V}}(C)$. But this is impossible since L is dominated. ■

Lemma 67 *Let C be a variable-preserving clause and let σ be a flat substitution. Then $C\sigma$ is also variable-preserving.*

Proof. We have to show that for all $L \in C$, $L\sigma$ is variable-preserving in $C\sigma$. Let $C = L \vee D$. Since C is variable-preserving, L is variable-preserving in C, thus one of the following conditions holds:

- L is of the form $t \not\simeq x$ (or $x \not\simeq t$), where $x \in I_{\mathcal{V}}(t)$. Since σ is flat, $x\sigma$ is either a variable or a constant symbol. If $x\sigma \in \mathcal{V}$ then by Lemma 59, $x\sigma \in I_{\mathcal{V}}(t\sigma)$ thus $t\sigma \not\simeq x\sigma$ is variable-preserving in $C\sigma$. Otherwise $x\sigma$ is ground hence $I_{\mathcal{V}}(x\sigma) = \emptyset$ and $t\sigma \not\simeq x\sigma$ is also variable-preserving by Condition (1b).

- L is of the form $t \not\simeq s$, where $I_{\mathcal{V}}(t) = \emptyset$ or $I_{\mathcal{V}}(s) = \emptyset$. Say $I_{\mathcal{V}}(t) = \emptyset$. Then by Lemma 59 we have $I_{\mathcal{V}}(t\sigma) = \emptyset$, hence $L\sigma$ is variable-preserving in $C\sigma$ by Condition (1b).

- L is of the form $t \not\simeq s$, where $I_{\mathcal{V}}(t) \cup I_{\mathcal{V}}(s) \subseteq I_{\mathcal{V}}(D)$. Let y be a variable in $I_{\mathcal{V}}(t\sigma) \cup I_{\mathcal{V}}(s\sigma)$. By Lemma 59, $y = x\sigma$ for some $x \in I_{\mathcal{V}}(t) \cup I_{\mathcal{V}}(s)$. Then $x \in I_{\mathcal{V}}(D)$ by hypothesis, and by Lemma 59, $x\sigma = y \in I_{\mathcal{V}}(D\sigma)$. Thus Condition (1c) holds and $L\sigma$ is variable-preserving in $C\sigma$.

- L is of the form $t \simeq s$, where $t, s \notin \mathcal{V}$ and $I_{\mathcal{V}}(t) = I_{\mathcal{V}}(s)$. By Corollary 60, $I_{\mathcal{V}}(t\sigma) = I_{\mathcal{V}}(s\sigma)$. Moreover since $t, s \notin \mathcal{V}$, necessarily, $t\sigma, s\sigma \notin \mathcal{V}$ Therefore $L\sigma$ is variable-preserving in $C\sigma$.

- If L is of the form $t \simeq s$, where $t, s \in T_0$ and $\{t, s\} \cap \mathcal{V} \subseteq I_{\mathcal{V}}(D)$. Then since σ is flat, $t\sigma, s\sigma \in T_0$. Let $x \in \{t\sigma, s\sigma\} \cap \mathcal{V}$. Obviously, x is of the form $y\sigma$ for some $y \in \{t, s\}$, and $y \in I_{\mathcal{V}}(D)$ by hypothesis. Hence, $x \in I_{\mathcal{V}}(D\sigma)$ by Lemma 59. ∎

We show that the sets $I_{\mathcal{V}}(t)$ are stable by replacement.

Lemma 68 *Let t, s be two terms, p be a position in t, and assume that $t|_p$ and s are not variables. If $I_{\mathcal{V}}(s) = I_{\mathcal{V}}(t|_p)$ then $I_{\mathcal{V}}(t[s]_p) = I_{\mathcal{V}}(t)$.*

PROOF. We prove the result by induction on the length of p. Note that since $t|_p$ is not a variable, t cannot be a variable.

If $p = \epsilon$, then $t|_p = t$ and $t[s]_p = s$. Thus we have $I_{\mathcal{V}}(t) = I_{\mathcal{V}}(t[s]_p) = I_{\mathcal{V}}(s)$.

If $p = i.q$, then t is of the form $f(t_1, \ldots, t_n)$, thus $t|_p = t_i|_q$ and $t[s]_p = f(t_1, \ldots, t_{i-1}, t_i[s]_q, t_{i+1}, \ldots, t_n)$. Since $t|_p$ and s are not variables, t_i and $t_i[s]_q$ cannot be variables. By the induction hypothesis $I_{\mathcal{V}}(t_i[s]_q) = I_{\mathcal{V}}(t_i)$, hence by Definition 56,

$$\begin{aligned} I_{\mathcal{V}}(t[s]_p) &= \{t_j \mid j \in \mathrm{Inst}(f), t_j \in \mathcal{V}\} \cup \bigcup_{j \in [1..n] \setminus \{i\}} I_{\mathcal{V}}(t_j) \cup I_{\mathcal{V}}(t_i[s]_p) \\ &= \{t_j \mid j \in \mathrm{Inst}(f), t_j \in \mathcal{V}\} \cup \bigcup_{j \in [1..n]} I_{\mathcal{V}}(t_j) \\ &= I_{\mathcal{V}}(t). \end{aligned}$$

∎

Lemma 68 implies that variable-preserving clauses are also stable by replacement:

Lemma 69 *Let C be a variable-preserving clause, $u = C|_p$ be a non-variable term occurring in C and v be a non-variable term such that:*

- *$I_{\mathcal{V}}(u) = I_{\mathcal{V}}(v)$,*
- *if $u \in \Sigma_0$ then $v \in \Sigma_0$.*

Then $C[v]_p$ is variable-preserving.

PROOF. Let L be the literal containing the term t such that $u = t|_q$ for some position q. By Lemma 68, $I_{\mathcal{V}}(t) = I_{\mathcal{V}}(t[v]_p)$, and since u and v are not variables, it is simple to check that if L satisfies one of Conditions (1a)-(2a), then so does the literal obtained after replacing u by v. If L satisfies Condition (2b), then necessarily u is a constant, hence by hypothesis, so is v. Again, the literal obtained after replacing u by v satisfies Condition (2b). ∎

The previous results allow us to prove the stability of variable-preserving clauses for the calculus.

Theorem 70 *Let S be a variable-preserving clause set. Assume that $S' \rightarrow^{\text{sel}}_{<,\sigma} C$, where all the clauses in S' are pairwise variable-disjoint renamings of clauses in S, and suppose that σ is flat. Then C is also variable-preserving.*

PROOF. We distinguish several cases according to the rule used to derive C. Note that in all cases, by Proposition 63, none of the premisses of the inference step are variable-eligible.

C is generated by the superposition or paramodulation rule. This means that C is of the form $D_1[v]_p\sigma \vee D_2'\sigma$, where D_1 and $D_2 = u \simeq v \vee D_2'$ are clauses in S', $t = D_1|_p$, and $\sigma = mgu(t, u)$. Since D_1 and D_2 are variable-preserving, by Lemma 67, $D_1\sigma$ and $D_2\sigma$ are also variable-preserving. Since there is no superposition/paramodulation into variables, $D_1|_p \notin \mathcal{V}$, and since $u \simeq v \vee D_2'$ is not variable-eligible, $u \notin \mathcal{V}$. Since D_2 is variable-preserving, one of Conditions (2a) or (2b) must hold for literal $u \simeq v$.

- If Condition (2a) holds, then $I_\mathcal{V}(u) = I_\mathcal{V}(v)$, and v cannot be a variable. By definition of the ordering, if $u\sigma$ is a constant, then $v\sigma$ must also be a constant (since constants are strictly smaller than complex terms).
- If Condition (2b) holds, then u is necessarily a constant. If v were a variable, then we would have $v \in I_\mathcal{V}(D')$. Thus, D' would contain a term $f(t_1, \ldots, t_n)$ of which v is a subterm, and we would have $u < f(t_1, \ldots, t_n)$, contradicting the fact that $u \simeq v$ is a maximal literal in D_2. Thus, v must be a constant, and $I_\mathcal{V}(u) = I_\mathcal{V}(v) = \emptyset$.

By Lemma 67, $D_1\sigma$ is variable-preserving. Since σ is an mgu of u and t, we have $I_\mathcal{V}(t\sigma) = I_\mathcal{V}(u\sigma)$, and by Corollary 60, $I_\mathcal{V}(t\sigma) = I_\mathcal{V}(v\sigma)$. Therefore, by Lemma 69, $D_1[v]_p\sigma$ is variable-preserving, and by Proposition 64, $D_1[v]_p\sigma \vee D_2\sigma$ is also variable-preserving.

Since $I_\mathcal{V}(u\sigma) = I_\mathcal{V}(v\sigma)$ and $v\sigma$ occurs in $D_1[v]_p\sigma$, by Proposition 58, $(u \simeq v)\sigma$ is dominated in $D_1[v]_p\sigma \vee D_2\sigma$. By Lemma 66, $D_1[v]_p\sigma \vee D_2'\sigma$ is variable-preserving.

C is generated by the reflection rule. C is of the form $D\sigma$, where S' contains a clause of the form $t \not\simeq s \vee D$ and $\sigma = mgu(t, s)$. By Lemma 67, $(t \not\simeq s \vee D)\sigma$ is variable-preserving, we show that $t\sigma \not\simeq s\sigma$ is dominated in $(t \not\simeq s \vee D)\sigma$.

Let $x \in I_\mathcal{V}(t\sigma) \cup I_\mathcal{V}(s\sigma)$; by Lemma 59, $x = y\sigma$, where $y \in I_\mathcal{V}(t) \cup I_\mathcal{V}(s)$. Since $t \not\simeq s$ is variable-preserving in $t \not\simeq s \vee D$, one of Conditions (1a)-(1c) must hold.

- If Condition (1a) holds, then we have, say, $s \in I_\mathcal{V}(t)$. By Definition 56, this implies that s is a strict subterm of t, which is impossible since t, s are unifiable.
- If Condition (1b) holds, then $I_\mathcal{V}(t) = \emptyset$ or $I_\mathcal{V}(s) = \emptyset$, say $I_\mathcal{V}(t) = \emptyset$. By Lemma 59 $I_\mathcal{V}(t\sigma) = \emptyset$, and since $t\sigma = s\sigma$, we conclude that $I_\mathcal{V}(s\sigma) = \emptyset$. This contradicts the fact that $x \in I_\mathcal{V}(t\sigma) \cup I_\mathcal{V}(s\sigma)$.
- If Condition (1c) holds, then $y \in I_\mathcal{V}(L')$, for some $L' \in D$. But in this case, by Lemma 59, $x = y\sigma \in I_\mathcal{V}(L'\sigma) \subseteq I_\mathcal{V}(D\sigma)$.

Thus $t\sigma \not\simeq s\sigma$ is dominated in $(t \not\simeq s \vee D)\sigma$ and by Lemma 66, $D\sigma$ is variable-preserving.

C is generated by the equational factorisation rule. C is of the form $(D \vee s \not\simeq v \vee t \simeq s)\sigma$, where S' contains a clause of the form $(D \vee u \simeq v \vee t \simeq s)$, and $\sigma = mgu(u, t)$.

By definition, since $D \vee u \simeq v \vee t \simeq s$ is variable-preserving, we have $I_\mathcal{V}(t) = I_\mathcal{V}(s)$ and $I_\mathcal{V}(u) = I_\mathcal{V}(v)$, regardless of which of Conditions (2a) or (2b) holds. Thus $I_\mathcal{V}(v\sigma) = I_\mathcal{V}(u\sigma)$ by Corollary 60, and since σ is a unifier of u and t, we deduce that $I_\mathcal{V}(v\sigma) = I_\mathcal{V}(t\sigma)$ and $I_\mathcal{V}(s\sigma) \cup I_\mathcal{V}(v\sigma) \subseteq I_\mathcal{V}(s\sigma) \cup I_\mathcal{V}(t\sigma)$. Therefore, $(s \not\simeq v)\sigma$ satisfies Condition (1c), and is variable-preserving in C.

We now show that $(D \vee t \simeq s)\sigma$ is variable-preserving. By Lemma 67, $(D \vee u \simeq v \vee t \simeq s)\sigma$ is variable-preserving. But since $I_{\mathcal{V}}(u\sigma) = I_{\mathcal{V}}(v\sigma)$ and $u\sigma = t\sigma$, we conclude that $I_{\mathcal{V}}((u \simeq v)\sigma) \subseteq I_{\mathcal{V}}((s \simeq t)\sigma)$, which means that $(u \simeq v)\sigma$ is dominated in $(D \vee u \simeq v \vee t \simeq s)\sigma$, thus $(D \vee t \simeq s)\sigma$ is variable-preserving, by Lemma 66. We conclude that C is variable-preserving. ∎

5.3 Controlled Sets of Clauses

We have shown that every variable-preserving set of clauses S is stable by the inference system, thus guaranteeing that all clauses appearing in a derivation tree for S are non variable-eligible. We now investigate how to guarantee that a set of clauses is simply refutable by introducing the class of controlled clauses.

Definition 71 A clause C is *controlled* if:

- C is variable-preserving,
- the non-flat literals in C are quasi-flat and quasi-closed.

A set of clauses is *controlled* if all the clauses it contains are controlled. ◊

We provide conditions guaranteeing the stability of controlled clauses under certain conditions.

Lemma 72 *The following properties hold:*

1. *Every flat and ground clause is controlled.*
2. *If C and D are controlled, then so is $C \vee D$.*
3. *If σ is a flat substitution and C is a controlled clause, then so is $C\sigma$.*

Proof. Item 1 is obvious, and item 2 is an immediate consequence of Proposition 64. For item 3, we verify that the conditions of Definition 71 are preserved:

- The clause $C\sigma$ remains variable-preserving by Lemma 67.
- By Lemma 53, quasi-flatness and quasi-closedness are preserved by instantiation. Thus, all controlled literals remain controlled after instantiation. ∎

We also show that controlled clauses remain so after particular replacement operations, and that particular inference steps are guaranteed to generate such clauses.

Lemma 73 *Let C be a clause and let t, s be two non variable terms. Assume that:*

- *s is quasi-flat and quasi-closed;*
- *$I_{\mathcal{V}}(s) = I_{\mathcal{V}}(t)$;*
- *if $t \in \Sigma_0$, then $s \in \Sigma_0$.*

If $C[t]_p$ is controlled, then $C[s]_p$ is controlled.

Proof. By Lemma 69, C is variable-preserving, and by Lemma 54, all its non-flat literals are quasi-flat and quasi-closed. ∎

Lemma 74 *Let D_1, D_2 denote (not necessarily distinct) controlled clauses. If $\{D_1, D_2\} \rightarrow^{\mathrm{sel}}_{<,\sigma} C$, then σ is flat and C is controlled.*

PROOF. Since D_1 and D_2 are variable-preserving, by Theorem 70, C is variable-preserving. We distinguish three cases, according to the rule used to derive C.

C is generated by the superposition or paramodulation rule. This means that D_1 is of the form $L[t]_p \vee D'_1$, D_2 of the form $u \simeq v \vee D'_2$, and C is of the form $(L[v]_p \vee D'_1 \vee D'_2)\sigma$, where $\sigma = mgu(t, u)$.

By hypothesis, t and u are quasi-flat and quasi-closed, and by Lemma 55, σ must be flat. By applying Lemma 72 (3), we deduce that $(L[t]_p \vee D'_1)\sigma$ and $(u \simeq v \vee D'_2)\sigma$ are both controlled. Since D_2 is variable-preserving, one of the conditions of Definition 61 must hold. If Condition 2a holds, then v cannot be a variable. If Condition 2b holds, then v cannot be a variable either, since $u \simeq v$ is selected. Furthermore, if u is a constant, the head symbol of v cannot be a function symbol, since otherwise, we would have $v > u$. Therefore, we have $u \in \Sigma_0 \Rightarrow v \in \Sigma_0$, and by Lemma 73, $(L[v]_p \vee D'_1)\sigma$ is controlled. By Theorem 70, $C = (L[v]_p \vee D'_1 \vee D'_2)\sigma$ is variable-preserving, and it is simple to verify that C satisfies the quasi-flatness and quasi-closedness conditions.

C is generated by the reflection rule. This means that C is of the form $D'_1\sigma$, where $D_1 = (t \not\simeq s) \vee D'_1$ and $\sigma = mgu(t, s)$. If $t \simeq s$ is flat, then σ is obviously flat. Otherwise, since D_1 is controlled, t and s must be quasi-flat and quasi-closed, thus, σ must be flat by Lemma 55. By Theorem 70, $C = D'_1\sigma$ is variable-preserving, and it is controlled by Lemma 72 (3).

C is generated by the equational factorisation rule. This means that $C = (D'_1 \vee s \not\simeq v \vee t \simeq s)\sigma$, where $D_1 = (D'_1 \vee u \simeq v \vee t \simeq s)$, the selected literal is $t \simeq s$, and $\sigma = mgu(t, u)$. Since $t \simeq s$ is variable-preserving, one of Conditions 2a or 2b of Definition 61 must hold. First assume that Condition 2a holds, so that t is not a variable. If u is a variable, then it must occur in a complex term in $D'_1 \vee t \simeq s$, which would prevent $t\sigma$ from being an elligible term. If u is a constant, then t must also be a constant, and $\sigma = id$ is flat. Otherwise, t and u must both be quasi-flat and quasi-closed since D_1 is controlled, and again, σ must be flat by Lemma 55. Now assume that Condition 2b holds. Then t cannot be a variable since literal $t \simeq s$ is selected, and D_1 must be flat and ground by Condition 3 of Definition 52. Thus, necessarily, $\sigma = id$ is flat. Therefore, C is variable-preserving by Theorem 70, and controlled by Lemma 72 (3). ∎

We obtain the main result of this section:

Theorem 75 *Every set of clauses that is controlled is simply provable.*

PROOF. We prove that all derivation trees for S are simple and have a controlled root, by induction on their depth. Let $\tau = [C, \mathcal{T}, \sigma]$, and assume all the derivation trees in $\mathcal{T}$ are simple with controlled roots. Then by Lemma 74, C is controlled, and we now verify that τ satisfies the conditions of Definition 29.

1. The clauses occurring in $\mathcal{T}$ are controlled, hence by Lemma 74, σ is flat.
2. Since C is controlled, it is variable-preserving, and by Proposition 63, it is not variable-eligible. Furthermore, every controlled clause is quasi-flat by definition.
3. Since C is controlled, every term it contains is quasi-closed, which means that Condition (3) trivially holds.

4. Suppose C is generated by the superposition/paramodulation of $D_1 = u \simeq v \vee D_1'$ into a clause D_2, with the term $u\sigma$ being replaced by the term $v\sigma$. Then by hypothesis, D_1 is variable-preserving, and one of Conditions (2a) or (2b) of Definition 61 must hold. If Condition (2a) holds, then neither v nor $v\sigma$ is a variable. If Condition (2b) holds, both u and v are flat, and if they are variables, then they must appear in $I_{\mathcal{V}}(D_1')$. Thus, the only way for $u \simeq v$ to be maximal in D_1 is for u and v to both be constants, hence $v\sigma$ cannot be a variable.

5. By the induction hypothesis, every derivation tree in $\mathcal{T}$ is simple.

Therefore, τ is simple, which completes the proof. ∎

Theorem 75 makes the class of controlled clause sets a good candidate for applying the instantiation method described in Section 2. Unfortunately, several theories of interest are non controlled. For instance, the theory of arrays, with the axiom $i \simeq j \vee \mathtt{select}(\mathtt{store}(t,i,v),j) \simeq \mathtt{select}(t,j)$, is non controlled. Indeed, either $1 \in \mathrm{I}_0(\mathtt{select})$ and in this case $\mathtt{select}(\mathtt{store}(t,i,v),j)$ is non quasi-flat, or $1 \in \mathrm{I}_{\mathrm{nv}}(\mathtt{select})$ and $\mathtt{select}(t,j)$ is non quasi-closed.

In order to overcome this problem, we introduce a slightly more general class that allows us to handle such theories. The idea is that non controlled clause sets may be transformed into controlled ones using superposition from $\mathcal{C}$-equations. For instance, in the above theory, by applying the superposition rule on the term $\mathtt{store}(t,i,v)$ we shall ensure that the variable t is instantiated by a constant symbol, thus $\mathtt{select}(t,j)$ becomes quasi-closed.

6 $\mathcal{C}$-controllable Sets of Clauses

Intuitively, the more general class of $\mathcal{C}$-controllable sets of clauses is obtained by relaxing the quasi-flatness and quasi-closedness conditions on some of the terms that occur in the clauses. The terms for which the conditions can be relaxed are those that contain distinguished function symbols of Σ.

Definition 76 We denote by $\mathcal{C}$ a subset of Σ containing no constant symbol. ◊

Of course, if the clauses that contain symbols in $\mathcal{C}$ are not controlled, then some restrictions must be imposed on the inferences they are involved in. This is done by constraining form of controlled clauses containing symbols in $\mathcal{C}$.

6.1 $\mathcal{C}$-equations

We begin by introducing the class of $\mathcal{C}$-equations, along with some of the properties they satisfy.

Definition 77 A clause C is a *$\mathcal{C}$-equation* if it is of the form $f(a_1,\ldots,a_n) \simeq b \vee D$, where $a_1,\ldots,a_n,b \in \Sigma_0$, $f \in \mathcal{C}$ and D is flat and ground.

The clause C is *strongly controlled* if it is controlled and contains no occurrence of symbols in $\mathcal{C}$. ◊

We prove some closure properties on $\mathcal{C}$-equations and strongly controlled clauses. First, we show that $\mathcal{C}$-equations are closed for the relation defined in Definition 42:

Proposition 78 *Let C be a clause and E be a $\mathcal{C}$-equation. For all sets of clauses S and all flat and ground clauses D, if $C \equiv_D^S E$, then C is also a $\mathcal{C}$-equation.*

Proof. Since $C \equiv_D^S E$, by definition, C is obtained by replacing some constants in E by other constants, and the result is obvious. ∎

The results of Section 5.3 on controlled clauses can be transposed to $\mathcal{C}$-equations and stronlgy controlled clauses:

Lemma 79 *Let C and D be clauses, and σ denote a substitution:*

1. *If C is flat and ground then C is strongly controlled.*
2. *If C and D are strongly controlled, then so is $C \vee D$.*
3. *If σ is flat and C is a $\mathcal{C}$-equation (resp. a strongly controlled clause), then so is $C\sigma$.*

Proof. Item 1 results from the fact that $\mathcal{C}$ only contains function symbols, and Item 2 is an immediate consequence of Lemma 72 (2). For Item 3, if C is a $\mathcal{C}$-equation then the result is obvious, since C is ground. Otherwise, the result is a direct consequence of Lemma 72 (3), since $\mathcal{C}$ only contains function symbols, and σ is a flat substitution. ∎

Lemma 80 *Let C be a clause and let t, s be two non variable terms such that:*

- *s is quasi-flat, quasi-closed, and contains no symbol in $\mathcal{C}$;*
- *$I_{\mathcal{V}}(s) = I_{\mathcal{V}}(t)$;*
- *if $t \in \Sigma_0$, then $s \in \Sigma_0$.*

If $C[t]_p$ is either a $\mathcal{C}$-equation or a strongly controlled clause, then so is $C[s]_p$.

Proof. First assume that $C[t]_p$ is a $\mathcal{C}$-equation, and is therefore of the form $f(a_1, \ldots, a_n) \simeq b \vee C'$, where $a_1, \ldots, a_n, b \in T_0$, and C' is flat and ground. Necessarily, t is either a constant, or the term $f(a_1, \ldots, a_n)$. If t is a constant, then $s \in \Sigma_0$ by hypothesis and $C[s]_p$ is also $\mathcal{C}$-equation. Otherwise, $t = f(a_1, \ldots, a_n)$, thus $C[s]_p = s \simeq b \vee C'$, and since $s \simeq b$ is controlled (because s is quasi-flat and quasi-closed), this clause is strongly controlled.

Now assume that C is a strongly controlled clause. Then by Lemma 73, $C[s]_p$ is controlled, and since s contains no symbol in $\mathcal{C}$ by hypothesis, $C[s]_p$ is strongly controlled. ∎

Lemma 81 *Let D_1, D_2 denote (not necessarily distinct) clauses that are either $\mathcal{C}$-equations or strongly controlled clauses. If $\{D_1, D_2\} \rightarrow^{\text{sel}}_{<,\sigma} C$, then C is either a $\mathcal{C}$-equation or a strongly controlled clause. Furthermore, if neither D_1 nor D_2 is flat and ground, then C is strongly controlled.*

Proof. If C is generated by a unary inference rule, then D_1 can obviously not be a $\mathcal{C}$-equation, hence C is strongly controlled by Lemma 74. Still by Lemma 74, if neither D_1 nor D_2 is a $\mathcal{C}$-equation, then again C is strongly controlled. Now assume that C is generated by the superposition or paramodulation rule, where D_1 or D_2, possibly both, is a $\mathcal{C}$-equation. This means that D_1 is of the form $L[t]_p \vee D'_1$, D_2 of the form $u \simeq v \vee D'_2$, and C is of the form $(L[v]_p \vee D'_1 \vee D'_2)\sigma$, where $\sigma = mgu(t, u)$.

If D_2 is a $\mathcal{C}$-equation, then necessarily, u is of the form $f(a_1, \ldots, a_n)$ where $f \in \mathcal{C}$, v is a constant, and D'_2 is flat and ground. By hypothesis, D_1 is either a $\mathcal{C}$-equation or contains no symbols in $\mathcal{C}$. Thus, since u and t are unifiable, it must be a $\mathcal{C}$-equation of the form $u \simeq s \vee D'_1$, where $s \in \Sigma_0$, and D'_1 is flat and ground. Therefore, in this case, $C = s \simeq v \vee D'_1 \vee D'_2$ is strongly controlled.

If D_1 is a $\mathcal{C}$-equation and D_2 is not, then L must be the literal of the form $f(c_1 \ldots, c_n) \simeq d$, and t must be a subterm of $f(c_1, \ldots, c_n)$. We cannot have $t = f(c_1, \ldots, c_n)$ since otherwise u would have to contain a symbol in $\mathcal{C}$, thus t is a constant. Since D_2 is strongly controlled, it is also variable-preserving, and by Proposition 63, it is not variable-eligible. Thus, u is necessarily a constant, and D_2 must be flat and ground by Condition 3 of Definition 52 for the selection function. In this case, C is also a $\mathcal{C}$-equation. ∎

6.2 $\mathcal{C}$-Restricted Terms

Our first goal is to control the inferences that can be performed on a given (non controlled) clause C, in order to ensure that the following conditions hold: (i) Superposition from a $\mathcal{C}$-equation is possible, in order to transform C into a controlled clause; and (ii) No inference is possible between C and a controlled clause D that is not a $\mathcal{C}$-equation.

In order to guarantee that (i) holds, it suffices to assume that every eligible term t contains a term of the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{C}$, and $t_1, \ldots, t_n \in T_0$. For (ii), we exploit the fact that the term s in D that is unified with t is necessarily quasi-closed (since D is controlled), thus there are some positions in s along which variables cannot occur (see Definition 4). We shall assume that the term $f(t_1, \ldots, t_n)$ occurs in t at such a position. Since D is not a $\mathcal{C}$-equation, it does not contain f, thus t cannot be unified with s (since t contains an occurrence of f at a position in which no variable occur in s). This implies that no inference is possible between C and D.

This yields the following definition (recall that $\mathrm{I}_{\mathrm{nv}}(f)$ denotes the set of indices of f that must not be variables).

Definition 82 A term t is *$\mathcal{C}$-restricted* if it is of the form $f(t_1, \ldots, t_n)$ where one of the following conditions holds:

- $f \in \mathcal{C}$, and $f(t_1, \ldots, t_n)$ is of depth 1 and linear.
- There exists at least an $i \in \mathrm{I}_{\mathrm{nv}}(f)$ such that t_i is $\mathcal{C}$-restricted, and for every $j \in [1..n]$, if $j \in \mathrm{I}_{\mathrm{nv}}(f)$ then either $t_j \in \Sigma_0$ or t_j is $\mathcal{C}$-restricted. ◊

The fact that i is in $\mathrm{I}_{\mathrm{nv}}(f)$ ensures that no inference is possible with a controlled term except with a $\mathcal{C}$-equation (see Lemma 85). The second subcondition in Point 2 ensures the term will become quasi-closed after all symbols in $\mathcal{C}$ have been eliminated by superposition. The fact that $f(t_1, \ldots, t_n)$ is linear is not really restrictive and useful for technical reasons (see the proof of Theorem 102): informally it ensures that superposition into $f(t_1, \ldots, t_n)$ can be restricted to the equations already occurring in the initial clause set (the replacement of constant symbols by ground flat superposition can be postponed *after* the application of the superposition rule into $f(t_1, \ldots, t_n)$). This is useful to ensure that the derivation tree satisfies Condition 3 in Definition 29.

Example 83 Assume that $\mathcal{C} = \{f, g\}$ and that $\mathrm{I}_{\mathrm{nv}}(h) = \{1\}$. Then:

- $f(x)$, $h(f(x), a)$, $h(f(x), g(x))$ and $h(h(g(x), y), x)$ are $\mathcal{C}$-restricted.
- $h(x, a)$ and $h(h(x, y), f(x))$ are not. ♣

The following proposition provides a sufficient condition that guarantees the occurrence of a constructor in a $\mathcal{C}$-restricted term.

Proposition 84 *Let t be a quasi-flat and $\mathcal{C}$-restricted term, and let $s = f(s_1, \ldots, s_n)$ be a subterm of t. If $s_i \in \mathcal{V}$ for some $i \in \mathrm{I}_{\mathrm{nv}}(f)$, then $f \in \mathcal{C}$ and $s_1, \ldots, s_n \in T_0$.*

Proof. The proof is by induction on the size of t. If $t = s$ then since we have $i \in \mathrm{I}_{\mathrm{nv}}(f)$ and $s_i \in \mathcal{V}$, the second item in Definition 82 cannot hold. Thus we have $f \in \mathcal{C}$ and t is of depth 1 which completes the proof.

Otherwise, $t = g(t_1, \ldots, t_m)$ where s is a subterm of t_j, for some $j \in [1..m]$. Since s is a subterm of t_j, the latter cannot be in T_0, and since t is quasi-flat, $j \notin \mathrm{I}_0(g)$. Since $\mathrm{I}_0(g) \cup \mathrm{I}_{\mathrm{nv}}(g) = [1..m]$, we deduce that $j \in \mathrm{I}_{\mathrm{nv}}(g)$. By hypothesis t is $\mathcal{C}$-restricted, hence t_j must be either a constant symbol or a $\mathcal{C}$-restricted term. Since s is a subterm of t_j, the latter cannot be a constant symbol, and is therefore a $\mathcal{C}$-restricted term. Since t is quasi-flat, so is t_j, thus we can apply the induction hypothesis to obtain the result. ■

The following lemma allows to control the inferences that can be performed on restricted terms.

Lemma 85 *Let $t \notin \mathcal{V}$ be a quasi-closed term such that for every position p in t distinct from ϵ, we have $\mathrm{head}(t|_p) \notin \mathcal{C}$. Let s be a $\mathcal{C}$-restricted term of depth at least 2. Then t and s are not unifiable.*

PROOF. The proof is by induction on the size of s. By definition s must be of the form $f(s_1, \ldots, s_n)$. Since t is not a variable, we may assume that t is of the form $f(t_1, \ldots, t_n)$, otherwise t and s are obviously not unifiable. Since the depth of s is strictly greater than 1, the subterms $s_1, \ldots, s_n$ cannot all be flat. Thus by the second item of Definition 82, there exists an $i \in \mathrm{I}_{\mathrm{nv}}(f)$ such that s_i is $\mathcal{C}$-restricted; necessarily, s_i is of depth at least 1. Assume that s_i is of depth 1. In this case, it must satisfy the conditions of the first item of Definition 82, thus its head symbol must be in $\mathcal{C}$. Since $i \in \mathrm{I}_{\mathrm{nv}}(f)$ and t is quasi-closed, t_i cannot be a variable; furthermore, by hypothesis, the head symbol of t_i does not belong to $\mathcal{C}$, hence t_i and s_i are not unifiable. If s_i is of depth strictly greater than 1, then since t_i is quasi-closed, we may apply the induction hypothesis and conclude that s_i and t_i cannot be unifiable. Hence the result. ∎

6.3 $\mathcal{C}$-controllable clauses

We introduce the more general class of $\mathcal{C}$-controllable clauses. Informally, a clause set S will be *$\mathcal{C}$-controllable* if every non controlled clause in S can be transformed into a controlled clause by a sequence of superposition inferences from $\mathcal{C}$-equations. Such inference steps are always simple, thus this property, together with Theorem 75, will ensure the existence of a simple refutation tree for every unsatisfiable clause, *provided the only inferences that can be performed on non controlled clause sets are superpositions from $\mathcal{C}$-equations.* This means that one has to check that the clause set is *partially* saturated: no inference is possible between two non controlled clauses, neither between a non controlled clause and a controlled clause that is not a $\mathcal{C}$-equation. Designing (sufficiently general) decidable conditions ensuring these properties is the purpose of the following sections (this is not trivial since the considered clause sets are infinite in general).

As previously explained, one has to ensure that two non controlled clauses cannot interfere with each other (i.e. no inference is possible). We have to check that this property holds not only for the clauses occurring in S, but also for all clauses that can be derived by superposition. This is done by computing all the clauses that can be derived from S by superposition from $\mathcal{C}$-equations and from ground flat clauses (this process – although terminating – is obviously inefficient but as we shall see we do not need to compute explicitly all these clauses). Such a set will be denoted by $[S]$. This set is finite, thus it is easy to check that it is saturated.

The equations used during the superposition process will be collected and explicitly added as "constraints" to the clause. This provides additional useful information and allows us to discard some possible inferences from non controlled clauses if they are redundant with the $\mathcal{C}$-equations previously used to derive these clauses. The following example illustrates this point. Let $S = \{\mathtt{cons}(\mathtt{car}(x), \mathtt{cdr}(x)) \simeq x, \mathtt{car}(\mathtt{cons}(x, y)) \simeq x, \mathtt{cdr}(\mathtt{cons}(x, y)) = y\}$, the well-known theory of lists. The reader can check that S is non controlled. We try to show that S is $\mathcal{C}$-controllable. Using the $\mathcal{C}$-equation $\mathtt{car}(a) \simeq b$ one get $\mathtt{cons}(b, \mathtt{cdr}(a)) \simeq a$. The obtained clause is still non controlled. Unfortunately it interferes with the clause $\mathtt{car}(\mathtt{cons}(x, y)) = x$, which contradicts the previous property (no inference within non controlled clauses). Fortunately, the obtained clause is $\mathtt{car}(a) \simeq b$ which is redundant with (actually identical in this case) the clause used to derive $\mathtt{cons}(b, \mathtt{cdr}(a)) \simeq a$.

In order to formalize this idea, we introduce the notion of e-clauses.

Definition 86 An *e-clause* is a pair $[C \mid \phi]$ where C is a clause and ϕ is a set (or conjunction) of equations.$\Diamond$

The conjunction of equations ϕ in an e-clause $[C \mid \phi]$ intuitively denotes a set of literals that have been paramodulated into a clause that is not necessarily controlled, to generated the clause C. The set ϕ will be used to harness the inferences that admit C as a premice. Note that e-clauses are similar to constrained clauses (see, e.g., [18]) but the equations in ϕ do more than constrain the variables appearing in C.

The following relation computes new e-clauses by superposition and simultaneous adds the corresponding equations in the constraint part.

Definition 87 We denote by $\rightsquigarrow$ the smallest reflexive and transitive relation on e-clauses satisfying the following condition: For all e-clauses $[C \mid \phi]$ and for all terms $f(t_1, \ldots, t_n)$ occurring at a position p in C, if $f \in \mathcal{C} \cup \Sigma_0$ and $t_1, \ldots, t_n \in T_0$ then

$$[C \mid \phi] \quad \rightsquigarrow \quad [C[c]_p \mid \phi \cup \{t \simeq c\}]\theta,$$

where $c \in \Sigma_0$ and θ is a substitution mapping every variable in t to constant symbols. $\Diamond$

The relation $\rightsquigarrow$ permits to generate e-clauses containing fewer symbols in $\mathcal{C}$. This relation resembles the flattening operation described previously, but is not applied to all the function symbols in the signature; it also serves as a renaming operation for constants. Among the clauses that are generated by the relation, it will not be necessary to constrain those that are controlled. The whole set of e-clauses that can be deduced in this way (omitting controlled clauses) is denoted by $[S]$.

Definition 88 If S is a set of clauses, we denote by $[S]$ the smallest set of e-clauses such that the following conditions hold:

- If $C \in S$ and C is not controlled then $[C \mid \emptyset] \in [S]$.
- If $[C \mid \phi] \in [S]$, $[C \mid \phi] \rightsquigarrow [D \mid \psi]$ and D is not strongly controlled, then $[D \mid \psi] \in [S]$. $\Diamond$

Example 89 Let $S = \{\mathtt{cons}(\mathtt{car}(x), \mathtt{cdr}(x)) \simeq x\}$. Assume that $\mathtt{car}, \mathtt{cdr}, \mathtt{cons} \in \mathcal{C}$. The reader can check that $[S]$ consists of the following clauses:

1	$[\mathtt{cons}(\mathtt{car}(x), \mathtt{cdr}(x)) \simeq x \mid \emptyset]$	% the clause in S
2	$[\mathtt{cons}(b, \mathtt{cdr}(a)) \simeq a \mid \mathtt{car}(a) \simeq b]$	% from E-Clause 1
3	$[\mathtt{cons}(\mathtt{car}(a), b) \simeq a \mid \mathtt{cdr}(a) \simeq b]$	% from E-Clause 1
4	$[\mathtt{cons}(b, c) \simeq a \mid \mathtt{car}(a) \simeq b, \mathtt{cdr}(a) \simeq c]$	% from E-Clause 2
5	$[\mathtt{cons}(a, \mathtt{cdr}(a)) \simeq a \mid \mathtt{car}(a) \simeq b, b \simeq a]$	% from E-Clause 2
6	$[\mathtt{cons}(\mathtt{car}(a), a) \simeq a \mid \mathtt{cdr}(a) \simeq a, b \simeq a]$	% from E-Clause 3
7	$[\mathtt{cons}(b, b) \simeq a \mid \mathtt{car}(a) \simeq b, \mathtt{cdr}(a) \simeq c, c \simeq b]$	% from E-Clause 4
8	$[\mathtt{cons}(b, b) \simeq b \mid \mathtt{car}(a) \simeq b, \mathtt{cdr}(a) \simeq c, c \simeq b, a \simeq b]$	% from E-Clause 7

The remaining e-clauses are equivalent to the ones above (or subsumed), modulo a renaming of constant symbols. For instance, one could generate $[\mathtt{cons}(c, b) \simeq a \mid \mathtt{cdr}(a) \simeq b, \mathtt{car}(a) \simeq c]$ using E-Clause 3 and $\mathtt{car}(a) \simeq c$, but this e-clause is equivalent to E-Clause 4 (up to a renaming of constant symbols). Notice that the e-clause $[d \simeq a \mid \mathtt{car}(a) \simeq b, \mathtt{cdr}(a) \simeq c, \mathtt{cons}(b, c) \simeq d]$ that can be generated from E-Clause 4 (replacing $\mathtt{cons}(b, c)$ by d) does not occur in $[S]$ because it is strongly controlled. ♣

We provide a link between the relations $\equiv_C^S$ of Definition 42 and $\rightsquigarrow$.

Proposition 90 *Let $[C \mid \phi]$ be an e-clause. If $C \equiv_D^S C'$ then $[C \mid \phi] \rightsquigarrow [C' \mid \phi \vee \psi]$, where $\forall e \in \psi$, $e \vee D$ is redundant w.r.t. S.*

Proof. This follows immediately from Definitions 42 and 87. Indeed, since $C \equiv_D^S C'$, this means several terms occurring in C are replaced to yield C', and if a term t is replaced by s, then there is a clause $D' \subseteq D$ such that $t \simeq s \vee D'$ occurs in S. The equation $t \simeq s$ occurs in ψ, and obviously, $t \simeq s \vee D$ is subsumed by $t \simeq s \vee D'$. ■

Proposition 91 $[S \cup S'] = [S] \cup [S']$.

We now define the class of $\mathcal{C}$-controllable sets of clauses, which, as we shall prove, are simply provable.

Definition 92 A set of clauses S is *$\mathcal{C}$-controllable* if:

1. S is quasi-flat.
2. The controlled clauses in S that contain symbols in $\mathcal{C}$ are $\mathcal{C}$-equations.
3. If $[C \mid \phi] \in [S]$ then C is not variable-eligible.
4. If $[C \mid \phi] \in [S]$ then every eligible term in C that is not a constant is $\mathcal{C}$-restricted.
5. If $[C \mid \phi]$ and $[D \mid \psi]$ are two (variable-disjoint renamings of) e-clauses in $[S]$ and if $\{C, D\} \rightarrow^{\text{sel}}_{<,\sigma} E$, then:
   - either E is redundant w.r.t. $\phi \cup \psi \cup S$,
   - or $\sigma = id$ and E is strongly controlled. $\Diamond$

Condition 3 ensures that the clauses are non variable-eligible which is necessary to guarantee that a simple derivation tree exists. Condition 4 ensures that the only possible inferences are superpositions from $\mathcal{C}$-equations (as explained in Section 6.2). Condition 5 ensures that no (non trivial) inference is possible inside the set. We make a useful exception (second item) for inferences yielding a clause that is controlled and that is not a $\mathcal{C}$-equation, provided they are ground. This is possible because ground inferences are always simple. This exception is useful for some theories because $[S]$ may contain $\mathcal{C}$-equations generated by superposition from non controlled sets and interfering with each others. For instance for the theory of lists, on get generate equations $\mathtt{car}(a) \simeq b$ and $\mathtt{car}(a) \simeq c$. These two equations interfere, yielding $b \simeq c$. However: (i) the result is strongly controlled (ii) the inference is ground. Relaxing the condition to allow non ground inferences (e.g. flat) would be incorrect as the following example shows. Let $S = \{p(x, f(x), y), \neg p(u, u, f(v)), f(a) \simeq a\}$ where $f \in \mathcal{C}$. The reader can check that our instantiation scheme replaces the variables x, u, v by a, but not the variable y. On the other hand the only clause that can be generated from clauses in $[S]$ is $\Box$ (which is obviously strongly controlled).

The conditions in Definition 92 are obviously decidable if S and Σ_0 are finite since $[S]$ is finite in this case. However, $[S]$ can be very large if there are many constant symbols, thus it would be inefficient in practice to generate this set explicitly. Instead, $[S]$ should be computed in a symbolic way, replacing constant symbols by special variables, and the conditions should be checked directly on this abstract representation of $[S]$ rather than on $[S]$ itself, taking into account the fact that the constants can be renamed arbitrarily. This affects both the unification algorithm (constant symbols are allowed to be substituted by other constant symbols) and the selection function (one has to check that a term is eligible modulo a renaming of constant symbols).

The class of $\mathcal{C}$-controllable sets of clause contains the class of controlled sets of clauses:

Proposition 93 *A set of clauses is controlled if and only if it is $\emptyset$-controllable.*

Proof. Let S denote a set of clauses, and assume S is $\emptyset$-controllable. Then in particular, Condition 4 must hold for S. Thus, if there is an e-clause $[C \mid \phi]$ in $[S]$, then all the eligible terms in C must be $\mathcal{C}$-restricted. This is impossible since $\mathcal{C}$ is empty, thus, $[S]$ must be empty, and every clause in S must be controlled.

Conversely, if S is controlled, then $[S]$ is empty and Conditions 2, 3, 4 and 5 trivially hold. By hypothesis, S is quasi-flat, and we have the result. ∎

6.4 Completeness for $\mathcal{C}$-controllable sets of clauses

In this section, we prove that every $\mathcal{C}$-controllable set of clauses is simply provable. In order to do so, we first introduce a class of clause sets that interact in a particular way with $\mathcal{C}$-controllable sets of clauses.

The set of clauses that can be generated by the superposition calculus, starting from a $\mathcal{C}$-controllable set of clauses may be infinite and is not controlled in general. However, all these clauses satisfy some interesting property: either they are strongly controlled, or they are obtained from $\mathcal{C}$-equations in S or clauses occurring in $[S]$ by ground flat superposition. We define the clauses satisfying these conditions as *S-contained*:

Definition 94 Let S be a set of clauses. A set of clauses S' is *S-contained* if for all $C \in S'$ such that C is not strongly controlled, C is of the form $C_1 \vee C_2$ where C_2 is flat and ground, and there exists a clause D such that $D \equiv^{S'}_{C_2} C_1$ (see Definition 42) and:

1. either D is a $\mathcal{C}$-equation in S,
2. or $[S]$ contains an e-clause $[D \mid \phi]$ such that for every equation $(t \simeq s) \in \phi$, the clause $t \simeq s \vee C_2$ is redundant w.r.t. S'. ◊

If Point 1 holds, this obviously implies that C is a $\mathcal{C}$-equation. Case 2 does not cover case 1 because $[D \mid \emptyset]$ does not occur in $[S]$ if $D \in S$ and D is controlled.

The following propositions state some useful properties of S-contained clause sets.

Proposition 95 *If S_1 and S_2 are two sets of clauses that are S-contained, then so is $S_1 \cup S_2$.*

Proposition 96 *Let S, S' be two sets of clauses. Suppose that S is $\mathcal{C}$-controllable and that S' is S-contained. If C is a clause in S' that is not strongly controlled, then every eligible term t in C is either a constant or $\mathcal{C}$-restricted.*

Proof. By hypothesis, C is of the form $C_1 \vee C_2$, where C_2 is flat and ground, and there exists a clause D such that $D \equiv^{S'}_{C_2} C_1$. If D is a $\mathcal{C}$-equation then so is C, hence t must be of the form $f(c_1, \ldots, c_n)$ where $f \in \mathcal{C}$ and $c_1, \ldots, c_n \in \Sigma_0$, and the proof is obvious. Otherwise, by Point 2 of Definition 94, there exists a set of equations ϕ such that $[D \mid \phi] \in [S]$. By Proposition 90, $[D \mid \phi] \rightsquigarrow [C_1 \mid \phi \vee \psi]$ for some set of equations ψ. Since D is not strongly controlled, it must contain a symbol in $\mathcal{C}$, hence, so must C_1, which is not strongly controlled either. Thus by definition of $[S]$, the e-clause $[C_1 \mid \phi \vee \psi]$ is in $[S]$. If t occurs in C_2 then C_1 be strictly flat and strongly controlled. Otherwise, t is eligible in C_1 hence must be $\mathcal{C}$-restricted by Point 4 of Definition 92, since S is $\mathcal{C}$-controllable by hypothesis. ∎

Proposition 97 *Let S, S' be two sets of clauses. Suppose that S is quasi-flat and that S' is S-contained. Then every clause in S' is quasi-flat.*

Proof. Let $C \in S'$. If C is strongly controlled then all its non-flat literals are controlled, hence quasi-flat by Definition 71. Otherwise, C is obtained from a quasi-flat clause by adding literals that are flat and ground, by instantiating some variables by constant symbols and by replacing some subterms by constant symbols. Obviously, the obtained clause is still quasi-flat. ∎

Proposition 98 *Let S, S' be two sets of clauses. Suppose that S is $\mathcal{C}$-controllable and that S' is S-contained. Then the clauses in S' are not variable-eligible.*

PROOF. Let $C \in S'$. We consider the conditions that may be satisfied by C:

- If C is controlled, then it is variable-preserving by Definition 71, and by Proposition 63, it is not variable-eligible.
- Otherwise C is of the form $C_1 \vee C_2$, where C_2 is flat ground and $\exists [D \mid \phi] \in [S]$ such that $D \equiv^{S'}_{C_2} C_1$. By Point (3) of Definition 92, D is not variable-eligible, thus neither are C_1 and C.

The next lemma shows that the application of an inference rule between two clauses satisfying Condition 2 of Definition 94 generates a clause satisfying strong properties.

Lemma 99 *Let S, S' be two sets of clauses. Suppose that S is $\mathcal{C}$-controllable and that S' is S-contained. Let $D_1, D_2 \in S'$ be two clauses that satisfy Condition 2 of Definition 94, and are not flat and ground. If $\{D_1, D_2\} \rightarrow^{\text{sel}}_{<,\sigma} C$, then one of the following conditions holds:*

- *C is redundant in $S \cup S'$;*
- *C is strongly controlled and $\sigma = id$;*

PROOF. By Point (2) of Definition 94 for $i = 1, 2$, D_i is of the form $D'_i \vee D''_i$ where D''_i is flat and ground, and there exists an e-clause $[E_i \mid \phi_i] \in [S]$ such that $E_i \equiv^{S'}_{D''_i} D'_i$, and $\forall e \in \phi_i$, $e \vee D''_i$ is redundant in S'. By definition of $[S]$, neither E_1 nor E_2 can be strongly controlled.

The flat and ground literals in D_1, D_2 cannot be selected, otherwise the clause would be flat and ground. Therefore C must be of the form $C' \vee D''_1 \vee D''_2$, where $\{D'_1, D'_2\} \rightarrow^{\text{sel}}_{<,\sigma} C'$. By Proposition 90, $[E_i \mid \phi_i] \rightsquigarrow [D'_i \mid \phi_i \cup \psi_i]$, where $\forall e \in \psi_i, e \vee D''_i$ is redundant w.r.t. S'. Since E_1 and E_2 are not strongly controlled, neither are D'_1 and D'_2, thus for $i = 1, 2$, the e-clause $[D'_i \mid \phi_i \cup \psi_i]$ is in $[S]$.

Since S is $\mathcal{C}$-controllable, by Point (5) of Definition 92, one of the following conditions holds:

- C' is redundant in $\phi_1 \cup \phi_2 \cup \psi_1 \cup \psi_2 \cup S$. In this case, $C' \vee D''_1 \vee D''_2$ must be redundant in $\{e \vee D''_1 \vee D''_2 \mid e \in \phi_1 \cup \psi_1 \cup \phi_2 \cup \psi_2\} \cup S$. But for all $e \in \phi_1 \cup \psi_1 \cup \phi_2 \cup \psi_2$, $e \vee D''_1 \vee D''_2$ is redundant in S', hence C is redundant in $S \cup S'$ (by Proposition 8).
- C' is strongly controlled and $\sigma = id$. Then C is also strongly controlled and satisfies the second condition of the lemma. ∎

The following proposition states that S-contained clause sets are stable by ground flat superposition.

Proposition 100 *Let S and S' be sets of clauses such that S is $\mathcal{C}$-controllable and S' is S-contained. If $D_1, D_2 \in S'$, D_2 is flat and ground, and $\{D_1, D_2\} \rightarrow^{\text{sel}}_{<,\sigma} C$, then σ is flat and $S' \cup \{C\}$ is S-contained.*

PROOF. Assume that D_1 is flat. By Proposition 98 it cannot be variable-eligible, thus it must be ground by Proposition 7. Hence both D_1 and D_2 are ground and flat and so is C, thus C is strongly controlled by Lemma 79 (1). Therefore, $S' \cup \{C\}$ is S-contained.

Now assume that D_1 is not flat. Then $D_1 \neq D_2$ hence C is deduced by superposition. Note that since D_1 is not variable-eligible, the selected literal in D_1 cannot be of the form $x \simeq t$ with $x \not\in t$. This selected literal cannot be of the form $a \simeq b$ with a, b constant symbols either, since such a literal cannot be maximal in D_1 by definition of the ordering $<$.

If the superposition rule is applied from D_1 into D_2, then since D_2 is flat and ground, the selected term in D_1 must also be flat. But we have just seen that this case is impossible. Therefore, C is deduced by superposition from D_2 into D_1, and since there is no superposition into variables, C is of the form $D_1[b]_p \vee D'_2$, where $D_1|_p = a$ and $D_2 = a \simeq b \vee D'_2$. If D_1 is strongly controlled, then C is also strongly controlled. We

now assume that D_1 is of the form $D'_1 \vee D''_1$ where D''_1 is flat ground and $D'_1 \equiv^{S'}_{D''_1} E$. Necessarily, p must be a position in D'_1: indeed, D_1 is not flat, hence the literals in D''_1 cannot be selected. Thus C is of the form $(D'_1[b]_q \vee D''_1 \vee D'_2)$.

$$D'_1[b]_q \equiv^{S'}_{D'_2} D_1[a]_q \equiv^{S'}_{D''_1} E,$$

thus $D'_1[b]_q \equiv^{S'}_{D''_1 \vee D'_2} E$. Furthermore, $D''_1 \vee D'_2$ is flat and ground, and if $e \vee D''_1$ is redundant w.r.t. S' so is $e \vee D''_1 \vee D'_2$. This completes the proof. ∎

We now prove that S-contained clause sets are stable by superposition, and that all the inferences correspond to flat substitutions.

Lemma 101 *Let S and S' be sets of clauses such that S is $\mathcal{C}$-controllable and S' is S-contained. Let S'' denote a set of pairwise variable-disjoint renamings of clauses in S'. If $S'' \rightarrow^{\mathrm{sel}}_{<,\sigma} C$ and if C is not redundant in $S \cup S'$, then σ is flat and $S' \cup \{C\}$ is S-contained.*

Proof. We assume that no clause in S'' is flat and ground (otherwise the proof follows by Proposition 100). If all the clauses in S'' are controlled, then by Lemma 81, σ is flat and C is strongly controlled; hence $S' \cup \{C\}$ is S-contained. If the clauses in S'' are not controlled, then they must satisfy Condition 1 of Definition 94, hence C is strongly controlled and σ is flat by Lemma 99 (because C is not redundant by hypothesis).

We now further assume that one clause in S'' is controlled, and the other is not. Thus, S'' contains two clauses of the form $L[t]_p \vee D$ and $u \simeq v \vee D'$, and C is of the form $(L[v]_p \vee D \vee D')\sigma$, where σ is the mgu of t and u. Since no clause in S' is variable-eligible by Proposition 98, u is not a variable, and by definition of the calculus, neither is t. Furthermore, u cannot be a constant since, by definition of the ordering, that would imply $u \simeq v \vee D$ is flat and ground. Thus t and u are terms of the form $f(t_1, \ldots, t_n)$ and $f(u_1, \ldots, u_n)$ respectively. We distinguish two cases, depending of which of the clauses in S'' is controlled.

The clause $L \vee D$ is controlled, but $u \simeq v \vee D'$ is not. We assume that either L is negative, or p is not a root position (otherwise the rôles of $L \vee D$ and $u \simeq v \vee D'$ can be swaped and the proof follows from the next point). By Definition 71, this implies that t is bothe quasi-flat and quasi-closed, and that it cannot contain any symbol in $\mathcal{C}$. By Proposition 96, u must be $\mathcal{C}$-restricted. If u is of depth 1 then $f \in \mathcal{C}$, which is impossible since t contains no symbol in $\mathcal{C}$. Thus u is of depth at least 2 and by Lemma 85, t and u cannot be unifiable, and we obtain a contradiction.

$u \simeq v \vee D'$ is controlled but not $L \vee D$. Since S' is S-contained, by Definition 94, $L \vee D$ must be of the form $E_1 \vee E_2$, where E_2 is flat and ground, and there exists a clause E such that $E_1 \equiv^{S'}_{E_2} E$. Since $L \vee D$ is not a $\mathcal{C}$-equation, neither is E, thus, there exists an e-clause $[E \mid \phi] \in [S]$ such that $E_1 \equiv^{S'}_{E_2} E$, and for all $e \in \phi$, $e \vee E_2$ is redundant w.r.t. S'. Obviously, L must occur in E_1, since L contains t which is not flat. Therefore, $L \vee D$ is of the form $L \vee D_1 \vee E_2$, where $E_1 = L \vee D_1$. Consequently, since $E_1 \equiv^{S'}_{E_2} E$, the latter is of the form $L' \vee D'_1$, where $L \equiv^{S'}_{E_2} L'$, and $D_1 \equiv^{S'}_{E_2} D'_1$.

Since $u \simeq v \vee D'$ is controlled, u contains no symbol in $\mathcal{C}$, except possibly at its root position if $u \simeq v \vee D'$ is a $\mathcal{C}$-equation. By Point (4) of Definition 92, every eligible term in E that is not a constant is $\mathcal{C}$-restricted; thus, in particular, t must be $\mathcal{C}$-restricted. Since t and u are unifiable, t must be of depth 1 by Lemma 85, and therefore, $f \in \mathcal{C}$. By Definition 71, this implies that $u \simeq v \vee D'$ is a $\mathcal{C}$-equation. Thus D' is ground and flat and $v \in \Sigma_0$. Moreover, u is of the form $f(u_1, \ldots, u_n)$ where $u_1, \ldots, u_n \in \Sigma_0$. Consequently, σ must be flat, and $x\sigma \in \Sigma_0$ for every variable x occurring in t.

Since $L' \vee D'_1 \equiv^{S'}_{E_2} L \vee D_1$, by Proposition 90 $[L' \vee D'_1 \mid \phi] \rightsquigarrow [L \vee D_1 \mid \phi \cup \psi]$, where $\forall e \in \psi, e \vee E_2$ is redundant in S'. Moreover, $[L \vee D_1 \mid \phi \cup \psi] \rightsquigarrow [L[v]_p \vee D_1 \mid \phi \cup \psi \cup \{t \simeq v\}]\sigma$. By definition of $[S]$, since $[L' \vee D' \mid \phi] \in [S]$, either $[L[v]_p \vee D_1 \mid \phi \cup \psi \cup \{t \simeq v\}]\sigma$ is strongly controlled, in which case C is also strongly controlled, or it is in $[S]$. Since $t\sigma = u$, the clause $(t\sigma \simeq v) \vee D'$ is in S'. Furthermore, for every $e \in \phi \cup \psi$, $e \vee E_2$ is redundant in S' (this follows immediately from the definition of ϕ and ψ), hence $C = L[v]_p \vee D_1 \vee E_2 \vee D'$ satisfies Condition 2 in Definition 94.

We are now in a position to prove the main result of this section:

Theorem 102 *If S is a $\mathcal{C}$-controllable set of clauses, then S is simply provable.*

PROOF. Let τ be a derivation tree for S, and let S_τ denote the set of clauses occurring in τ. We prove by induction on τ that there exists a simple derivation tree τ' for S such that:

- τ and τ' have the same root,
- $S_{\tau'}$ is S-contained.

Let C denote the root of τ. If $C \in S$ then the proof is immediate: it suffices to take $\tau = \tau'$. Assume now that C is deduced by applying an inference rule between two clauses D_1, D_2 (with possibly $D_1 = D_2$) using the substitution $\sigma = mgu(t, s)$, where t, s are terms occurring respectively in D_1 and D_2. Let τ_1, τ_2 be the derivation trees of roots D_1, D_2, and let $S' = S_{\tau_1} \cup S_{\tau_2}$. We assume that C is not redundant w.r.t. $S \cup S'$ (otherwise the inference is useless).

By the induction hypothesis, we may assume that τ_1, τ_2 are simple, and that S_{τ_1} and S_{τ_2} are both S-contained. This implies that S' is S-contained by Proposition 95. By Lemma 101, σ is flat and $S' \cup \{C\}$ is S-contained; thus $S_\tau = S' \cup \{C\}$ is also S-contained. By Proposition 98, C is not variable-eligible, and by Proposition 97, C is quasi-flat. Hence Conditions 1 and 2 of Definition 29 hold for τ. Furthermore, since the subtrees of τ_1 and τ_2 are all simple, Condition 5 of Definition 29 also holds for τ.

Assume that Condition 4 does not hold, i.e., a superposition rule is applied from a clause of the form $D_2 = u \simeq v \vee D_2' \in S'$, where v is a variable. In this case, $u \simeq v \vee D_2'$ cannot be variable-preserving, hence is non controlled. But we have seen in the proof of Lemma 101 that superposition from a non controlled clause is impossible.

There remains to prove that Condition 3 of Definition 29 is satisfied. This is actually not the case for τ in general. When this is not the case, we show how to construct another derivation tree τ' with the same root as τ, and such that Condition 3 holds for τ'.

Assume that $t|_p = f(t_1, \ldots, t_n)$, where $t_i \in \mathcal{V}$, for some $i \in \mathrm{I_{nv}}(f)$; the other case is symmetrical. This implies that t is not quasi-closed, i.e., D_1 is not controlled; by Proposition 96, t is therefore $\mathcal{C}$-restricted. Furthermore, since S' is S-contained, necessarily, D_1 must satisfy Point 2 of Definition 94. If D_2 satisfies Point 2 of Definition 94, then by Lemma 99, we must have $\sigma = id$, since C is not redundant in $S \cup S'$. Thus, Condition 3 of Definition 29 holds for τ which is simple. We now assume that D_2 does not satisfy Point 2 of Definition 94; this implies that D_2 is controlled.

By Proposition 84, since t is $\mathcal{C}$-restricted and quasi-flat, $t|_p$ must be of depth 1 and f must belong to $\mathcal{C}$. Since D_2 is controlled, s cannot be variable-eligiblein D_2, and the symbol f must occur in s. Thus, since $f \in \mathcal{C}$ occurs in D_2, the latter must be a $\mathcal{C}$-equation, and be of the form $f(s_1, \ldots, s_n) \simeq c \vee E$, where $s = f(s_1, \ldots, s_n)$. Therefore, C must be of the form $(D_1[c]_q \vee E)\sigma$. Furthermore, since S' is S-contained, D_2 satisfies Point (1) of Definition 94, so that S contains a clause $D_2' = f(s_1', \ldots, s_n') \simeq c' \vee E'$ such that $D_2' \equiv_E^{S'} D_2$. In particular, for all $j \in [1..n]$, $s_j \equiv_E^{S'} s_j'$, hence $s_j \simeq s_j' \vee E$ is redundant w.r.t. S'. For all $j \in [1..n]$, if $t_j \in \Sigma_0$, then we must have $s_j = t_j$ (since t, s are unifiable, and $s_1, \ldots, s_n \in \Sigma_0$). If $t_j \neq s_j'$, then, by (unordered) superposition from the clause $s_j \simeq s_j' \vee E$, we can replace t_i by s_i'. We obtain a term that is necessarily unifiable with $f(s_1', \ldots, s_n')$, since t is linear by Definition 82. Consequently, we can apply the superposition rule between D_1 and the resulting clause, to obtain a clause that subsumes $(D_1[c']_q \vee E \vee E')\sigma$. Using the superposition rule again on constant symbols, we can transform E' into E, since $E \equiv_E^{S'} E'$, and c' into c, since $c \equiv_E^{S'} c'$. Thus, the derivation tree τ thus constructed admits C as a root. In this derivation tree, t is unified with the term $f(s_1', \ldots, s_n')$ that occurs in S. Thus Condition 3 holds for this inference. The additional inference steps are ground, thus trivially satisfy the desired conditions. This proves that the obtained derivation tree τ' is simple, and of course $S_{\tau'}$ is S-controlled (since the clauses in $S_{\tau'}$ are obtained from clauses in S_τ by replacing constant symbols by other constant symbols). ∎

Theorem 102 provides an instantiation-based decision procedure for the class of $\mathcal{C}$-controllable clause sets.

7 Combinations of theories.

We now investigate the case where the background theory of SMT problem to solve is a combination of simpler theories. The problem that needs to be solved is under what conditions it is guaranteed that the instantiation scheme can be applied to the combination of theories, provided it can be applied on the individual theories.

Theorem 103 *Given a signature Σ, let $\mathcal{C}_1, \mathcal{C}_2 \subseteq \Sigma$, such that $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$. Let S_1 and S_2 denote sets of clauses such that:*

- *S_1 is $\mathcal{C}_1$-controllable and S_2 is $\mathcal{C}_2$-controllable;*
- *the controlled clauses in S_1 (resp. S_2) that contain symbols in $\mathcal{C}_2$ (resp. $\mathcal{C}_1$) are $\mathcal{C}_2$-equations (resp. $\mathcal{C}_1$-equations);*
- *$[S_1]$ and $[S_2]$ have no function symbols in common.*

Then $S_1 \cup S_2$ is $(\mathcal{C}_1 \cup \mathcal{C}_2)$-controllable.

Proof. We prove that $S = S_1 \cup S_2$ satisfy the conditions of Definition 92 for the set $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. Since S_1 and S_2 are quasi-flat, so is S; by hypothesis, the controlled clauses in S that contain symbols in $\mathcal{C}$ are $\mathcal{C}$-equations. Thus, Conditions 1 and 2 hold. By Proposition 91, $[S] = [S_1] \cup [S_2]$, thus Conditions 3 and 4 are trivially satisfied.

Assume $[C \mid \phi] \in [S_1]$ and $[D \mid \psi] \in [S_2]$; it is simple to verify that Condition 5 holds in all the other cases. If $\{C, D\} \rightarrow^{\mathrm{sel}}_{<,\sigma} E$, then we may assume without loss of generality that E is generated by a paramodulation of $C = u \simeq v \vee C'$ into $D[t]_p$, and that σ is the mgu of u and t. Since C is not strongly controlled, u cannot be flat; and since there is no paramodulation into variables, t is also of depth at least 1. But u and t cannot have any function symbol in common by hypothesis, therefore, they are not unifiable. Hence, Condition 5 is trivially satisfied in this case. ∎

Although the second and third conditions may seem restrictive, they are trivially satisfied by theories that are axiomatized over disjoint signatures. Theorem 103 also shows that, thanks to the flattening operation, the addition of any set of ground clauses to a $\mathcal{C}$-controllable set is harmless:

Corollary 104 *If S is $\mathcal{C}$-controllable and S' only consists of Σ-equations and flat ground clauses, then $S \cup S'$ is $\mathcal{C}$-controllable.*

In particular, since any set of ground clauses can be flattened into a set of Σ-equations, if the instantiation scheme is correct for a set of clauses, then it is also correct for the latter, augmented with a set of ground clauses. This result permits us to consider applying the instantiation scheme in two ways: it can be applied *eagerly*, by instantiating the entire original problem, or it can be applied *lazily*, in a more DPLL(T)-like manner, on a conjunction of unit clauses that is a candidate model for the problem. It would be worthwhile to investigate which of the two manners is the more efficient.

8 Examples

Here are some examples of theories that are $\mathcal{C}$-controllable. This is not obvious since one must fix the set of indexes $\mathrm{I}_0(f), \mathrm{I}_{\mathrm{nv}}(f), \mathrm{Inst}(f)$ for every symbol f, to compute the set $[S]$ and to check that it satisfies the Conditions of Definition 92. The syntactic criteria have been implemented and checked automatically. Our decision procedure applies to all these theories.

Natural Numbers

$$\begin{array}{ll} (n_1) & 0 \not\simeq \mathtt{succ}(x) \\ (n_2) & x \simeq y \vee \mathtt{succ}(x) \not\simeq \mathtt{succ}(y) \\ (n_3) & 0 < \mathtt{succ}(x) \\ (n_4) & x \not< y \vee \mathtt{succ}(x) < \mathtt{succ}(y) \end{array}$$

Integer Offsets

$$\begin{array}{lll} (i_1) & p(s(x)) \simeq x & \\ (i_2) & s(p(x)) \simeq x & \\ (i_3) & s^{n-i}(x) \not\simeq p^i(x) & n > 0 \end{array}$$

The theory of Integer Offsets Modulo k can also be handled.

Ordering

$$\begin{array}{ll} (o_1) & x \not\prec y \vee y \not\prec z \vee x \prec z \\ (o_2) & x \not\prec y \vee y \not\prec x \end{array}$$

Arrays

$$\begin{array}{ll} (a_1) & \mathtt{select}(\mathtt{store}(t,i,v),i) \simeq v \\ (a_2) & i \simeq j \vee \mathtt{select}(\mathtt{store}(t,i,v),j) \simeq \mathtt{select}(t,j) \\ (a_3) & \mathtt{symmetric}(t) \not\simeq \mathtt{true} \vee \mathtt{select}(\mathtt{select}(t,i),j) \simeq \mathtt{select}(\mathtt{select}(t,j),i) \\ (a_4) & \mathtt{injective}(t) \not\simeq \mathtt{true} \vee i \simeq j \vee \mathtt{select}(t,i) \not\simeq \mathtt{select}(t,j) \end{array}$$

The theory of records can be handled in a similar way. The literals $\mathtt{symmetric}(t) \not\simeq \mathtt{true}$ and $\mathtt{injective}(t) \not\simeq \mathtt{true}$ *must* be selected in the clauses above.

Encryption

$$\begin{array}{ll} (e_1) & \mathtt{dec}(\mathtt{enc}(x,y),y) = x \\ (e_2) & \mathtt{enc}(\mathtt{dec}(x,y),y) = x \end{array}$$

These axioms encode encryption and decryption with a symmetric key. The encryption operation takes a clear-text and a key and produces a cipher-text. The decryption operation inverses the encryption by extracting a clear-text from a cipher-text using the same key.

(Possibly Empty) Lists

$$\begin{array}{ll} (l_1) & \mathtt{car}(\mathtt{cons}(x,y)) \simeq x \\ (l_2) & \mathtt{cdr}(\mathtt{cons}(x,y)) \simeq y \\ (l_3) & x \simeq \mathtt{nil} \vee \mathtt{cons}(\mathtt{car}(x),\mathtt{cdr}(x)) \simeq x \\ (l_4) & \mathtt{cons}(x,y) \not\simeq \mathtt{nil} \end{array}$$

(Possibly Empty) Doubly Linked Lists

$$
\begin{array}{ll}
(ll_1) & x = \mathtt{nil} \vee \mathtt{next}(x) = \mathtt{nil} \vee \mathtt{prev}(\mathtt{next}(x)) = x \\
(ll_2) & x = \mathtt{nil} \vee \mathtt{prev}(x) = \mathtt{nil} \vee \mathtt{next}(\mathtt{prev}(x)) = x \\
(ll_3) & \mathtt{prev}(x) \simeq \mathtt{nil} \vee \mathtt{prev}(y) \simeq \mathtt{nil} \vee x \simeq y \vee x \simeq \mathtt{nil} \vee y \simeq \mathtt{nil} \\
 & \qquad \vee \mathtt{prev}(x) \not\simeq \mathtt{prev}(y) \\
(ll_4) & \mathtt{next}(x) \simeq \mathtt{nil} \vee \mathtt{next}(y) \simeq \mathtt{nil} \vee x \simeq y \vee x \simeq \mathtt{nil} \vee y \simeq \mathtt{nil} \\
 & \qquad \mathtt{next}(x) \not\simeq \mathtt{next}(y)
\end{array}
$$

ll_3 and ll_4 are logical consequence of ll_1, ll_2, but it is necessary to include these axioms explicitly in order to satisfy the conditions of Definition 92.

Other recursive data-structures can be handled in a similar way.

Others

The union of these theories is also $\mathcal{C}$-controllable. Furthermore, if a theory $\mathcal{T}$ is $\mathcal{C}$-controllable, and if S is a set of clauses containing only $\mathcal{C}$-equations and ground flat clauses, then $\mathcal{T} \cup S$ is also $\mathcal{C}$-controllable (this follows immediately from Definition 92). This property is essential in practice, because one only has to check the controllability condition on $\mathcal{T}$.

One last interesting point is that the superposition calculus does not necessarily terminate on $\mathcal{C}$-controllable clause sets. For instance, $g(x) \simeq f(g(y))$ is obviously $\mathcal{C}$-controllable (even controlled), but the superposition calculus deduces an infinite number of clauses of the form $g(x) \simeq f^n(g(y))$. This shows evidence of the interest of the instantiation method introduced in Section 2.

Limitations

The following theory, together with the previous axioms $(a_1), (a_2)$ for the theory of arrays, is non $\mathcal{C}$-controllable.

$$(\mathtt{cst}) \quad \neg \mathtt{cst}(a, v) \vee \mathtt{select}(a, x) \simeq v$$

$\mathtt{cst}(a, v)$ expresses the fact that the array a only contains the value v. The theory cannot be controlled: since the equation $i \simeq j$ occurs in (a_2), j must occur in $I_{\mathcal{V}}(\mathtt{select}(\mathtt{store}(t, i, v), j) \simeq \mathtt{select}(t, j))$ (according to Definition 61). But then j must be in $\mathrm{Inst}(\mathtt{select})$, hence the clause $\mathtt{select}(a, x) \simeq v$ cannot be variable-preserving (since $x \in I_{\mathcal{V}}(\mathtt{select}(a, x))$ but x does not occur in the term v).

The theory is also non $\mathcal{C}$-controllable, since it would imply that $\mathtt{select} \in \mathcal{C}$ (so that (a_2) or $(\mathtt{cst})$ can be reduced into a controlled clause by superposition from a $\mathcal{C}$-equation, by Point 3 in Definition 92). But obviously, in this case new $\mathcal{C}$-equations can be derived (e.g. by superposition into the term $\mathtt{store}(t, i, v)$ in (a_1)). These $\mathcal{C}$-equations are non strongly controlled, which is explicitly forbidden by Point 3 in Definition 92.

The next example shows that the instantiation scheme does not work on this theory:

Example 105 We consider the theory $(a_1), (a_2), (\mathtt{cst})$ with the following ground equations:

$$
\begin{array}{ll}
1 & \mathtt{cst}(a, 0) \\
2 & \mathtt{cst}(b, 1) \\
3 & b \simeq \mathtt{store}(a, 0, 1) \\
4 & 0 \not\simeq 1
\end{array}
$$

The obtained clause set is unsatisfiable. Indeed, clauses 1 and 2 imply that $\mathtt{select}(a, 1) = 0$, $\mathtt{select}(b, 1) = 1$. But since $b = \mathtt{store}(a, 0, 1)$ and $0 \neq 1$ we have by (a_2): $\mathtt{select}(b, 1) = \mathtt{select}(a, 1) = 0 \neq 1$. Unfortunately, it is easy to check that the instantiation scheme cannot instantiate the variable j in (a_2) by 1 (this is obvious, since 1 does not occur on the scope of a function symbol $\mathtt{select}$).

Thus the clause set obtained by instantiation is satisfiable. ♣

References

[1] A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures, 2006. To appear in ACM Transactions on Computational Logic.

[2] A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 183(2):140–164, 2003.

[3] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.

[4] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 3(4):217–247, 1994.

[5] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, February 2009.

[6] M. P. Bonacina and M. Echenim. On variable-inactivity and polynomial T-satisfiability procedures. *J. of Logic and Computation*, 18(1):77–96, 2008.

[7] M. P. Bonacina and M. Echenim. Theory decision by decomposition. *Journal of Symbolic Computation*, pages 1–42, To appear.

[8] M. P. Bonacina, S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In U. Furbach and N. Shankar, editors, *Proc. IJCAR-3*, volume 4130 of *LNAI*, pages 513–527. Springer, 2006.

[9] A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In E. A. Emerson and K. S. Namjoshi, editors, *Proc. VMCAI-7*, volume 3855 of *LNCS*, pages 427–442. Springer, 2006.

[10] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215, July 1960.

[11] L. de Moura and N. Bjørner. Engineering DPLL(T) + saturation. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proc. IJCAR-4*, volume 5195 of *LNAI*, pages 475–490. Springer, 2008.

[12] L. M. de Moura and N. Bjørner. Efficient e-matching for smt solvers. In F. Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2007.

[13] D. L. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[14] H. Ganzinger and K. Korovin. Integrating equational reasoning into instantiation-based theorem proving. In *Computer Science Logic (CSL'04)*, volume 3210 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2004.

[15] Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Annals of Mathematics and Artificial Intelligence*, 2008. (Accepted for publication).

[16] S. Jacobs. Incremental instance generation in local reasoning. In F. Baader, S. Ghilardi, M. Hermann, U. Sattler, and V. Sofronie-Stokkermans, editors, *Notes 1st CEDAR Workshop, IJCAR 2008*, pages 47–62, 2008.

[17] J. Jouannaud and C. Kirchner. Solving equations in abstract algebras: a rule based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Essays in Honor of Alan Robinson*, pages 91–99. The MIT-Press, 1991.

[18] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990.

[19] H. Kirchner, S. Ranise, C. Ringeissen, and D.-K. Tran. Automatic combinability of rewriting-based satisfiability procedures. In M. Hermann and A. Voronkov, editors, *Proc. LPAR-13*, volume 4246 of *LNCS*, pages 542–556. Springer, 2006.

[20] S. Lee and D. A. Plaisted. Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.

[21] C. Lynch and D.-K. Tran. Automatic decidability and combinability revisited. In F. Pfenning, editor, *Proc. CADE-21*, volume 4603 of *LNAI*, pages 328–344. Springer, 2007.

[22] D. A. Plaisted and Y. Zhu. Ordered semantic hyper-linking. *J. Autom. Reasoning*, 25(3):167–217, 2000.

[23] S. Ranise and D. Deharbe. Applying light-weight theorem proving to debugging and verifying pointer programs. In *Proc. of the 4th Workshop on First-Order Theorem Proving (FTP'03)*, volume 86 of *Electronic Notes in Theoretical Computer Science*, 2003.

[24] U. Waldmann and V. Prevosto. SPASS+T. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Proc. ESCoR Workshop, FLoc 2006*, volume 192 of *CEUR Workshop Proceedings*, pages 18–33, 2006.