Fundamenta Informaticae 0–28 0
IOS Press

A Resolution calculus for first-order schemata

Vincent Aravantinos
LIG/CNRS
Vincent.Aravantinos@imag.fr

Mnacho Echenim
LIG/Grenoble INP-Ensimag
Mnacho.Echenim@imag.fr

Nicolas Peltier*
LIG/CNRS
Nicolas.Peltier@imag.fr

Abstract. We devise a resolution calculus that tests the satisfiability of infinite families of clause sets. For schemata of propositional clause sets, we prove that this calculus is sound, refutationally complete, and terminating. The calculus is extended to first-order clauses, for which termination is lost, since the satisfiability problem is not semi-decidable for non-propositional clauses. The expressive power of the considered logic is strictly greater than the one considered in [6].

Keywords: Propositional logic, first-order logic, schemata, resolution calculus

1. Introduction

Many problems in mathematics or in formal verification can be specified as schemata of formulæ, corresponding to infinite sequences of structurally similar formulæ in some base language (e.g. propositional

Address for correspondence: LIG, 220, rue de la Chimie 38400 Saint Martin d'Hères, France
*This work has been partly funded by the project ASAP of the French *Agence Nationale de la Recherche* (ANR-09-BLAN-0407-01).

logic). Such schemata are parameterized by a natural number n and are usually defined by induction on n; in fact, the signature itself may depend on n. A typical example is the family of formulæ:

$$p_0 \wedge (\bigwedge_{i=1}^{n} p_i \Rightarrow p_{i+1}) \wedge \neg p_{n+1},$$

and this schema is obviously unsatisfiable for all values of $n \in \mathbb{N}$. Another example is:

$$(p \Leftrightarrow (\neg p \Leftrightarrow (\cdots (\neg p \Leftrightarrow (p \Leftrightarrow (\cdots)) \cdots))))$$

which formally corresponds to the following formula ϕ_n defined by induction:

$$\phi_0 \rightarrow \texttt{true}$$

$$\phi_{n+1} \rightarrow \neg p \Leftrightarrow (p \Leftrightarrow \phi_n)$$

It is straightforward to verify that ϕ_n is equivalent to `true` if n is even and to `false` if n is odd.

Examples of applications include the formal verification of inductively-defined hardware [13], since parameterized circuits can be easily described by schemata of propositional formulæ. Another application is the automated verification of code fragments containing loops (the parameter then encodes the number of iterations). Similarly, some inductive proofs in mathematics may be modeled as schemata of proofs in first-order logic, the parameter being in this case the natural number on which the induction is based (see [8] for an example of application of this technique). While existing SAT-solvers can be used to reason on any particular *instance* of these formulæ, proving that a property holds *for every value of the parameter* obviously requires much more sophisticated reasoning techniques, using some form of mathematical induction and invariant generation.

Proof procedures already exist to test the validity of *propositional* schemata [6]. The satisfiability problem, which consists in deciding whether there is a value of the parameter n for which the schema is satisfiable, is actually undecidable in general (it is obviously semi-decidable), but decision procedures can be defined for various subclasses, namely for *regular* [3], *regularly nested* [4] or *bound-linear* [6] schemata. The present paper describes a new decision procedure, that is based on the *resolution calculus* [17]. The proposed calculus is sound, refutationally complete and terminating. Its main advantage is that, unlike the procedure proposed in [3] and [4], its extension to first-order clauses is straightforward (see Section 8) although in this case termination and even refutational completeness can no longer be ensured. If restricted to propositional logic, the considered class of schemata is similar, but more general, than the class of regular schemata introduced in [3]. It is not comparable to the other known decidable classes (note that regularly nested and bound-linear schemata may be transformed into regular ones at the cost of an exponential blow-up, see [6] for details).

Families of (propositional) formulæ may be expressed using a first-order syntax. For instance the first schema above can be denoted as follows:

$$\exists n\, p_0 \wedge \forall i\, (i \geq 1 \wedge i \leq n \wedge p_i \Rightarrow p_{\mathtt{s}(i)}) \wedge \neg p_{\mathtt{s}(n)},$$

where the symbol `s` denotes the successor function. However, it should be noted that the formula is unsatisfiable *only if* n *is interpreted on the set of natural numbers, and* `s` *as the standard successor function*. The formula is clearly satisfiable on other domains (for instance one can take $\mathtt{s}(0) = 0$,

$\mathtt{s}(1) = 1$, $p(0) = \mathtt{true}$, $p(1) = \mathtt{false}$ and $n = 1$). The usual resolution calculus therefore fails to prove the unsatisfiability of the previous schema. Testing the satisfiability of a formula in a fixed domain (such as $\mathbb{N}$) is a very complex problem from a theoretical point of view. Due to well-known theoretical limitations, no refutationally complete proof procedure can possibly exist: the problem is *not* semi-decidable. The integration of linear arithmetic into the superposition calculus [9] is considered in [16, 1] ([10] describes a general approach for hierarchic reasoning). In [15], a superposition-based calculus is proposed for reasoning on formulæ in which the domain of interpretation of some constants (or existential variables) is fixed. These calculi are in some sense the starting point of our work. However, they are not complete in general and they do not terminate on the class of problems that we consider (completeness results exist for such calculi but they do not apply to our case).

We first show that propositional schemata can be encoded as sets of clauses in which some constant symbols (the parameters) denote natural numbers and must be interpreted as elements of $\mathbb{N}$. The inductive rules defining the indexed formulæ are also encoded as clauses. Then we define a resolution calculus operating on such sets of clauses. In this context, an application of the resolution rule can be interpreted either as a resolution inference in the usual sense or as an unfolding of an inductive definition. The resolution rule does not terminate in general, even if the clause set is unsatisfiable (i.e. the calculus is *not* complete, although a very weak form of completeness can be stated). However we define a loop detection rule that is able to ensure both completeness and termination by automatically bounding the value of the parameter. This is done by generating so-called *pruning clauses* that are of the form $n < k$, where n is the parameter and k is a natural number. Once such a pruning clause is generated, the problem becomes essentially equivalent to a propositional one. In general, the pruning clauses are *not* logical consequence of the axioms, but they can be safely added into the clause sets while retaining satisfiability.

The language we consider is of course weaker than those in [16, 1, 15] but on the other hand this approach ensures termination and refutational completeness for schemata of propositional clause sets. Although loop detection techniques and termination results already exist for the calculus in [15] (see also [14]), strong restrictions are imposed on the syntax of the formulæ (e.g. they must be Horn), which are *not* in general fulfilled by the formulæ we consider in the present paper, even in the propositional case. Thus the techniques previously used to ensure termination are too weak for our purpose.

The rest of the paper is structured as follows. Section 2 defines the syntax and semantics of the considered logic. In Section 3 we define the resolution calculus and establish a weak form of refutational completeness, then some additional syntactic restrictions are imposed on the clause sets in Section 4. In Section 5, we provide some hints on the expressive power of the obtained language and show how to encode schemata of propositional formulæ. Sections 6 and 7 are the core of our work: in Section 6 a loop detection technique is presented and, in Section 7, the termination and refutational completeness of the resulting calculus is proved. Section 8 extends the calculus to first-order clauses (this is done as usual by lifting), although in this case refutational completeness does not hold (the satisfiability problem is not semi-decidable for schemata of first-order clause sets). Section 9 briefly concludes our work and identifies some lines of future research.

2. Preliminaries

For clarity and simplicity, we first restrict ourselves to schemata of *propositional* clause sets. The calculus will later be extended to first-order clauses (see Section 8). Informally, the syntax and semantics of our

logic are almost identical to a very simple subclass of clausal logic, except that we consider a particular set of constant symbols, called *parameters*, that must be interpreted as natural numbers. Every predicate symbol is monadic and the only function symbols on the natural numbers are $0, \mathtt{s}$ ($\mathtt{s}$ stands for the successor function).

Let Ω be a set of unary predicate symbols, or (indexed) *propositional variables*, and let $\mathcal{P}$ and $\mathcal{V}$ be two disjoint sets of *variables*. The elements in $\mathcal{P}$ are the *parameters*, and those in $\mathcal{V}$ are the *index variables*. Throughout this paper, parameters are denoted by n, m and index variables by x, y, z. The letters i, j, k, l will represent natural numbers (i.e., they are meta-variables).

The set of *(arithmetic) terms* $\mathcal{T}$ is the least set satisfying $\mathcal{V} \subseteq \mathcal{T}$, $0 \in \mathcal{T}$ and $t \in \mathcal{T} \Rightarrow \mathtt{s}(t) \in \mathcal{T}$. Note that $\mathcal{T}$ does *not* contain any occurrence of a parameter. A term is *ground* iff it is of the form $\mathtt{s}^i(0)$ (with $i \in \mathbb{N}$), in which case it may be viewed as a natural number and simply denoted by i. The set of ground terms is denoted by $\mathbb{N}$.

An *atom* is either of the form $n \approx t$ where $n \in \mathcal{P}$ and $t \in \mathcal{T}$, or of the form p_t where $p \in \Omega$ and $t \in \mathcal{T}$; the atom is *ground* if t is ground. Atoms of the form $n \approx t$ are called *equational atoms* and those of the form p_t are called *indexed atoms*. Note that parameters only occur in equational atoms. A *literal* is either an atom (positive literal) or the negation of an atom (negative literal). A *clause* is a finite multiset (written as a disjunction) of literals and the empty clause is denoted by $\Box$. An *equational literal* (resp. *indexed literal*) is a literal whose atom is an equational atom (resp. an indexed atom). We denote by C^{eq} and C^{idx} the multisets (disjunctions) of equational literals (resp. indexed literals) in C. If S is a set of clauses then S^{idx} denotes the set of clauses $\{C^{idx} \mid C \in S\}$. A clause C is *purely equational* iff $C^{idx} = \emptyset$ and *parameter-free* iff $C^{eq} = \emptyset$.

For every expression (term, atom, literal or clause) $\mathcal{E}$, $var(\mathcal{E})$ denotes the set of index variables occurring in $\mathcal{E}$. The *depth* of an expression is defined as usual:

- $\mathrm{depth}(x) \stackrel{\text{def}}{=} 0$ if $x \in \mathcal{V}$, $\mathrm{depth}(0) \stackrel{\text{def}}{=} 0$, $\mathrm{depth}(\mathtt{s}(t)) \stackrel{\text{def}}{=} 1 + \mathrm{depth}(t)$,
- $\mathrm{depth}(\neg p_t) \stackrel{\text{def}}{=} \mathrm{depth}(p_t) \stackrel{\text{def}}{=} \mathrm{depth}(t)$, $\mathrm{depth}(n \not\approx t) \stackrel{\text{def}}{=} \mathrm{depth}(n \approx t) \stackrel{\text{def}}{=} \mathrm{depth}(t)$,
- $\mathrm{depth}(\bigvee_{i=1}^{n} l_i) \stackrel{\text{def}}{=} \max_{i \in [1,n]} \mathrm{depth}(l_i)$ (by convention $\mathrm{depth}(\Box) \stackrel{\text{def}}{=} 0$).

A *substitution* σ is a function mapping every index variable x to a term $x\sigma \in \mathcal{T}$ (in particular, parameters cannot be mapped). The *domain* $dom(\sigma)$ of σ is the set of index variables x such that $x\sigma \neq x$. For every expression $\mathcal{E}$, $\mathcal{E}\sigma$ denotes the expression obtained from $\mathcal{E}$ by replacing every variable x by $x\sigma$. A substitution σ is *ground* iff for every $x \in dom(\sigma)$, $x\sigma$ is ground. A *renaming* is an injective substitution σ such that $x\sigma \in \mathcal{V}$ for every $x \in dom(\sigma)$. A substitution σ is *flat* if for every $x \in \mathcal{V}$, $x\sigma \in \mathcal{V} \cup \{0\}$. Substitution σ is a *unifier* of $t_1, \ldots, t_n$ iff $t_1\sigma = \ldots = t_n\sigma$. As is well known, any unifiable set of terms has a most general unifier (unique up to a renaming), denoted by $\mathrm{mgu}(t_1, \ldots, t_n)$. Note that in our simple case, all terms are of the form $\mathtt{s}^i(t)$ where $t \in \{0\} \cup \mathcal{V}$. Thus any unifier of $\mathtt{s}^i(t)$ and $\mathtt{s}^j(s)$ where $t, s \in \{0\} \cup \mathcal{V}$, if it exists, is either empty or of one of the forms $t \mapsto \mathtt{s}^{j-i}(s)$ (if $t \in \mathcal{V}$ and $j \geq i$) or $s \mapsto \mathtt{s}^{i-j}(t)$ (if $s \in \mathcal{V}$, $t \neq s$ and $j < i$).

An *interpretation* I is a function mapping:

- Every parameter to a ground term in $\mathcal{T}$ (i.e. a natural number).
- Every ground indexed atom to a truth value `true` or `false`.

An interpretation I *validates*:

- A ground atom $n \approx t$ iff $I(n) = t$.
- A ground atom p_t iff $I(p_t) = \texttt{true}$.
- A ground literal $\neg a$ iff I does not validate a.
- A ground clause C iff it validates at least one literal in C.
- A non ground clause C iff for every substitution σ of domain $var(C)$, I validates $C\sigma$.
- A set of clauses S iff it validates every clause in S.

We write $I \models S$ iff I validates S (in which case I is called a *model* of S). If S, S' are two sets of clauses, we write $S \models S'$ iff for every interpretation I, we have $I \models S \Rightarrow I \models S'$. By definition, an interpretation I validates a clause $n \not\approx \mathtt{s}^i(x)$ iff $I(n) < i$. Thus the notation $n < i$ can be used as a shorthand for the clause $n \not\approx \mathtt{s}^i(x)$.

Proposition 2.1. The satisfiability problem is decidable for (finite) purely equational clause sets.

Proof:
By definition of the semantics, a purely equational clause set $\{C_1, \ldots, C_k\}$ is equivalent to the formula $\bigwedge_{i=1}^{k} \forall \vec{x_i} C_i$ where $\vec{x_i}$ is the set of index variables in C_i. The only symbols occurring in this formula (beside the variables in $\vec{x_i}$) are 0, $\mathtt{s}$, $\approx$, $\not\approx$ and the parameters. Since 0, $\mathtt{s}$ are uninterpreted, $\{C_1, \ldots, C_k\}$ is satisfiable iff the formula $\exists \vec{n}. \bigwedge_{i=1}^{k} \forall \vec{x_i} C_i$ holds in the empty theory, where $\vec{n}$ is the set of parameters in $C_1, \ldots, C_n$. The satisfiability problem is well known to be decidable for such formulæ (see, e.g., [11]). □

Remark 2.1. Note that by definition, any clause of the form $n \not\approx u \vee n \not\approx v \vee C$, where u and v are not unifiable, is valid. For instance, $n \not\approx \mathtt{s}(x) \vee n \not\approx x$ holds in any interpretation, since n cannot be equal to both $\mathtt{s}(t)$ and t for any $t \in \mathcal{T}$.

3. The calculus

3.1. Definition

Given two terms u and v, we write $u \lhd v$ iff $v = \mathtt{s}^i(u)$ for some $i > 0$. This ordering is extended into a pre-ordering on atoms as follows: $p_u \lhd q_v$ iff $u \lhd v$.

We assume that a total ordering $\prec$ is given on the elements of Ω. The ordering $\prec$ is extended into an ordering on atoms as follows: $a \prec a'$ iff either $a \lhd a'$ or there exists a term u such that $a = p_u$, $a' = q_u$ and $p \prec q$. By definition, the equational atoms are not comparable. Note that $\prec$ is stable by substitution, i.e. $a \prec a' \Rightarrow a\sigma \prec a'\sigma$, for every substitution σ. The orderings $\lhd$ and $\prec$ are extended to index literals by ignoring negation symbols and to clauses by the multiset extension.

Let sel be a *selection function* mapping every clause C to a *possibly empty* set of *selected index literals* in C. We assume that sel satisfies the following conditions: for every clause C such that $C^{idx} \neq \emptyset$,

either sel(C) is a nonempty set of $\lhd$-maximal negative literals or sel(C) is the set of $\prec$-maximal literals in C^{idx}. In particular, by definition, equational literals are never selected, thus sel$(C) = \emptyset$ if $C^{idx} = \emptyset$.

For instance, consider the clause

$$n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee \neg p_x \vee \neg p_{\mathsf{s}(x)} \vee q_{\mathsf{s}(x)} \vee r_{\mathsf{s}(x)},$$

and assume that $p \prec q \prec r$. Then $p_{\mathsf{s}(x)}$, $q_{\mathsf{s}(x)}$ and $r_{\mathsf{s}(x)}$ are $\lhd$-maximal, and $r_{\mathsf{s}(x)}$ is $\prec$-maximal. The selection function sel can contain the literals $\neg p_{\mathsf{s}(x)}$ or $r_{\mathsf{s}(x)}$ but neither $\neg p_x$ nor $q_{\mathsf{s}(x)}$.

The Resolution calculus is defined, as usual, by the rules depicted in Figure 1 (we externalize factorization for technical convenience).

Resolution	$\dfrac{p_u \vee C \qquad \neg p_v \vee D}{(C \vee D)\sigma}$	(i)
Factorization	$\dfrac{p_u \vee p_v \vee C}{(p_u \vee C)\sigma}$ $\qquad$ $\dfrac{\neg p_u \vee \neg p_v \vee C}{(\neg p_u \vee C)\sigma}$	(ii)

where the following conditions hold:

***(i)*:** $\sigma = \text{mgu}(u, v)$, and $p_u\sigma$ and $\neg p_v\sigma$ are selected;

***(ii)*:** $\sigma = \text{mgu}(u, v)$, and $p_u\sigma$ or $\neg p_u\sigma$ is selected.

Figure 1. The Resolution calculus

We denote by $\mathcal{R}es(S)$ the set of clauses that can be deduced from (pairwise index variable disjoint renamings of) clauses in S by resolution or factorization in one step. A *derivation from* a clause set S is a sequence of clauses $C_1, \ldots, C_n$ such that for every $i \in [1, n]$, $C_i \in S \cup \mathcal{R}es(\{C_1, \ldots, C_{i-1}\})$. We write $S \vdash C$ iff there exists a derivation $C_1, \ldots, C_n$ from S such that $C = C_n$. A derivation *of* S is a derivation containing every clause in S. For instance $\neg p_{\mathsf{s}(0)}, \neg p_x \vee p_{\mathsf{s}(x)}, \neg p_0$ is a derivation of $\{\neg p_0\}$ from $\{\neg p_{\mathsf{s}(0)}, \neg p_x \vee p_{\mathsf{s}(x)}\}$.

A clause C is *redundant* w.r.t. a clause set S (written $C \sqsubseteq S$) iff for every ground substitution σ of domain *var*(C), there exist $D_1, \ldots, D_n \in S$ and $\sigma_1, \ldots, \sigma_n$ such that $D_1\sigma_1, \ldots, D_n\sigma_n \models C\sigma$ and $D_1\sigma_1, \ldots, D_n\sigma_n \preceq C\sigma$. If S is a clause set, we write $S \sqsubseteq S'$ iff every clause in S is redundant w.r.t. S'. A clause set S is *saturated* iff $\mathcal{R}es(S) \sqsubseteq S$.

Note that the notion of refutation differs from the usual one: since no rule can be applied on the equational part of the clauses, these literals cannot be eliminated:

Definition 3.1. A *refutation* of S is a derivation from S of a finite unsatisfiable set of purely equational clauses.

The satisfiability of this set can be tested by Proposition 2.1.

3.2. Basic properties of the calculus

It is straightforward to check that the above rules are sound, i.e., that the conclusions are logical consequences of the premises.

Proposition 3.1. Let S be a set of clauses. Then $S \models \mathcal{R}es(S)$.

We prove a weak form of refutational completeness. A first step towards this proof is to notice that the calculus is complete for parameter-free clause sets:

Proposition 3.2. Let S be a saturated set of parameter-free clauses. If S is unsatisfiable then $\Box \in S$.

Proof:
This result is obvious: when the clauses under consideration contain no equational literals, this calculus coincides with the ordinary Resolution calculus. □

Theorem 3.1. Let S be a saturated set of clauses. If S is unsatisfiable then there exists a set of purely equational clauses $S' \subseteq S$ that is unsatisfiable.

Proof:
Let I be an interpretation of the parameters in S, let S_g be the set of ground instances of the clauses in S and consider the set of ground clauses S_I obtained from S_g by:

- deleting every clause containing an equational literal that is true in I,
- removing all equational literals in the remaining clauses.

By construction, S_I contains no equational literal, hence no parameter, and if S_I admits a model J, then obviously $I \cup J$ is a model of S. Thus, by hypothesis, S_I must be unsatisfiable. We verify that S_I is saturated.

Since S is saturated, and since the notion of saturatedness only depends on ground instances, it is clear that S_g is also saturated. Let $p_u \vee C$ and $\neg p_u \vee D$ be two clauses in S_I on which the Resolution rule applies. By definition, S_g contains two clauses $p_u \vee C \vee C'$ and $\neg p_u \vee D \vee D'$, where C', D' are purely equational clauses that are false in I. Since S_g is saturated, there exist n clauses $E_1, \ldots, E_n \in S_g$ such that $E_1, \ldots, E_n \models C \vee D \vee C' \vee D'$ and $E_1, \ldots, E_n \preceq C \vee D \vee C' \vee D'$. Assume w.l.o.g. that there exists a $k \in [0, n]$ such that for every $i \in [1, k]$, $I \models E_i^{eq}$ and for every $i \in [k+1, n]$, $I \not\models E_i^{eq}$ (in other words, the k clauses such that $I \models E_i^{eq}$ are assumed to be at the beginning of the sequence $E_1, \ldots, E_n$). By construction, $E_{k+1}^{idx}, \ldots, E_n^{idx} \in S_I$, and we show that $E_{k+1}^{idx}, \ldots, E_n^{idx} \models C \vee D$.

Let J be an interpretation satisfying $E_{k+1}^{idx}, \ldots, E_n^{idx}$, and consider the interpretation J' defined as follows: $J'(n) \stackrel{\text{def}}{=} I(n)$ and $J'(p_t) \stackrel{\text{def}}{=} J(p_t)$ for every ground atom p_t. By construction, $J' \models E_i^{eq}$, for every $i \in [1, k]$, and $J' \models E_i^{idx}$, for every $i \in [k+1, n]$. Therefore $J' \models E_1, \ldots, E_n$, and $J' \models C \vee D \vee C' \vee D'$. But by hypothesis $I \not\models C' \vee D'$, hence, $J \models C \vee D$ and $E_{k+1}^{idx}, \ldots, E_n^{idx} \models C \vee D$. Since $E_1, \ldots, E_n \preceq C \vee D \vee C' \vee D'$ and since index literals and equational literals are not comparable, necessarily, $E_{k+1}^{idx}, \ldots, E_n^{idx} \preceq C \vee D$. Consequently, $C \vee D$ is redundant in S_I. The proof that any factor of a clause in S_I is redundant is similar.

This proves that $\mathcal{R}es(S_I) \sqsubseteq S_I$ and that S_I is saturated. By Proposition 3.2, we deduce that $\Box \in S_I$, which means that S contains a purely equational clause C_I that is false in I. The existence of such a clause is therefore guaranteed for every interpretation I, hence S contains a set S' of purely equational clauses such that for every interpretation I of the parameters in S', $I \not\models S'$. This implies that S' is unsatisfiable. □

Theorem 3.1 does *not* imply semi-decidability because S' may well be infinite. For example, consider the set $S = \{n \not\approx x \vee p_x, p_y \vee \neg p_{\mathsf{s}(y)}, \neg p_0\}$, which formalizes the following statements:

- p_n holds,
- if p_{i+1} holds, then so does p_i,
- p_0 does not hold.

It is clear that S is unsatisfiable. Yet, this is not detected in finite time by the calculus, which generates an infinite set of purely equational clauses of the form $n \not\approx \mathsf{s}^i(0)$:

1	$n \not\approx x \vee p_x$	(given)
2	$p_y \vee \neg p_{\mathsf{s}(y)}$	(given)
3	$\neg p_0$	(given)
4	$n \not\approx \mathsf{s}(y) \vee p_y$	(resolution $1, 2$, $x \mapsto \mathsf{s}(y)$)
$4'$	$n \not\approx \mathsf{s}(y') \vee p_{y'}$	(renaming, 4)
5	$n \not\approx 0$	(resolution $1, 3$, $x \mapsto 0$)
6	$n \not\approx \mathsf{s}(0)$	(resolution $4, 3$, $y \mapsto 0$)
7	$n \not\approx \mathsf{s}(\mathsf{s}(y)) \vee p_y$	(resolution $4', 2$, $y' \mapsto \mathsf{s}(y)$)
8	$n \not\approx \mathsf{s}(\mathsf{s}(0))$	(resolution $7, 3$, $y \mapsto 0$)
	$\ldots$	

This set of clauses is clearly unsatisfiable, but every finite subset it contains is satisfiable.

A natural way of verifying that this set is unsatisfiable would be to realize that clauses 1 and 3 generate clause 4, which states that p_{n-1} holds, and this clause is identical to clause 1, up to a *shift* of parameter n. By iteration, it is possible to generate any clause of the form $n \not\approx \mathsf{s}^k(x) \vee p_x$ for $k \in \mathbb{N}$, stating that p_{n-k} holds. Then the infinite descent principle implies that p_0 must also hold, a contradiction. It turns out that the detection of such a loop between clauses 1 and 4 implies that it is not necessary to consider an arbitrary value for n, and that it can safely be assumed that $n < 3$. Indeed, it can be shown that any interpretation mapping n to an integer greater than 3 can be transformed into another one mapping n to a strictly smaller integer, such that both interpretations agree on the truth value of S. The translation into clausal form of this constraints yields the purely equational clause $n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(x)))$, and the calculus can be used to prove that $S \cup \{n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(x)))\}$ is unsatisfiable in finite time.

In what follows, we exploit this observation to introduce a rule that will permit to prune infinite derivations and thus regain refutational completeness for the calculus. After restricting the form of the clauses under consideration, which in particular can only admit a single parameter n, we define a loop-detection rule that is based on so-called *levels* of sets of clauses. We show that once such a loop has been detected, a *pruning clause* representing an upper-bound constraint on parameter n can safely be added to the considered set of clauses, and termination will be ensured, provided a fair strategy is applied for the calculus.

4. A restriction of the language

In order to conveniently exploit the properties of the calculus, we restrict ourselves to a particular class of clause sets. The clauses in this class have a simple structure that will be used to prove the results leading to the recovery of refutational completeness, and, as shown in Section 5, no loss of expressiveness is entailed for our purpose. This particular class is based on so-called t-*clauses*:

Definition 4.1. Let $t \in \{0\} \cup \mathcal{V}$. A clause C is a t-*clause* iff every indexed atom occurring in it is of the form $p_{\mathtt{s}^i(t)}$ for some $i \in \mathbb{N}$.

Proposition 4.1.

1. If C is a t-clause, with $t \in \mathcal{V} \cup \{0\}$, and σ is flat then $C\sigma$ is a $t\sigma$-clause.
2. If C, D are two t-clauses then $C \vee D$ is a t-clause.

Informally, the restrictions we impose to the sets of clauses under consideration are the following:

- The clauses contain at most one index variable and clauses containing a variable cannot contain any ground term (i.e. no occurrence of constant symbol 0). For instance, the clauses $p_x \vee p_y$ and $p_x \vee q_0$ do not satisfy these restrictions.
- The non-equational part of any clause is of a limited depth
- The equational part of the clauses must be negative.

This yields the following formal definition:

Definition 4.2. A clause C is *normalized* iff there exists a term $t \in \mathcal{V} \cup \{0\}$ such that the following conditions hold:

- C^{eq} is either empty or of the form $n \not\approx \mathtt{s}^k(t)$.
- C^{idx} is a t-clause of depth 0 or 1.
- If $t = 0$ then $\mathrm{depth}(C^{idx}) = 0$.

A set of clauses S is *normalized* iff the following conditions hold:

1. S contains at most one parameter (always denoted by n in what follows).
2. Every clause in S is normalized.

Example 4.1. The clauses $p_x \vee \neg p_{\mathtt{s}(x)}$, $n \not\approx \mathtt{s}(\mathtt{s}(x)) \vee p_x$ and $n \not\approx 0 \vee q_0$ are normalized, but $n \not\approx 0 \vee p_{\mathtt{s}(0)}$ and $n \not\approx \mathtt{s}(x) \vee \neg p_0 \vee q_x$ or $p_{\mathtt{s}(\mathtt{s}(x))}$ are not.

The following proposition is a direct consequence of the definitions of the ordering $\prec$ and of normalized clauses.

Proposition 4.2. Consider a normalized clause C that is not purely equational, and let l be a $\prec$-maximal index literal in C. Then:

- l is the unique $\prec$-maximal literal in C;
- if the index of l is a variable, then C^{idx} is of depth 0.

Proposition 4.3. If the Factorization rule can be applied to a normalized clause, then this application is trivial, i.e., it consists in removing a duplicate literal.

Proof:
Assume the Factorization rule can be applied to a normalized clause $p_u \vee p_v \vee C'$ (the proof is identical if it is applied to a clause $\neg p_u \vee \neg p_v \vee C'$). By definition, u and v are either both ground or both of the form $\mathsf{s}^i(x)$ and $\mathsf{s}^j(x)$ for some variable x. By hypothesis these two terms are unifiable, thus necessarily $u = v$ and the mgu of u and v is empty. Therefore, the rule simply removes a duplicate literal. □

We introduce the notion of *level* which is a measure on clauses. As we shall see, the level of normalized clauses cannot decrease with inferences.

Definition 4.3. The *level* of a normalized clause is an element of $\mathbb{N} \cup \{\bot\}$, defined as follows:

- If $C^{eq} = \Box$ then $\mathit{level}(C) \stackrel{\text{def}}{=} \bot$.
- If $C^{eq} \neq \Box$ then $\mathit{level}(C) \stackrel{\text{def}}{=} \mathrm{depth}(C^{eq}) - \mathrm{depth}(C^{idx}) + 1$.

In other words, a normalized clause of level $\bot$ is parameter-free, and a normalized clause of level $j \in \mathbb{N}$ is of the form $n \not\approx \mathsf{s}^{j+\epsilon-1}(t) \vee C$, where C is a t-clause of depth ϵ.

If $i \in \mathbb{N} \cup \{\bot\}$, then $S|_i$ denotes the set of clauses in S of level i. If I is a subset of $\mathbb{N} \cup \{\bot\}$, then $S|_I$ denotes the set of clauses whose levels are in I. We order the levels in $\mathbb{N} \cup \{\bot\}$ by using the usual ordering on natural numbers and by assuming that $\bot < i$ for every $i \in \mathbb{N}$. Thus if i is a natural number then $[\bot, i]$ denotes the set $\{\bot\} \cup [0, i]$.

Definition 4.4. A set of clauses is *k-normalized* if it is normalized and the level of every clause in S is in $[\bot, k]$.

Example 4.2. The levels of $n \not\approx \mathsf{s}(0) \vee p_0$, $n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee p_{\mathsf{s}(x)}$ and $n \not\approx \mathsf{s}(x) \vee p_x$ are all equal to 2. For any parameter-free and normalized clause C, the level of $n \not\approx 0 \vee C$ is 1, that of $n \not\approx \mathsf{s}(0) \vee C$ is 2, etc., since because C necessarily has depth 0 in this case. The level of $n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee p_x$ is 3 (and so is the level of its instance $n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(y))) \vee p_{\mathsf{s}(y)}$), the level of $n \neq \mathsf{s}(\mathsf{s}(x)) \vee p_{\mathsf{s}(x)}$ is 2.

Intuitively, a clause of level $\bot$ expresses either some universal property such as $\forall x\, \neg p_x \vee p_{\mathsf{s}(x)}$ or a ground property, e.g. p_0, $p_{\mathsf{s}(0)}$,... A clause of a level distinct from $\bot$ is of the form $n \not\approx \mathsf{s}^k(t) \vee C$ where the only indices occurring in C are t or $\mathsf{s}(t)$. Such a clause can be viewed as an implication: $n \approx \mathsf{s}^k(t) \Rightarrow C$. If $t = 0$ then the clause states a ground property that must be true when n is equal to k. If t is a variable, then the clause expresses an assertion that must hold when $n \geq k$ and the value of t is $n - k$. This could be written $n \geq k \Rightarrow C\{t \mapsto n - k\}$ but of course this is not a clause.

Proposition 4.4. For all $k \in \mathbb{N} \cup \{\bot\}$, the number of normalized clauses of any level k on a given finite signature Ω is at most $2^{4|\Omega|+1}$, up to a renaming and up to the duplication of literals.

Proof:
This is an immediate consequence of the fact that any normalized clause of level k is of the form $n \not\approx \mathsf{s}^{k-\epsilon+1}(t) \vee C$ or C, where C is a t-clause of depth $\epsilon \in \{0, 1\}$, thus every literal in C is of the form p_t, $\neg p_t$, $p_{\mathsf{s}(t)}$ or $\neg p_{\mathsf{s}(t)}$, for some $p \in \Omega$. Since $t \in \{0\} \cup \mathcal{V}$, we have the result. □

The following lemma states that the class of normalized clauses is preserved by the inference rules.

Lemma 4.1. If S is a set of normalized non-valid clauses then any non-valid clause in $\mathcal{R}es(S)$ is also normalized.

Proof:
Proposition 4.3 proves this is the case for the Factorization rule. Let C be a non-valid clause deduced from two normalized clauses D_1 and D_2 by resolution. By definition, D_1^{idx} and D_2^{idx} are of the form $p_{u_1} \vee D_1'$ and $\neg p_{u_2} \vee D_1'$ where u_1 and u_2 are unifiable; furthermore, since S is normalized, there exist terms t_1 and t_2 such that D_1^{idx} is a t_1-clause and D_2^{idx} a t_2-clause. For $i = 1, 2$, let $D_i'' \stackrel{\text{def}}{=} D_i^{eq}$; by hypothesis, D_i'' is either empty or of the form $n \not\approx \mathsf{s}^{k_i}(t_i)$. Thus C is of the form $(D_1' \vee D_2' \vee D_1'' \vee D_2'')\sigma$, where $\sigma = \text{mgu}(u_1, u_2)$. By definition, any atom occurring in D_i' is of the form q_{t_i} or $q_{\mathsf{s}(t_i)}$, hence u_i is either t_i or $\mathsf{s}(t_i)$, and σ must be of one of the following forms: $\emptyset$, $\{t_2 \mapsto t_1\}$, $\{t_2 \mapsto \mathsf{s}(t_1)\}$ (when $t_2 \in \mathcal{V}$ and $t_1 \in \mathcal{V} \cup \{0\}$) or $\{t_1 \mapsto \mathsf{s}(t_2)\}$ (when $t_1 \in \mathcal{V}$ and $t_2 \in \mathcal{V} \cup \{0\}$).

If σ is empty or of the form $t_2 \mapsto t_1$ or $t_1 \mapsto t_2$, then $t_1\sigma = t_2\sigma$[1]. Since σ is flat, by Proposition 4.1 (1), $D_1'\sigma$ is a $t_i\sigma$-clause, and by Proposition 4.1 (2), $D_1'\sigma \vee D_2'\sigma$ is a $t_1\sigma$-clause. Assume that D_1'' and D_2'' are not empty and that $k_1 \neq k_2$. Then C contains the disjunction $n \not\approx \mathsf{s}^{k_1}(t_1\sigma) \vee n \not\approx \mathsf{s}^{k_2}(t_2\sigma)$. Since $t_1\sigma = t_2\sigma$ and $k_1 \neq k_2$, $\mathsf{s}^{k_1}(t_1\sigma)$ and $\mathsf{s}^{k_2}(t_2\sigma)$ are not unifiable, hence C is valid. Since we assumed C is not valid, $(D_1'' \vee D_2'')\sigma$ is either empty or of the form $n \not\approx \mathsf{s}^{k_1}(t_1\sigma)$, thus C is normalized.

Assume that σ is $t_2 \mapsto \mathsf{s}(t_1)$, the proof is similar if σ is $t_1 \mapsto \mathsf{s}(t_2)$. If $D_2'\sigma$ contains an atom of the form $q_{\mathsf{s}(t_2)}$ then by definition of the ordering, $p_{t_2} \lhd q_{\mathsf{s}(t_2)}$ and by definition of the selection function, the literal $\neg p_{t_2}$ cannot be selected, which is impossible. Hence every atom in $D_2'\sigma$ is of the form $q_{t_2\sigma} = q_{\mathsf{s}(t_1)}$, hence $D_2\sigma'$ is a t_1-clause. It is clear that $D_1'\sigma$ is a t_1-clause, since σ is the identity on D_1', thus by Proposition 4.1 (2), $D_1'\sigma \vee D_2'\sigma$ is a t_1-clause. Assume that D_1'' and D_2'' are not empty and that $k_1 \neq k_2 + 1$. Then C contains the disjunction $n \not\approx \mathsf{s}^{k_1}(t_1\sigma) \vee n \not\approx \mathsf{s}^{k_2}(t_2\sigma)$. Since $t_2\sigma = \mathsf{s}(t_1)$ and $k_1 \neq k_2 + 1$, $\mathsf{s}^{k_1}(t_1\sigma)$ and $\mathsf{s}^{k_2}(t_2\sigma)$ are not unifiable, hence C is valid. Since C was assumed not to be valid, $(D_1'' \vee D_2'')\sigma$ is either empty or of the form $n \not\approx \mathsf{s}^{k_2+1}(t_1)$ and C is normalized. □

We relate the level of any clause to that of its parents: intuitively, the levels of the clauses generated by the calculus can never be decreasing. This is obvious for the Factorization rule which removes duplicate literals (see Proposition 4.3), the following lemma proves the result for the Resolution rule.

Lemma 4.2. Let S be a set of normalized clauses. Let C be a non-valid clause of level $j \in \mathbb{N}$ in $\mathcal{R}es(S)$, deduced from parent clauses D_1, D_2 in S of levels $k_1, k_2 \in \mathbb{N} \cup \{\bot\}$ respectively. The following conditions hold:

[1]Note that if σ is empty then u_1, u_2 are ground thus, since D_1, D_2 are normalized we must have $u_1 = u_2 = t_1 = t_2 = 0$.

- $k_1, k_2 \in \{\bot, j, j-1\}$.
- If C is ground then $k_1, k_2 \in \{\bot, j\}$.
- If $k_1 \neq \bot$ and $k_2 \neq \bot$ then $k_1 = k_2$.

Proof:
By definition, there exists a term $t \in \mathcal{V} \cup \{0\}$ such that C is of the form $n \not\approx \mathtt{s}^{j-1+\epsilon}(t) \vee C'$ where C' is a t-clause of depth ϵ (recall that $j \neq \bot$ by hypothesis). Similarly, if D_i is not parameter-free, then it is of the form $n \not\approx \mathtt{s}^{k_i-1+\epsilon_i}(t_i) \vee D_i'$ where D_i' is a t_i-clause of depth ϵ_i, and $\mathtt{s}^{j-1+\epsilon}(t) = \mathtt{s}^{k_i-1+\epsilon_i}(t_i)\sigma$, where σ is the mgu of two terms u_1 and u_2 occurring in selected literals in D_1 and D_2 respectively.

The terms u_1 and u_2 are of the form $0, x_i$ or $\mathtt{s}(x_i)$ where x_i is a variable, several cases are distinguished according to the form of σ, the mgu of u_1 and u_2.

- If σ is empty then $u_1 = u_2 = 0$. In this case, D_1, D_2 are ground, hence C is also ground. Furthermore, if D_i is not parameter-free, then $j - 1 + \epsilon = k_i - 1 + \epsilon_i$. By the definition of closed normalized clauses, $\epsilon = \epsilon_i = 0$, hence $j = k_i$, and if neither D_1 nor D_2 is parameter-free, then $k_1 = k_2$.
- If σ is of the form $x_1 \mapsto x_2$, $x_1 \mapsto 0$ or $x_2 \mapsto 0$, then D_1^{idx} and D_2^{idx} must be of the same depth, hence $\epsilon_1 = \epsilon_2$. Assume that D_i is not parameter-free. Since σ is flat, $j - 1 + \epsilon = k_i - 1 + \epsilon_i$, thus $k_i = j + \epsilon - \epsilon_i$. If $\epsilon = 0$ then $k_i \in \{j, j-1\}$. If $\epsilon = 1$ then C' is of depth 1. Since every literal in C' occurs either in $D_1'\sigma$ or in $D_2'\sigma$, one of them, say $D_1'\sigma$, is of depth 1. But σ is flat, therefore, in this case, D_1' is also of depth 1 and $\epsilon_1 = \epsilon_2 = \epsilon = 1$. Therefore, $k_i = j$. Thus, in all cases, $k_1, k_2 \in \{\bot, j, j-1\}$, and if neither D_1 nor D_2 is parameter-free, then $k_1 = k_2$.

  If C is ground, then either $u_1 = 0$ or $u_2 = 0$, hence $\epsilon_1 = \epsilon_2 = 0$. But in this case, necessarily, $\epsilon = 0$, hence $k_1, k_2 \in \{\bot, j\}$.
- If σ is of the form $x_1 \mapsto \mathtt{s}(x_2)$, then $u_1 = x_1$ and $u_2 = \mathtt{s}(x_2)$. In this case C is not ground. If D_1 is not parameter-free, then $j - 1 + \epsilon = k_1 - 1 + (\epsilon_1 + 1)$, thus $k_1 = j + \epsilon - \epsilon_1 - 1$. But since u_1 is maximal, necessarily, $\epsilon_1 = 0$, which implies that $k_1 = j + \epsilon - 1 \in \{j-1, j\}$. If D_2 is not parameter-free, then $j - 1 + \epsilon = k_2 - 1 + \epsilon_2$, thus $k_2 = j + \epsilon - \epsilon_2$. Furthermore, since $u_2 = \mathtt{s}(x_2)$ occurs in D_2, necessarily $\epsilon_2 = 1$ and $k_2 = j + \epsilon - 1 = k_1 \in \{j-1, j\}$.
- The case where σ is of the form $x_2 \mapsto \mathtt{s}(x_1)$ is symmetrical.

□

5. Expressive power

Although the class of normalized clause sets is strongly restricted from a syntactic point of view it is actually quite expressive. This section is devoted to showing how literals that are outside the class can be encoded into normalized clause sets.

5.1. Encoding translation on indices

We first consider atoms of the form $p_{\mathtt{s}^k(x)}$, where $k \geq 2$. Such atoms are encoded by new predicate symbols p^k_x, with the intended meaning $p^k_x \Leftrightarrow p_{\mathtt{s}^k(x)}$. The following axioms ensure that p^k_x have the intended interpretation:

$$\mathcal{A}_k \stackrel{\text{def}}{=} \{p^l_x \Leftrightarrow p^{l-1}_{\mathtt{s}(x)}, p^0_x \Leftrightarrow p_x \mid k \geq l > 0\}.$$

Note that $\mathcal{A}_k$ is finite and normalized. The following result is proved by a straightforward induction on l:

Proposition 5.1. If $I \models \mathcal{A}_k$ then for every $l \in [0, k]$, $I \models (p^l_x \Leftrightarrow p_{\mathtt{s}^l(x)})$.

5.2. Encoding inequalities and equalities

The formula $x + l > \epsilon.n + k$ (with $l, k \in \mathbb{N}$, $\epsilon \in \{0, 1\}$) is encoded by a predicate $q^{l,\epsilon,k}_x$, whose semantics are defined by the following axioms:

$\mathcal{A}'_{l',k'}$ is the set of clauses of the form:

$$q^{l,\epsilon,k}_x \Leftrightarrow q^{0,\epsilon,k}_{\mathtt{s}^l(x)}$$
$$q^{0,\epsilon,k+1}_{\mathtt{s}(x)} \Leftrightarrow q^{0,\epsilon,k}_x$$
$$\neg q^{0,\epsilon,k}_0$$
$$q^{0,0,0}_{\mathtt{s}(x)}$$
$$n \not\approx x \vee q^{0,1,0}_{\mathtt{s}(x)}$$
$$n \not\approx x \vee \neg q^{0,1,0}_x$$
$$q^{0,1,0}_{\mathtt{s}(x)} \vee \neg q^{0,1,0}_x$$

where $l \in [0, l']$, $k \in [0, k']$ and $\epsilon \in \{0, 1\}$.

Again, $\mathcal{A}'_{l',k'}$ is finite. It is not normalized due to the index $\mathtt{s}^l(x)$ in the first axiom, but it is clear that the transformation of Section 5.1 yields an equivalent normalized formulation.

Proposition 5.2. Let $I \models \mathcal{A}'_{l',k'}$. For every $(l, k, \epsilon) \in [0, l'] \times [0, k'] \times \{0, 1\}$, we have $I \models q^{l,\epsilon,k}_x \Leftrightarrow x + l > \epsilon.n + k$.

Proof:
Assume that $l = k = 0$. If $\epsilon = 0$ then the interpretation of $q^{l,\epsilon,k}$ is fixed by the third and fourth axioms and we have indeed $q^{0,0,0}_{\mathtt{s}(x)}$ (i.e. $\mathtt{s}(x) > 0$) and $\neg q^{0,0,0}_0$ (i.e. $0 \not> 0$). If $\epsilon = 1$, then the interpretation of $q^{l,\epsilon,k}$ is fixed by the fifth and sixth axioms that state that we have $q^{0,1,0}_{\mathtt{s}(x)}$ if $x = n$ (i.e. $\mathtt{s}(n) > n$) and $\neg q^{0,1,0}_x$ if $x = n$ (i.e. $n \not> n$). Furthermore, the last axiom ensures that if $q^{0,1,0}_{\mathtt{s}(x)}$ holds if $q^{0,1,0}_x$ holds. Since $q^{0,1,0}_{\mathtt{s}(n)}$ holds, this implies that $q^{0,1,0}_{\mathtt{s}(\mathtt{s}(n))}$,$q^{0,1,0}_{\mathtt{s}(\mathtt{s}(\mathtt{s}(n)))}$, ... hold, and since $\neg q^{0,1,0}_n$ does not hold, this implies that $q^{0,1,0}_{n-1}$,$q^{0,1,0}_{n-2}$, ... does not hold. Thus $q^{0,1,0}_x$ is true (resp. false) if $x > s(n)$ (resp. $x < n$).

By using the second and third axiom we can show by a straightforward induction on x that $q^{0,\epsilon,k}_x$ is equivalent to $q^{0,\epsilon,k-x}_0$ if $x < k$ and to `false` otherwise.

Then the first axiom states that $q^{l,\epsilon,k}_x$ is equivalent to $q^{0,\epsilon,k}_{x+l}$. □

The formulæ $x + l \leq \epsilon.n + k$ can be easily encoded as $x + l + 1 \not> \epsilon.n + k$. Then $x + l \approx \epsilon.n + l$ can be written as $x + l + 1 \geq \epsilon.n + l \wedge x + l \leq \epsilon.n + l + 1$.

5.3. Encoding schemata with several parameters

The restriction to one parameter entails no loss of generality. Indeed, let S be a clause set with k parameters $n_1, \ldots, n_k$. We introduce a new parameter n (which encodes the max of the n_i's). The formula $x > n_l$ are defined by atoms r^l_x, defined by the following formulæ $\mathcal{A}''$:

$$\neg r^l_x \vee r^l_{\mathsf{s}(x)}$$
$$\neg r^l_0$$
$$n \not> x \vee r^l_{\mathsf{s}(x)}$$

where $l \in [1, k]$.

Notice that the obtained formula is normalized.

Proposition 5.3. If $I \models \mathcal{A}''$ then for every $l \in [1, k]$ there is exactly one natural number n_l such that $I \models r^l_x$ iff $x > n_l$. Furthermore, $n_l \leq n$.

Proof:
Let n_l be the lowest natural number such that $I \models r^l_{\mathsf{s}(n_l)}$. n_l exists since $r^l_{\mathsf{s}(n_l)}$ holds (by the third axiom). We have $I \models \neg r^0_l$, thus by minimality of n_l $r^l_{n_l}$ cannot hold. Then by using the first axiom we show that $I \models r^l_x$ for every $x > n_l$ and that $I \models \neg r^l_x$ for every $x \leq n_l$. □

Then the formula $x \approx n_l$ is written $x + 1 > n_l \wedge x \not> n_l$.

5.4. Inductive definitions

Now, consider an inductive definition: $\phi_0 \Leftrightarrow B$ and $\phi_{k+1} \Leftrightarrow I$ if $k \geq 0$ where B and I denotes formulæ and where I contains atoms of the form p_k or $p_{\mathsf{s}(k)}$ or inductively defined formulæ ψ_k (with possibly $\phi = \psi$). This definition can be easily expressed in our formalism by considering every such formula ϕ as a predicate symbol:

$$\phi_0 \Leftrightarrow B$$

$$\phi_{\mathsf{s}(x)} \Leftrightarrow I.$$

One gets a 1-normalized clause set by translation into clausal form. More complex inductive definition may be encoded in the same way. In particular, formulæ of the form $\bigvee_{i=a}^{b} \phi_i$ are easily encoded by an atom $\psi_{b-a+1} = \bigvee_{x=1}^{b-a+1} \phi_{x-1+a}$ defined as follows:

$$\psi_0 \Leftrightarrow \texttt{false}$$

$$\psi_{x+1} \Leftrightarrow \phi_{x-1+a} \vee \psi_x.$$

After translation into clausal normal form and after removing the translation as explained in Section 5.1, one gets a normalized clause set.

5.5. Regular schemata

Using the previous transformations it is easy to prove that every regular schema (in the sense of [6]) can be encoded into a normalized clause set, i.e. that for every regular schema ϕ one can construct a normalized clause set S such that ϕ and S are equisatisfiable in a strong sense: every model of S is also a model of ϕ and every model of ϕ can be extended into a model of S. Furthermore, the size of S is linear w.r.t. that of ϕ (if natural numbers are encoded as unary terms in the original schema[2] and if a structural-preserving transformation is used to compute clausal forms).

Furthermore, normalized clause sets allow one to express properties that cannot be stated as regular schemata, for instance the formula $\bigwedge_{x=0}^{\infty} p_x$ is easily expressed by the normalized clause set $\{p_x\}$ but it is not a regular schema (due to the unbounded conjunction).

6. Loop detection

In this section we extend the calculus proposed in Section 3 by defining a *loop detection rule* that is capable of pruning infinite derivations. To help the reader grasp the following definitions and lemmata, we first provide an informal high level description of the rule. Let S be a clause set. The set of clauses $S' = \{C \mid S \vdash C\}$ generated from S can be partitioned into an infinite sequence of clause sets $S'|_\bot$, $S'|_1, \ldots, S'|_k, \ldots$, where for every $k \in \mathbb{N} \cup \{\bot\}$, $S'|_k$ contains exactly the clauses of level k in S' (see Definition 4.3). Note that by Lemma 4.2, the clauses in $S'|_k$ (where $k > 0$) must be generated either from clauses in $S'|_k \cup S'|_\bot$ or from clauses in $S'|_{k-1} \cup S'|_\bot$ (except of course for those that already occur in the original clause set S). By definition of the notion of level, each of the sets $S'|_k$ can be viewed as a formula of the form $n \not\approx \mathsf{s}^k(x) \vee S_k$, where S_k is a conjunction of x-clauses[3] Since the set of x-clauses is finite there must exist, by the pigeonhole principle, two natural numbers i and $j > 0$ such that $S_i = S_{i+j}$. This means that the set of clauses generated at level i is equivalent to the set of clauses generated at level $i + j$, up to a shift on the parameter n: indeed if $S_i = S_{i+j}$ then $n \not\approx \mathsf{s}^{i+j}(x) \vee S_{i+j}$ is equivalent to $n \not\approx \mathsf{s}^i \vee S_i$, up to the shift $n \mapsto n+j$. We will show that, under some particular conditions, the existence of such a "cycle" in the derivation permits to deduce an upper bound on the value of the parameter n: if S is satisfiable then it has a model I such that $I(n) < i + j$. The intuition is that if a model I such that $I(n) > i+j$ exists, then it is possible to obtain another model J such that $J(n) = I(n) - j$, by applying the translation $n \mapsto n-j$ on I. This implies that the constraint $n < i+j$ (written $n \not\approx \mathsf{s}^{i+j}(x)$ in clausal form) can be safely added to S'. It is then clear that every clause of a level strictly greater than $i + j$ is redundant w.r.t. $n < i+j$, since $n \not\approx \mathsf{s}^{i+j}(t) \vee C$ is obviously subsumed by $n \not\approx \mathsf{s}^{i+j}(x)$. Consequently, once the *pruning clause* $n < i + j$ has been generated, only finitely many non-redundant clauses can be deduced.

Obviously, this result does not hold for any arbitrary derivation: for instance if the initial clause set S contains clauses of arbitrary levels, then the value of the parameter must also be arbitrary. We now give the formal definition of the conditions that must be satisfied by the initial clause set S. We begin by slightly restricting the notion of redundancy:

[2] Otherwise a schema as simple as $p_1 \underbrace{0 \ldots 0}_{n}$ could not encoded in linear size.

[3] to simplify the informal presentation we assume that every clause $C \in S_k$ is of depth 1.

Definition 6.1. A clause C is *level-redundant* w.r.t. a set of clauses S (written $C \sqsubseteq_1 S$) iff for every ground substitution σ of domain $var(C)$ there exist clauses $C_1, \ldots, C_m$ in S such that:

- If C is not ground then neither are $C_1, \ldots, C_m$.
- The clauses in $C_1, \ldots, C_m$ are either of level $\bot$ or of the same level as C.
- There exist m substitutions $\theta_1, \ldots, \theta_m$ such that $C_1\theta_1, \ldots, C_m\theta_m \models C\sigma$ and $C_1\theta_1, \ldots, C_m\theta_m \preceq C\sigma$.

If S' is a set of clauses, then we write $S' \sqsubseteq_1 S$ iff $\forall C \in S', C \sqsubseteq_1 S$.
A set of clauses S is *saturated up to level* i iff $\mathcal{R}es(S|_{[\bot,i[}) \sqsubseteq_1 S$.

For instance, p_x is redundant w.r.t. $\{p_0, p_{\mathsf{s}(x)}\}$ but not level-redundant (since p_0 is ground and p_x is not). The clause $n \not\approx \mathsf{s}(x) \vee p_x \vee p_{\mathsf{s}(x)}$ is redundant w.r.t. $\{n \not\approx \mathsf{s}(x) \vee p_x\}$ but not level-redundant since $level(n \not\approx \mathsf{s}(x) \vee p_x) = 1$ and $level(n \not\approx \mathsf{s}(x) \vee p_x) = 2$.

The following property is an immediate consequence of Definition 6.1:

Proposition 6.1. If C is a clause of level i that is level-redundant w.r.t. a set of clauses S, then $S|_i \cup S|_\bot \models C$.

Definition 6.2. If S is a set of clauses and i is a natural number, $shift(S, i)$ denotes the set of clauses of the form $n \not\approx \mathsf{s}^{i+j}(x) \vee C$, where $x \in \mathcal{V}$ and $n \not\approx \mathsf{s}^j(x) \vee C \in S$.

Example 6.1. Consider the clause set

$$S = \{n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee \neg p_x \vee p_{\mathsf{s}(x)}, n \not\approx \mathsf{s}(x) \vee q_x, n \not\approx 0 \vee p_x, \neg r_x\}.$$

We have for example:

$$\begin{aligned}
shift(S,0) &= \{n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee \neg p_x \vee p_{\mathsf{s}(x)}, n \not\approx \mathsf{s}(x) \vee q_x\},\\
shift(S,1) &= \{n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(x))) \vee \neg p_x \vee p_{\mathsf{s}(x)}, n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee q_x\},\\
shift(S,2) &= \{n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(x)))) \vee \neg p_x \vee p_{\mathsf{s}(x)}, n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(x))) \vee q_x\}.
\end{aligned}$$

Note that $\neg r_x$ or $n \not\approx 0 \vee p_x$ cannot occur in $shift(S, i)$ because they contain no literal of the form $n \not\approx \mathsf{s}^j(x)$, where $x \in \mathcal{V}$.

Lemma 6.1 will set the foundations for the definition of the loop detection rule. This lemma applies when we detect that the clauses of a certain level i in S are logically entailed by those of a level $i+j > i$, up to a shift on parameter n. This means that any model of $S|_{i+j}$ is also a model of $S|_i$, up to the shift $n \mapsto n - j$. If we assume further that S is saturated up to level i, then we prove that for any model I of S such that $I(n) \geq i + j$, it is possible to construct an interpretation J of S such that $J(n) = I(n) - j < I(n)$, and which coincides with I for $S|_{[i,\infty[}$. Then, since S is saturated up to level i, it is possible to show that this interpretation can be extended into a model of S. Thus, satisfiability is preserved when the value of n is constrained to be strictly lower than $i + j$. As explained before, adding this assertion to S makes every clause of a level greater than $i + j$ redundant since it is subsumed by $n < i + j$.

Definition 6.3. Given a parameter n and an integer l, we write $n < l$ as a shorthand for the clause $n \not\approx \mathtt{s}^l(x)$ and call such a clause a *pruning clause*.

Given a set of clauses S a pruning clause C is a *compatible with* S if S is satisfiable exactly when $S \cup \{C\}$ is.

Lemma 6.1. Let S be a set of normalized clauses of parameter n and assume i, j are natural numbers satisfying the following conditions:

1. S is saturated up to level i.
2. $S|_i \cup S|_\perp \models S|_{[i,\infty[}$.
3. $S|_i$ contains no ground clause.
4. $j \neq 0$.
5. $S|_\perp \cup S|_{i+j} \models \mathit{shift}(S|_i, j)$.

Then $n < i + j$ is compatible with S.

Proof:
Let I be an interpretation satisfying S. W.l.o.g. we assume that the valuation of n in I is minimal (w.r.t. the usual ordering on natural numbers), i.e. for every interpretation J such that $J(n) < I(n)$, we have $J \not\models S$. Assume that $I \not\models \{n < i + j\}$, i.e., that $I(n) \geq i + j$ (which implies that $I(n) > 0$ since $j \neq 0$ by assumption). We construct an interpretation J as follows:

- $J(n) \stackrel{\text{def}}{=} I(n) - j$. By Condition 4, this implies that $J(n) < I(n)$ thus by hypothesis on the minimality of the valuation of n in I, $J \not\models S$.
- The value of the ground atom p_k (for $k \in \mathbb{N}$) is defined by induction on the ordering $\prec$:
  - If $k \leq I(n) - i - j + 1$ then $J(p_k) \stackrel{\text{def}}{=} I(p_k)$.
  - If $k > I(n) - i - j + 1$, then assume $J(q_{k'})$ is defined for every atom $q_{k'}$ such that $q_{k'} \prec p_k$. We set the value of $J(p_k)$ to `true` iff $S|_{[\perp,i[}$ contains a clause that admits a ground instance of the form $p_k \vee C$, such that for all indexed atoms q in C, $q \prec p_k$, and $J \not\models C$. Note that the value of C^{eq} is well-defined since it only depends on $J(n)$, and that by induction, the value of C^{idx} is also well-defined.

Let $S' = S|_{[\perp,i]}$. Since $S|_i \cup S|_\perp \models S|_{[i,\infty[}$ by Condition 2, and since $S = S|_{[\perp,i]} \cup S|_{[i,\infty[}$ by definition, necessarily, $S' \models S$. We show that $J \models S'$, thus contradicting the fact that $J \not\models S$. Let D be a clause in S' and θ be a ground substitution such that $J \not\models D\theta$. By definition, θ is either empty (if D is ground) or of the form $x \mapsto \mathtt{s}^k(0)$ (or simply $x \mapsto k$), where x is the unique variable in D.

First assume that D is of level i, i.e., that $D \in S|_i$. Then by Condition 5, $S|_\perp \cup S|_{i+j} \models \mathit{shift}(S|_i, j)$, and since $S|_\perp \cup S|_{i+j} \subseteq S$ and $I \models S$, this implies that $I \models \mathit{shift}(S|_i, j)$. By definition, D is of the form $n \not\approx \mathtt{s}^{i-1+\epsilon}(t) \vee D'$, where D' is a t-clause of depth ϵ. But D is not ground by Condition 3, thus t must be a variable x. This implies that $\mathit{shift}(S|_i, j)$ contains the clause $n \not\approx \mathtt{s}^{i-1+\epsilon+j}(x) \vee D'$, which is true

in I. Since $J \not\models D\theta$, necessarily $J(n) = i - 1 + \epsilon + k$, and $I(n) = J(n) + j = i - 1 + \epsilon + k + j$. By hypothesis $I \models (n \not\approx \mathtt{s}^{i-1+\epsilon+j}(x) \vee D')\theta$, therefore, $I \models D'\theta$. The indices occurring in D' are either x or $\mathtt{s}(x)$, thus the indices occurring in $D'\theta$ are either k or $k+1$. Furthermore, D is of depth $\epsilon = 1$ if an index $k+1$ occurs in $D'\theta$, and otherwise, $\epsilon = 0$. Hence, the maximal index that can occur in D' is $I(n) - i - j + 1$. By definition, I and J coincide on the propositions p_l such that $l \leq I(n) - i - j + 1$. Thus $J \models D'\theta$, which implies that $J \models D\theta$, a contradiction.

Now assume that $D \in S|_{[\bot,i[}$. W.l.o.g. we assume that $D\theta$ is the least instance w.r.t. $\prec$ of a clause in $S|_{[\bot,i[}$ such that $J \not\models D\theta$. Note that D cannot be empty since otherwise S would be unsatisfiable. If D is purely equational then D must be of the form $n \not\approx \mathtt{s}^l(t)$ for some $l < i$ and $t \in \mathcal{V} \cup \{0\}$ (because $D \in S|_{[0,i[}$ and D is normalized). Then D is equivalent either to $n < l$ (if $t \in \mathcal{V}$) or to $n \neq l$ (if $t = 0$). In the first case, we have $I \not\models D$, which is impossible since I is a model of S and $D \in S' \subseteq S$. In the second case, since $l < i$ and $J(n) = I(n) - j \geq i$ we deduce $J \models D$, a contradiction since we assumed $J \not\models D$.

We thus assume D contains at least one index literal, and denote by l_u the maximal index literal in D (which is unique by Proposition 4.2). D is of the form $l_u \vee D' \vee D''$, where D' is parameter-free, D'' is purely equational (any of them could be empty), and for all literals $l \in D'\theta$, $l \preceq l_u\theta$. By definition of the ordering $\prec$, this implies that the index of every literal $l \in D'\theta$ is less or equal to $u\theta$.

We distinguish two cases depending on the value of $u\theta$.

- If $u\theta \leq I(n) - i - j + 1$, then every index in $D'\theta$ is also less or equal to $I(n) - i - j + 1$. By construction of J, this implies that I and J coincide on $(l_u \vee D')\theta$. Thus, if I satisfies $(l_u \vee D')\theta$, then so does J, and this case is impossible. Hence, $I \not\models (l_u \vee D')\theta$, so that necessarily, $I \models D''\theta$. By hypothesis, D is a normalized t-clause in $S|_{[0,i[}$ that is not valid, hence D'' must be of the form $n \not\approx \mathtt{s}^m(t)$, where $t \in \mathcal{V} \cup \{0\}$, and if ϵ stands for the depth of $l_u \vee D'$, then $\mathit{level}(D) = m - \epsilon + 1 < i$. Since $J \not\models D''\theta$, either $J(n) = m$ (if $t = 0$), or $J(n) = m + k$ (if $t = x$); but the first case is impossible because $I(n) > i + j$ by hypothesis and $J(n) = I(n) - j > i$ by construction. Thus, $J(n) = m + k$, so that $k = J(n) - m > J(n) - i$. Since u is either x or $\mathtt{s}(x)$, we deduce that $u\theta$ is either k or $k+1$. In the latter case, $u\theta > J(n) - i + 1$. In the former case, $l_u \vee D'$ is of depth 0 by Proposition 4.2, and $\mathit{level}(D) = m + 1 < i$ so that $u\theta = k = J(n) - m > J(n) - i + 1$. Therefore, in every case, $u\theta > J(n) - i + 1 \geq I(n) - j - i + 1$ which contradicts our assumption.

- Now assume that $u\theta > I(n) - i - j + 1$. Since $D\theta$ is the minimal instance of a clause in $S|_{[\bot,i[}$ that is false in J and since S is saturated up to level i, we may assume that $D'\theta \prec l_u\theta$. Indeed, every literal in $D'\theta$ must be smaller than $l_u\theta$, and if l_u occurs in D' then the factorization rule applied to D would generate a clause with a strictly smaller instance that is false in J. Literal l_u cannot be positive because if this were the case, D would be of the form $p_u \vee D' \vee D'$ and by construction, $p_{u\theta}$ would be interpreted to $\mathtt{true}$ in J and $D\theta$ would be true in J. Therefore, l_u is a negative literal $\neg p_u$ and $J(p_{u\theta}) = \mathtt{true}$. Again by construction, $S|_{[\bot,i[}$ contains a clause of the form $E = p_v \vee E' \vee E''$ where E' is parameter-free and E'' is purely equational, there exists a substitution θ' such that $v\theta' = u\theta$ and $\forall l \in E', l\theta' \prec p_{v\theta'}$, and $J \not\models (E' \vee E'')\theta'$.

  The resolution rule applied to D and E generates a clause of the form $F = (E' \vee E'' \vee D' \vee D'')\sigma$, where σ is the mgu of u and v. An instance of this clause is $F' = E''\theta' \vee D''\theta \vee E'\theta' \vee D'\theta$, which is strictly smaller than $D\theta$ and is false in J. Since S is saturated up to level i, F is level-redundant in S, hence there exist clauses $C_1, \ldots, C_m \in S$ and substitutions $\sigma_1, \ldots, \sigma_m$ such that

$C_1\sigma_1, \ldots, C_m\sigma_m \preceq F'$ and $C_1\sigma_1, \ldots, C_m\sigma_m \models F'$. By Lemma 4.2, F is of level at most i, hence, so are $C_1, \ldots, C_m$. For all $j = 1, \ldots, m$, if C_j is of level i, then $J \models C_j\sigma_j$ by the first item above, and if the level of C_j is strictly less than i, then since $C_j\sigma_j \preceq F' \prec D\theta$, the minimality condition on $D\theta$ entails again that $J \models C_j\sigma_j$. Therefore, in all cases, $J \models F'$, which is impossible.

□

In practice, Lemma 6.1 by itself is not sufficient to define a suitable loop detection rule, indeed, Condition 2 is difficult to check and Condition 3 cannot be guaranteed. We now exhibit sufficient conditions guaranteeing the existence of a loop in a derivation.

Definition 6.4. A set of clauses S is *k-reducible* iff there exists a k-normalized set of clauses S' such that for every clause $C \in S$ there exists a derivation $C_1, \ldots, C_n$ from S' such that $C_n = C$ and $C_1, \ldots, C_n$ are level-redundant w.r.t. S.

Lemma 6.2. Let S be a k-reducible set of normalized clauses. For every $i \geq k$, $S|_i \cup S|_\perp \models S|_{i+1}$; hence $S|_i \cup S|_\perp \models S|_{[i,\infty[}$.

Proof:
Since S is k-reducible, there exists a k-normalized set of clauses S' such that every clause C in S of level $i+1$ is derivable from S', and every clause in the derivation is level-redundant w.r.t. S. We prove by induction on the length of the derivation that for every clause C' of level $i+1$ occurring in this derivation, $S|_i \cup S|_\perp \models C'$. Since C' is of level $i+1 > k$, the length of the derivation is greater than 0. We assume C' is deduced by resolution from parent clauses D_1, D_2, the case where C' is generated by the factorization rule is similar. By Lemma 4.2, D_1 and D_2 can be of levels $\perp$, i or $i+1$. If D_1 is of level $\perp$ or i, then $S|_i \cup S|_\perp \models D_1$ by Proposition 6.1. Otherwise by the induction hypothesis, $S|_i \cup S|_\perp \models D_1$. The same property holds for D_2; consequently, the parents of C' are logical consequences of $S|_i \cup S|_\perp$, hence $S|_i \cup S|_\perp \models C'$.

Since $S|_i \cup S|_\perp \models S|_{i+1}$, we also have $S|_i \cup S|_\perp \models S|_{i+1} \cup S|_\perp$, and a straightforward induction proves that for all $j \geq i$, $S|_i \cup S|_\perp \models S|_j$. Since $S|_{[i,\infty[} \stackrel{\text{def}}{=} \bigcup_{j \geq i} S|_j$, the second part of the lemma also holds. □

Definition 6.5. For all $i > 0$, we denote by $S|_i^\star$ the set of non-ground clauses of level i in S.

Lemma 6.3. Let S be a k-reducible set of normalized clauses. If $i > k$, then $S|_i^\star \cup S|_\perp \models S|_i$.

Proof:
Let $C \in S|_i$. Since S is k-reducible, by definition there exists a derivation of C from a set of k-normalized clauses, and every clause in this derivation is level-redundant w.r.t. S. We prove by induction on the length of the derivation that for every clause C' of level i occurring in the derivation, $S|_i^\star \cup S|_\perp \models C'$. This is obvious if C' is not ground. If C is ground, then since $i > k$, we assume C' was deduced from two clauses D_1 and D_2 by resolution (the case where C' was deduced by factorization is similar), and by Lemma 4.2, D_1 and D_2 are of level i or $\perp$. By the induction hypothesis, $S|_i^\star \cup S|_\perp \models D_1, D_2 \models C'$. □

Proposition 6.2. Let S be a set of clause and C be a clause of a level i, such that $C \sqsubseteq_1 S$. If C is not ground then $C \sqsubseteq_1 S|_\perp \cup S|_i^\star$.

Proof:
By definition there exist m clauses $C_1, \ldots, C_m$ in S, of levels $\bot$ or i such that for every substitution σ there exist n substitutions $\theta_1, \ldots, \theta_m$ such that $C\theta_1, \ldots, C\theta_m \models C$ and $C_1\theta_1, \ldots, C_m\theta_m \preceq C\sigma$. Furthermore, since C is not ground, neither are $C_1, \ldots, C_m$. For $i = 1, \ldots, m$, if C_i is of level $\bot$ then it occurs in $S|_\bot$ and if it is of level i, then, since it is not ground, it occurs in $S|^\star_i$. Thus $C_1, \ldots, C_m \in S|_\bot \cup S|^\star_i$ and $C \sqsubseteq_1 S|_\bot \cup S|^\star_i$. □

Definition 6.6 states the conditions that permit the generation of pruning clauses.

Definition 6.6. We denote by $\mathcal{P}r(S)$ the set of pruning clauses $n < i + j$ such that:

1. $j \neq 0$ and $i > k$.
2. S is saturated up to level i.
3. $S|^\star_{i+j} = \mathit{shift}(S|^\star_i, j)$ (up to a renaming of variables).

Theorem 6.1. Let S be a k-reducible set of normalized clauses. Any clause in $\mathcal{P}r(S)$ is compatible with S.

Proof:
Let i, j be natural numbers satisfying the conditions of Definition 6.6. It is clear that if $S \cup \{n < i + j\}$ is satisfiable then so is S, we now prove the other implication and assume that S is satisfiable. Let $T = S|_{[\bot,i[} \cup \bigcup_{l \geq i} S|^\star_l$. By definition, $T|_l = S|_l$ if $l < i$, and $T|_l = S|^\star_l$ if $l \geq i$. Furthermore, since S is k-reducible, so is T which is also normalized, and by Lemma 6.3, $S \equiv T$. Hence it suffices to show that $T \cup \{n < i + j\}$ is satisfiable. We prove that T satisfies the application conditions of Lemma 6.1.

1. T **is saturated up to level** i. Let C be a clause deduced from clauses in $T|_{[\bot,i[} = S|_{[\bot,i[}$. Since S is saturated up to level i, C is level-redundant with respect to S, i.e., $C \sqsubseteq_1 S$. If the level of C is strictly less than i, then we deduce that $C \sqsubseteq_1 S|_{[\bot,i[}$, hence $C \sqsubseteq_1 T$. Otherwise, by Lemma 4.2, C cannot be ground, thus $C \sqsubseteq_1 S|_\bot \cup S|^\star_i$ by Proposition 6.2.
2. $T|_i \cup T|_\bot \models T|_{[i,\infty[}$. This is a direct application of Lemma 6.2.
3. $T|_i$ **contains no ground clause**. This is obviously the case by definition of $S|^\star_i$.
4. $j \neq 0$. This is the case by hypothesis.
5. $T|_\bot \cup T|_{i+j} \models \mathit{shift}(T|_i, j)$. By hypothesis, $S|^\star_{i+j} = \mathit{shift}(S|^\star_i, j)$ and by construction, $S|_\bot \cup S|^\star_{i+j} = T|_\bot \cup T|_{i+j}$. Thus $T|_\bot \cup T|_{i+j} \models \mathit{shift}(S|^\star_i, j)$. Since $S|^\star_i = T|_i$, we have $\mathit{shift}(S|^\star_i, j) = \mathit{shift}(T|_i, j)$, hence the result.

□

Note that Conditions 2 and 3 in Definition 6.6 can be tested in polynomial time w.r.t. the size of the clause sets. Theorem 6.1 only applies to k-reducible clause sets. However, this is not really restrictive because, as we shall see, any clause set generated from a finite set of normalized clauses S' by the resolution calculus must be k-reducible, where k is the maximal level of the clauses in S' (see Lemma 7.1 for details).

Example 6.2. Consider the clause set $S = \{1, \ldots, 8\}$ in the example of page 7 (generated from the formula $p_n \wedge (\forall x\, p_{\mathtt{s}(x)} \Rightarrow p_x) \wedge \neg p_0$):

1	$n \not\approx x \vee p_x$	(level 1)	
2	$p_y \vee \neg p_{\mathtt{s}(y)}$	(level $\bot$)	
3	$\neg p_0$	(level $\bot$)	
4	$n \not\approx \mathtt{s}(y) \vee p_y$	(resolution, 1,2)	(level 2)
5	$n \not\approx 0$	(resolution, 1,3)	(level 1)
6	$n \not\approx \mathtt{s}(0)$	(resolution, 4,3)	(level 2)
7	$n \not\approx \mathtt{s}(\mathtt{s}(y)) \vee p_y$	(resolution, 2,4)	(level 3)
8	$n \not\approx \mathtt{s}(\mathtt{s}(0))$	(resolution, 7,3)	(level 3)

The initial clauses $1, 2, 3$ are of level $\bot$, 0 or 1 thus S is 1-reducible. Let $i = 2$ and $j = 1$, then $S|_i^\star = \{n \not\approx \mathtt{s}(y) \vee p_y\}$ and $S|_{i+j}^\star = \{n \not\approx \mathtt{s}(\mathtt{s}(y))\}$ (clauses 6 and 8 are dismissed since they are ground, according to Definition 6.5). Thus $\mathit{shift}(S|_i^\star, j) = \{n \not\approx \mathtt{s}(\mathtt{s}(y)) \vee p_y\} = S|_{i+j}^\star$, and Condition 3 of Definition 6.6 trivially holds. Furthermore, it is straightforward to check that S is saturated up to level i.

Hence the pruning rule can be applied and yields: $n \not\approx \mathtt{s}(\mathtt{s}(\mathtt{s}(x)))$, i.e. $n < 3$. Together with clauses 5, 6 and 8, we obtain a finite and purely equational clause set, whose unsatisfiability can be tested by standard algorithms. Therefore the initial clause set is unsatisfiable.

Example 6.3. Consider the following schema:

$$p_0 \vee \bigwedge_{x=0}^{n} (p_x \Rightarrow q_x) \wedge \bigwedge_{x=0}^{n} (q_x \Leftrightarrow \neg q_{\mathtt{s}(x)}) \wedge \neg q_n \wedge \neg q_{\mathtt{s}(n)}.$$

This schema can be encoded by the following clause set (see Section 5 for details):

1	p_0	(level $\bot$)
2	$\neg p_x \vee q_x$	(level $\bot$)
3	$\neg q_x \vee \neg q_{\mathtt{s}(x)}$	(level $\bot$)
4	$q_x \vee q_{\mathtt{s}(x)}$	(level $\bot$)
5	$n \not\approx x \vee \neg q_x$	(level 1)
6	$n \not\approx x \vee \neg q_{\mathtt{s}(x)}$	(level 0)

We apply the calculus (assuming that $p \prec q$):

7	q_0	(resolution 1,2)	(level $\bot$)
8	$n \not\approx 0$	(resolution $7, 5$)	(level 1)
9	$n \not\approx \mathtt{s}(x) \vee q_x$	(resolution 5,4)	(level 2)
10	$\neg p_{\mathtt{s}(x)} \vee \neg q_x$	(resolution $2, 3$)	(level $\bot$)
11	$n \not\approx \mathtt{s}(\mathtt{s}(x)) \vee \neg q_x$	(resolution $9, 3$)	(level 3)
12	$n \not\approx \mathtt{s}(\mathtt{s}(0))$	(resolution $11, 7$)	(level 3)
13	$n \not\approx \mathtt{s}(\mathtt{s}(\mathtt{s}(x))) \vee q_x$	(resolution $11, 4$)	(level 4)
14	$n \not\approx x \vee q_x$	(resolution $4, 6$)	(level 1)
15	$n \not\approx \mathtt{s}(x) \vee \neg q_x$	(resolution $3, 14$)	(level 2)
16	$n \not\approx \mathtt{s}(0)$	(resolution $7, 15$)	(level 2)
17	$n \not\approx \mathtt{s}(\mathtt{s}(x)) \vee q_x$	(resolution $4, 15$)	(level 3)
18	$n \not\approx \mathtt{s}(\mathtt{s}(\mathtt{s}(x))) \vee \neg q_x$	(resolution $17, 3$)	(level 4)
19	$n \not\approx \mathtt{s}(\mathtt{s}(\mathtt{s}(0)))$	(resolution $18, 7$)	(level 4)

Let $i = 2$, $j = 2$, and let $S' = \{1\text{-}18\}$. S' is saturated up to level 2. We have $S'|_2^\star = \{9, 15\}$, $S'|_4^\star = \{13, 18\}$, thus $\mathit{shift}(S|_2^\star, 2) = S|_4^\star$. The pruning rule applies and generates: $n \not\approx \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(x))))$ i.e. $n < 4$. Together with clauses 8, 10, 12 and 19, this yields a finite, purely equational, unsatisfiable, clause set.

7. Termination

In this section we define a *pruning rule* based on Theorem 6.1 and we show that the addition of this rule makes the calculus terminating, provided the rules are applied in a fair way.

The notion of level-redundancy is useful only to apply the pruning rule (Condition 1 in Definition 6.6 requires clause set to be partially saturated up to level-redundancy). Once a pruning clause has been generated, the more general and standard notion of redundancy can be used instead. This yields the following:

Definition 7.1. A clause C is *deletable* in a clause set S iff C is level-redundant in S or if C is redundant in S and S contains a pruning clause.

In order to take deletion rules into account, we define derivations as sequences of clause sets:

Definition 7.2. A *pruning derivation from* a clause set S is a (possibly infinite) sequence of clause sets $(S_i)_{i \in I}$, with $I = [0, n]$ or $I = \mathbb{N}$, such that $S_0 = S$ and for every $i \in I \setminus \{0\}$, one the following conditions holds:

- $S_i = S_{i-1} \cup \{C\}$ where $C \in \mathcal{R}es(S_{i-1})$ (`deduction` step).
- $S_i = S_{i-1} \cup \{C\}$ where $C \in \mathcal{P}r(S_{i-1})$ (`pruning` step).

- $S_i = S_{i-1} \setminus \{C\}$, where C is deletable in S_i (`deletion` step).

We write $S \vdash_p C$ if there exists a pruning derivation $S_1, \ldots, S_n$ from S such that $C \in S_n$.
The *limit* of a pruning derivation is the set clauses S_∞ defined by:

$$S_\infty \stackrel{\text{def}}{=} \bigcup_{j \in I} S_j.$$

In practice identifying all level-redundant or redundant clauses is unfeasible, thus only the clauses that are valid or subsumed are deleted.

A derivation $(S_i)_{i \in I}$ is *non-redundant* iff the deletion steps are applied with the highest priority and if the `deduction` and `pruning` step are applied only if C is not deletable in S_{i-1}. It is *fair* iff for every level l there exists an $i \in I$ such that all the clauses deducible from $S|_{[\bot,l]}$ are deletable in S_i (this may be enforced, for instance, by applying the inference rules with the highest priority on the clause of lower levels).

Lemma 7.1. Let $(S_i)_{i \in I}$ be a pruning derivation from a k-normalized clause set, and assume the S_i's $(i \in I)$ contain no pruning clause. Then for every $i \in I$, S_i is k-reducible.

Proof:
By definition every clause C in S_i can be deduced from clauses in S. Thus there exists a derivation $C_1, \ldots, C_n$ from S such that $C_n = C$. By definition, for every clause C_j $(1 \leq j \leq n)$, either $C_j \in S_i$, or C_j was deleted by a previous `deletion` step. In both cases C_j must be level-redundant with respect to S_i. □

Proposition 7.1. If C is redundant (resp. level-redundant) w.r.t. S then any clause D redundant (resp. level-redundant) w.r.t. $S \cup \{C\}$ is also redundant (resp. level-redundant) w.r.t. S.

Proof:
Let σ be ground substitution of domain $var(D)$. Since D is redundant w.r.t. $S \cup \{C\}$ there exist n clauses $D_1, \ldots, D_n \in S \cup \{C\}$ and n substitutions $\sigma_1, \ldots, \sigma_n$ such that $D_1\sigma_1, \ldots, D_n\sigma_n \models C\sigma$ and $D_1\sigma_1, \ldots, D_n\sigma_n \leq C\sigma$. Moreover, if D is level-redundant in $S \cup \{C\}$, then the $D_1, \ldots, D_n$ are of level $\bot$ or of the same level as D.

Let $i \in [1, n]$. We define a set of ground clauses E_i as follows. If $D_i \in S$ then $E_i \stackrel{\text{def}}{=} \{D_i\sigma_i\}$. Otherwise, by definition we must have $D_i = C$, thus there exist m clauses $C_1, \ldots, C_m \in S \cup \{C\}$ and m substitutions $\theta_1, \ldots, \theta_m$ such that $D_1\sigma_1, \ldots, D_m\theta_m \models D_i\sigma_i$ and $D_1\theta_1, \ldots, D_m\theta_m \leq D_i\sigma_i \leq C\sigma$. Moreover, if C is level-redundant, then the $C_1, \ldots, C_n$ are of level $\bot$ or of the same level as C. We define $E_i \stackrel{\text{def}}{=} \{D_1\theta_1, \ldots, D_m\theta_m\}$.

By construction, the clauses in $\bigcup_{i=1}^n E_i$ are ground instances of clauses in S, that are lower than $C\sigma$. Moreover, we have $\bigcup_{i=1}^n E_i \models \bigcup_{i=1}^n D_i\sigma \models C\sigma$ and if C and D are level-redundant in S and $S \cup \{C\}$ respectively, then every clause in $\bigcup_{i=1}^n E_i$ is of level $\bot$ or of the same level as C. □

Corollary 7.1. Let $(S_j)_{j \in I}$ is a fair, non-redundant pruning derivation. If C is deletable in S_j for some $j \in I$, then C is deletable in every set S_i such that $i \in I, i \geq j$.

Proof:
The proof is by induction on $i - j$. It is obvious if $i = j$. If $i > j$, then by the induction hypothesis C is deletable in S_{i-1}. Assume that S_{i-1} contains no pruning clause. Then C is level-redundantw.r.t. S_{i-1}. All the clauses occurring in S_i but not in S_{i-1} must be deletable in S_{i-1}, hence level-redundant w.r.t. S_{i-1}. By **??**, C is level-redundant w.r.t. S_i, hence C is deletable in S_i.

Assume that S_{i-1} contains a pruning clause $n \not\approx \mathsf{s}^k(x)$. Then C is redundant w.r.t. S_{i-1}. All the clauses occurring in S_i but not in S_{i-1} must be deletable in S_{i-1}, hence redundant w.r.t. S_{i-1}. By **??**, C is redundant w.r.t. S_i. If S_i contains a pruning clause then the proof is completed, since C is deletable in S_i. Otherwise, by definition, a `deletion` step must be applied on $n \not\approx \mathsf{s}^k(x)$, thus this clause is redundant w.r.t. S_{i-1}. Then S_{i-1} contains m clauses $D_1, \ldots, D_m$ (distinct from $n \not\approx \mathsf{s}^k(x)$) and m substitutions θ_j such that, in particular, $D_j\theta_j \preceq C\{x \mapsto t\}$, where t is any ground term of size greater than any term occurring in D_j. By definition of the ordering (since equational literals are not comparable) we must have $D_j\theta_j = \Box$ or $D_j\theta_j = C\{x \to t\}$. If $\Box \in S_i$ then C is level-redundant in S_i hence the proof is completed. Since t does not occur in D_j, D_j cannot be ground, hence D_j must be of the form $n \not\approx \mathsf{s}^{l'}(x)$, hence must be a pruning clause, which contradicts our assumption. □

Proposition 7.2. Let $(S_j)_{j\in I}$ is a non-redundant pruning derivation from a finite set of clauses. If I is infinite then so is S_∞.

Proof:
If S_∞ is finite and I is infinite, there exists $i \in I$ such that no "new" clauses are generated after step i, i.e. such that for every clause $C \in S_\infty$, C occurs in S_j for some $j \leq i$. Then C must be deletable in S_j, and by Corollary **??** every clause $C \in S_\infty$ must be deletable in every set S_k for $k \geq i$. But then, since the derivation is non-redundant, no `deduction` and `pruning` can occur after step i (since the adding of deletable clauses is forbidden). Since the initial clause set is finite, there can be only finitely many `deletion` steps. Thus, I must be finite. □

Proposition 7.3. Consider a non-redundant pruning derivation $(S_i)_{i\in I}$ from a finite clause set. Consider a pruning clause $C = n \not\approx \mathsf{s}^l(x)$ and let $i \in I$.

1. If $C \in S_i$, then any clause of a level at least l is deletable in S_i.
2. If S_∞ contains a pruning clause, then I is finite.

Proof:

1. Item 3 is a consequence of the fact that any normalized clause of level $k \geq l$ is of the form $n \not\approx \mathsf{s}^{k+1-\epsilon}(x) \vee D'$, where $\epsilon \in \{0, 1\}$, and such a clause is obviously deletable.
2. If S_∞ contains C, then C occurs in S_i for some $i \in I$ and by Item 3 all clauses of level greater than $l + 1$ are deletable in S_i. By Corollary **??**, these clauses are also deletable in every clause set S_j for $j \geq i$. Then, since $(S_j)_{j\in I}$ is non-redundant, the `deduction` and `pruning` steps after i cannot add any clause of a level greater than $l + 1$ by Item 3, and since there are only finitely many clauses of level at most l (up to deletion), Item 4 holds.

□

Theorem 7.1. If $(S_j)_{j\in I}$ is a fair, non-redundant pruning derivation from a finite and normalized clause set, then I is finite.

Proof:
Assume I is not finite, then S_∞ cannot contain $\square$, and by Proposition 7.1 (4), S_∞ cannot contain any pruning clause either. Thus, by Lemma 7.1, S_j is k-reducible for every $j \in I$. Since S_∞ is infinite by Proposition 4.4, S_∞ must contain clauses of arbitrary levels, and this implies that for all levels $i \in \mathbb{N}$, there exists a derivation step $\gamma(i)$ such that every clause generated after this step is of a level strictly greater than i.

Still by Proposition 4.4, there are at most $M = 2^{4|\Omega|+1}$ clauses of any given level, up to a renaming and the duplication of literals; there are therefore at most 2^M distinct normalized clause sets of the same level. We consider the integer $l = 2^M + k + 1$ and the set of clauses $S_{\gamma(l)}$ in the derivation, after which all generated clauses are of a level strictly greater than l. For all $m \leq l$, let $S'_m = \{C^{idx} \mid C \in S_{\gamma(l)}|_m\}$; by the pigeonhole principle, there exist $i, j \in \mathbb{N}$ such that $k < i \leq l$, $j \neq 0$ and $S'_i = S'_{i+j}$. This implies that $S_{\gamma(l)}|^\star_{i+j} = \mathit{shift}(S_{\gamma(l)}|^\star_i, j)$.

Since the derivation is fair by hypothesis and no clause of level i or $i+1$ can be generated after step $\gamma(l)$, the set $S_{\gamma(l)}$ is saturated up to level i. Therefore, the conditions of Theorem 6.1 hold, and the clause $n < i + j$ occurs in $\mathcal{P}r(S_{\gamma(l)})$. It is also generable from all subsequent clause sets in the derivation, thus by the fairness hypothesis, the `pruning` step is applied at some point in the derivation, hence S_∞ contains a pruning clause, which contradicts our initial assumption. $\square$

From the previous termination proof, it is easy to obtain an upper bound on the time complexity of the algorithm. At most $2^{2^{4|\Omega|+1}}$ clause sets of size at most $2^{4|\Omega|+1}$ can be generated, thus since the rules can be applied in polynomial time w.r.t. the size of the clause set, the algorithm is at most doubly exponential. This is worse than the complexity of the tableaux-based proof procedure described in [2] which is only simply exponential (if natural numbers are encoded in unary). Intuitively, this is due to the fact that the nodes in the tableaux are labeled by sets of literals, instead of sets of clauses.

8. Extension to first-order logic

Let Σ be a set of *function symbols* (distinct from 0 and `s`), let $\mathcal{V}_1$ be a set of *first-order variables* (disjoint from $\mathcal{V}$) and let Ω_1 be a set of *predicate symbols*. Each symbol f in $\Sigma \cup \Omega_1$ is mapped to a unique natural number, called the *arity* of f.

The set of *first-order terms* is built as usual on the signature Σ and on the set of variables $\mathcal{V}_1$. A *first-order clause* is simply a clause (as defined in Section 2), built on the set of propositional variables of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are first-order terms.

A *first-order substitution* is a mapping from $\mathcal{V}_1$ to the set of first-order terms. As usual, a substitution σ can be extended to any expression $\mathcal{E}$ and the image of $\mathcal{E}$ is denoted by $\mathcal{E}\sigma$. The notions of domain, ground substitution, unifier etc. are defined as in Section 2.

Note that the set of first-order terms and index terms are disjoint: for instance, $p_x(y)$ is a first-order clause but $p_x(x)$ or $p_x(0)$ are not.

We denote by $C\downarrow$ the set of clauses of the form $C\sigma$ where σ is a ground first-order substitution whose domain contains all the first-order variables in C (index variables are *not* instantiated). We also define

$S\downarrow \stackrel{\text{def}}{=} \bigcup_{C\in S} C\downarrow$. The notion of level and the notation $\mathit{shift}(S,i)$ extend straightforwardly to first-order clauses.

Proposition 8.1. For any first-order clause set S, $(S|_l)\downarrow = (S\downarrow)|_l$.

Proof:
This is immediate since if a clause C is of level l then obviously any instance $C\sigma$ of C (where σ is a first-order substitution) is also of level l: indeed, the level of a clause C does not depend on the propositional variables, but only on the equational part of C and on the indices occurring in C, which by definition are not affected by σ. □

Proposition 8.2. For any first-order clause set S and for any $i \in \mathbb{N}$, $(\mathit{shift}(S,i))\downarrow = \mathit{shift}((S\downarrow),i)$.

Proof:
The proof is straightforward, since the shift operation do not affect non-equational literals. □

The semantics of sets of first-order clauses are defined by interpreting a set S as the set of its ground instances. If Ω contains all ground atoms $p(t_1,\ldots,t_n)$, then $S\downarrow$ is simply a set of propositional clauses. Then for any interpretation I we define: $I \models S$ iff $I \models S\downarrow$.

As usual, we assume the selection function sel is extended to first-order clauses in such a way that if a literal $l\theta$ is selected in a clause $(l \vee C)\theta$ then l is also selected in $l \vee C$. The inference rules for first-order clauses are depicted in Figure 2. We denote by $\mathcal{R}es(S)$ the set of clauses C that are deducible from S (in one step). Note that the rules coincide with the ones of Figure 1 if the clauses are ground, which explains why we use the same notation $\mathcal{R}es(S)$.

$$\textit{Resolution}\quad \frac{p(\vec{t})_u \vee C \qquad \neg p(\vec{s})_v \vee D}{(C \vee D)\sigma\theta} \qquad (i)$$

$$\textit{Factorization}\quad \frac{p(\vec{t})_u \vee p(\vec{t})_v \vee C}{(p(\vec{s})_u \vee C)\sigma} \qquad \frac{\neg p(\vec{t})_u \vee \neg p(\vec{s})_v \vee C}{(\neg p(\vec{t})_u \vee C)\sigma\theta} \qquad (ii)$$

where the following conditions hold:

***(i)*:** $\sigma = \mathrm{mgu}(u,v)$, $\theta = \mathrm{mgu}(\vec{t},\vec{s})$ and $p_u\sigma\theta$ and $\neg p_v\sigma\theta$ are selected;

***(ii)*:** $\sigma = \mathrm{mgu}(u,v)$, $\theta = \mathrm{mgu}(\vec{t},\vec{s})$ and $p_u\sigma\theta$ or $\neg p_u\sigma\theta$ is selected.

Figure 2. The Resolution calculus (first-order clauses)

The notions of redundancy and level-redundancy can be extended to first-order clauses: C is *redundant* (resp. *level-redundant*) w.r.t. S iff every clause in $C\downarrow$ is redundant (resp. level-redundant) w.r.t. $S\downarrow$. The notion of saturation (up to a certain level) is the same as in the propositional case. Then $\mathcal{P}r(S)$ is defined as in Definition 6.6 and the notion of a pruning derivation can be extended as well. The following proposition relates as usual this "lifted" calculus to its ground version (including the pruning rule):

Lemma 8.1. Let S be a normalized set of first-order clauses.

- $\mathcal{R}es(S\downarrow) = (\mathcal{R}es(S))\downarrow$.
- If $C \in \mathcal{P}r(S)$ then $C \in \mathcal{P}r(S\downarrow)$.

Proof:
The first item is standard hence the proof is omitted. For the second item, consider a pruning clause of the form $n < i + j$ such that S, i and j satisfy the conditions of Definition 6.6. Then, by Condition 1 in Definition 6.6, S is saturated up to level i, i.e., $\mathcal{R}es(S|_{[\bot,i[}) \sqsubseteq_{\mathtt{1}} S$, thus $\mathcal{R}es(S|_{[\bot,i[})\downarrow \sqsubseteq_{\mathtt{1}} S\downarrow$. By the first item we get $\mathcal{R}es((S|_{[\bot,i[})\downarrow) \sqsubseteq_{\mathtt{1}} S\downarrow$, and by Proposition 8.1, $\mathcal{R}es((S\downarrow)|_{[\bot,i[}) \sqsubseteq_{\mathtt{1}} S\downarrow$, which means that $S\downarrow$ is saturated up to level i. Furthermore, by Condition 3 of Definition 6.6, $S|^{\star}_{i+j} = \mathit{shift}(S|^{\star}_{i}, j)$, thus $S|^{\star}_{i+j}\downarrow = \mathit{shift}(S|^{\star}_{i}, j)\downarrow$. By Proposition 8.2, we deduce $S|^{\star}_{i+j}\downarrow = \mathit{shift}(S|^{\star}_{i}\downarrow, j)$ and by Proposition 8.1, $S\downarrow|^{\star}_{i+j} = \mathit{shift}(S\downarrow|^{\star}_{i}, j)$. Thus $S\downarrow$, i and j satisfy the conditions of Definition 6.6, and $n < i + j \in \mathcal{P}r(S\downarrow)$. □

Proposition 8.3. For all sets of normalized first-order clauses S, if $S \vdash_p C$ then for every clause $D \in C\downarrow$, we have $S\downarrow \vdash_p D$. Furthermore, if $S\downarrow \vdash_p D$ then there exists a clause C such that $S \vdash_p C$ and $D \in C\downarrow$.

Proof:
This follows from Lemma 8.1 by an immediate induction on the length of the derivation. □

Theorem 8.1. For any set of normalized first-order clauses S, S is unsatisfiable iff there exists a (possibly infinite) unsatisfiable set of purely equational clauses S' such that for all $C \in S'$, $S \vdash_p C$.

Proof:
Assume S is unsatisfiable. Then so is $S\downarrow$ and by Theorem 3.1, there exists an unsatisfiable set of purely equational clauses S' such that for all $D \in S'$, $S\downarrow \vdash_p D$. By Proposition 8.3, for every clause $D \in S'$ there exists a clause C such that $S \vdash C$ and $D \in C\downarrow$. Since D is purely equational, so is C, hence $C\downarrow = \{C\}$ (since C contains no non equational literal, it contains no first-order variables). Thus $\forall C \in S', S \vdash_p C$.

Conversely, assume that there exists an unsatisfiable set of purely equational clauses S' such that for all $C \in S'$, $S \vdash_p C$. Then since $C\downarrow = \{C\}$ we have by Proposition 8.3: $S\downarrow \vdash_p C$. By Proposition 3.1 and Theorem 3.1 (stating the soundness of the resolution and pruning rules for propositional clauses), $S\downarrow$ is unsatisfiable. Hence S is also unsatisfiable. □

As already mentioned, Theorem 8.1 does *not* entail semi-decidability since S' may be infinite. Of course, termination cannot be ensured for first-order clauses (even if S is unsatisfiable) and Theorem 7.1 does not hold for first-order clauses (since there may be infinitely many clauses of a given level).

We now provide a simple example for which the calculus terminates.

Example 8.1. Let S be the following clause set:

1	$p_{\mathsf{s}(x)}(y, z, \text{cons}(y, v)) \vee \neg p_x(z, y, v)$
2	$p_0(y, z, \text{nil})$
3	$q_{\mathsf{s}(x)}(\text{cons}(u, v), v) \vee \neg q_x(y, v)$
4	$q_{\mathsf{s}(x)}(\text{cons}(u, \text{nil}), u) \vee \neg r_x$
5	$\text{even}_{\mathsf{s}(x)} \vee \text{even}_x$
6	$\neg\text{even}_{\mathsf{s}(x)} \vee \neg\text{even}_x$
7	even_0
8	$n \not\approx x \vee \neg\text{even}_x$
9	$n \not\approx x \vee \neg p_x(a, b, u) \vee \neg q_x(u, a)$
10	r_0

The symbols a, b are constant symbols, y, z, u, v are first-order variables; $p_x(y, z, u)$ holds if u is a list of length x of the form $y, z, y, z, y, \ldots$ and $q_x(u, v)$ holds if u is a list of length x whose last element is v. The atom r_x states that x is 0. Clauses 8 and 9 states that n is odd and that there is no list u satisfying both $p_n(a, b, u)$ and $q_n(u, a)$, which obviously contradicts the previous axioms. The calculus yields:

10	$n \not\approx 0$	(res,7, 8)
11	$n \not\approx \mathsf{s}(x) \vee \text{even}_x$	(res,5, 8)
12	$n \not\approx \mathsf{s}(x) \vee \neg p_x(b, a, v) \vee \neg q_{\mathsf{s}(x)}(\text{cons}(u, v), a)$	(res, 9, 1)
13	$n \not\approx \mathsf{s}(x) \vee \neg p_x(b, a, \text{nil}) \vee \neg r_x$	(res,12, 4)
14	$n \not\approx \mathsf{s}(x) \vee \neg r_x$	(res,13, 2)
15	$n \not\approx \mathsf{s}(0)$	(res,14, 10)
16	$n \not\approx \mathsf{s}(x) \vee \neg p_x(b, a, v) \vee \neg q_x(v, a)$	(res, 12, 3)
17	$n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee \neg\text{even}_x$	(res,11, 6)
18	$n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee \neg p_x(a, b, v) \vee \neg q_{\mathsf{s}(x)}(\text{cons}(u, v), a)$	(res, 16, 1)
19	$n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee \neg p_x(a, b, \text{nil}) \vee \neg r_x$	(res, 18,4)
20	$n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee \neg r_x$	(res, 19, 2)
21	$n \not\approx \mathsf{s}(\mathsf{s}(0))$	(res, 20, 10)
22	$n \not\approx \mathsf{s}(\mathsf{s}(x)) \vee \neg p_x(a, b, v) \vee \neg q_x(v, a)$	(res, 18, 3)
23	$n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(x))) \vee \text{even}_x$	(res,17, 5)
24	$n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(x))) \vee \neg p_x(b, a, v) \vee \neg q_x(\text{cons}(u, v), a)$	(res, 22, 1)
25	$n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(x))) \vee \neg p_x(b, a, \text{nil}) \vee \neg r_x$	(res, 24, 4)
26	$n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(x))) \vee \neg p_x(b, a, v) \vee \neg q_x(v, a)$	(res, 24, 3)
27	$n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(0))) \vee \neg r_x$	(res, 25, 2)
28	$n \not\approx \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$	(res, 27, 10)

At this point the pruning rule is applied (with $i = 2$ and $j = 2$), yielding the clause $n < 4$. With the clauses $10, 15, 21, 28$, this proves the unsatisfiability of the original clause set.

9. Conclusion

We have devised a resolution-based proof procedure capable of handling sets of propositional or first-order clauses indexed by natural numbers, which can be used to encode schemata of formulæ. In the propositional case, this procedure has been proven to be sound, refutationally complete and terminating. In the first-order case, the procedure is only sound (the satisfiability problem is not even semi-decidable).

Future work includes the implementation of the calculus and experimental comparison with the tableau-based prover described in [5] for propositional schemata. Although we have proven (see Section 7) that the complexity of the resolution-based procedure is exponentially greater than that of the tableaux-based procedure, this does not necessarily imply that it will be less efficient in practice. From a theoretical point of view, the extension to the equality case should also be considered (using the superposition calculus [9]).

A very natural direction of research is to extend the termination results of Section 7 to subclasses of first-order clauses. This may be done by restricting ourselves to syntactic subclasses for which the resolution calculus terminates (see for instance [12] for numerous examples of such classes). However, this is not as straightforward as one may think because termination is usually ensured thanks to some specific ordering restriction strategy. In general, there is no guarantee that these ordering restrictions will be compatible with the ones already considered in the present paper, which impose that the literals are ordered according to their indices. For instance, a seemingly natural idea is to extend the present work to schemata of ground clause sets with equality by using the superposition calculus instead of resolution. This would allow one to handle for instance clauses such as $a_i \approx b_i$ or $a_i \approx f(b_i)$, where a, b denotes indexed constant symbols. As is well-known, the superposition calculus always terminates on (non indexed) ground clauses. However, if indexed clauses are considered, termination *cannot be ensured*, due to the fact that a term indexed by $i+1$ must be always greater than a term indexed by i, even if the latter is actually less complex according to the usual ordering: for instance the set $\{p(a_i), a_{i+1} \approx f(a_i)\}$ entails an infinite number of distinct clauses, of the form $p(f^k(a_i))$ ($k \in \mathbb{N}$), obtained by repeatedly replacing a_{i+1} by $f(a_i)$. Note that these clauses are all of the same level. Thus the results in the present paper do *not* extend to ground clauses with equality. Actually the satisfiability problem turns out to be *undecidable* for schemata of ground clause sets with equality [7]. Similarly, our calculus does not terminate on the rather simple clause set $\{\neg p_x(f(y)) \vee p_{\mathsf{s}(x)}(y), n \not\approx x \vee \neg p_x(a)\}$, although it is monadic and obviously satisfiable, since the ordering conditions impose to resolve on the literal $p_{\mathsf{s}(x)}(y)$ (which in principle would be avoided by standard resolution strategies).

Another direction of research is to refine the loop detection rule in order to get rid of the saturation condition (which is difficult to enforce for sets of first-order clauses). This may be done by analyzing more precisely the search space in order to identify cycles in the proof tree. Rather than considering the clause set as a whole, one would then focus on the specific clauses that are relevant for proving the considered property. This line of research is currently under investigation.

References

[1] Althaus, E., Kruglov, E., Weidenbach, C.: Superposition Modulo Linear Arithmetic SUP(LA), *FroCoS 2009* (S. Ghilardi, R. Sebastiani, Eds.), 5749, Springer, 2009.

[2] Aravantinos, V., Caferra, R., Peltier, N.: A DPLL Proof Procedure for Propositional Iterated Schemata, *Workshop "Structures and Deduction 2009" (ESSLI)*, 2009.

[3] Aravantinos, V., Caferra, R., Peltier, N.: A Schemata Calculus For Propositional Logic, *TABLEAUX 09 (International Conference on Automated Reasoning with Analytic Tableaux and Related Methods)*, 5607, Springer, 2009.

[4] Aravantinos, V., Caferra, R., Peltier, N.: A Decidable Class of Nested Iterated Schemata, *IJCAR 2010 (International Joint Conference on Automated Reasoning)*, LNCS, Springer, 2010.

[5] Aravantinos, V., Caferra, R., Peltier, N.: RegSTAB: a SAT-Solver for Propositional Schemata, *IJCAR 2010 (International Joint Conference on Automated Reasoning)*, LNCS, Springer, 2010.

[6] Aravantinos, V., Caferra, R., Peltier, N.: Decidability and Undecidability Results for Propositional Schemata, *Journal of Artificial Intelligence Research*, **40**, 2011, 599–656.

[7] Aravantinos, V., Peltier, N.: Schemata of SMT problems, *TABLEAUX 11 (International Conference on Automated Reasoning with Analytic Tableaux and Related Methods)*, LNCS, Springer, 2011.

[8] Baaz, M., Hetzl, S., Leitsch, A., Richter, C., Spohr, H.: CERES: An analysis of Fürstenberg's proof of the infinity of primes, *Theor. Comput. Sci.*, **403**(2-3), 2008, 160–175.

[9] Bachmair, L., Ganzinger, H.: Rewrite-based Equational Theorem Proving with Selection and Simplification, *Journal of Logic and Computation*, **3**(4), 1994, 217–247.

[10] Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational Theorem Proving for Hierachic First-Order Theories, *Appl. Algebra Eng. Commun. Comput.*, **5**, 1994, 193–212.

[11] Comon, H., Lescanne, P.: Equational Problems and Disunification, *Journal of Symbolic Computation*, **7**, 1989, 371–475.

[12] Fermüller, C., Leitsch, A., Tammet, T., Zamov, N.: *Resolution Methods for the Decision Problem*, LNAI 679, Springer, 1993.

[13] Gupta, A., Fisher, A. L.: Parametric Circuit Representation Using Inductive Boolean Functions, *CAV* (C. Courcoubetis, Ed.), 697, Springer, 1993, ISBN 3-540-56922-7.

[14] Horbach, M., Weidenbach, C.: Deciding the Inductive Validity of FOR ALL THERE EXISTS * Queries, *CSL* (E. Grädel, R. Kahle, Eds.), 5771, Springer, 2009, ISBN 978-3-642-04026-9.

[15] Horbach, M., Weidenbach, C.: Superposition for fixed domains, *ACM Trans. Comput. Logic*, **11**(4), 2010, 1–35, ISSN 1529-3785.

[16] Korovin, K., Voronkov, A.: Integrating Linear Arithmetic into Superposition Calculus, *CSL 2007* (J. Duparc, T. A. Henzinger, Eds.), 4646, Springer, 2007.

[17] Leitsch, A.: *The resolution calculus*, Springer. Texts in Theoretical Computer Science, 1997.