A Paramodulation-based calculus for refuting schemata of clause sets defined by rewrite rules

N. Pelletier
University of Grenoble (CNRS/LIG)
January 11, 2013

Abstract
We devise a calculus based on the resolution and paramodulation rules and operating on schemata of formulae. These schemata are defined inductively, using convergent rewrite systems encoding primitive recursive definitions. The main original feature of this calculus is that the rules operate on formulae or terms occurring at arbitrary deep positions inside the considered schemata, thus affecting the corresponding rewrite system. Each inference step in the new calculus corresponds to several applications of the usual resolution or paramodulation rules over the considered instances. The calculus has been implemented in the proof editor Shred (available on the web). As an example of application we provide a formal refutation of a schema of clause sets generated by applying the CERES cut-elimination method on Fürtsenberg’s proof of the infinite of prime numbers [9].

1 Introduction

A schema of terms or formulae is an infinite family of standard (first-order) terms or formulae, parameterized by natural numbers, and defined by recursion on these numbers. For instance, the expression \( \phi_n : \bigvee_{i=1}^n p(a_i) \) is a schema of formulae, and \( \tau_n : f(a_n(f(a_{n-1}, \ldots, f(a_1, c), \ldots))) \) is a schema of terms. More formally, these two expressions can be defined by the following systems of rewrite rules:

\[
\begin{align*}
\phi_0 & \rightarrow \text{false} \\
\phi_{n+1} & \rightarrow \phi_n \lor p(a_{n+1})
\end{align*}
\]

and

\[
\begin{align*}
\tau_0 & \rightarrow c \\
\tau_{n+1} & \rightarrow f(a_{n+1}, \tau_n)
\end{align*}
\]

Such constructions are ubiquitous in mathematical textbooks, but they are usually considered only at the meta-level. Note that the parameter \( n \) is to be
interpreted on the set of natural numbers and can be quantified either universally or existentially. For instance, in order to prove that the following implication

\[
(q(a_0) \land \neg q(a_{n+1}) \land \bigwedge_{i=1}^{n} q(a_i) \Rightarrow (q(a_{i+1}) \lor p(a_i))) \Rightarrow \bigvee_{i=1}^{n} p(a_i)
\]

holds, for every \(n \in \mathbb{N}\), one has to prove that the formula

\[
\exists n \in \mathbb{N}, q(a_0) \land \neg q(a_{n+1}) \land \bigwedge_{i=1}^{n} q(a_i) \Rightarrow (q(a_{i+1}) \lor p(a_i)) \land \neg \bigvee_{i=1}^{n} p(a_i)
\]

is unsatisfiable. The variable \(n\) can be replaced by a constant symbol by skolemisation, but this constant must be interpreted as an element of \(\mathbb{N}\). Obviously, mathematical induction is required to prove this assertion (it does not hold if \(n\) is interpreted on an arbitrary domain).

There are several ways of handling such expressions. The standard approach consists in viewing the rewrite rules encoding the definition of the schemata as first-order formulae (either equalities or equivalences) which can be added as axioms to the considered formula, and to use the standard proof procedures for reasoning with first-order formulae, enriched by explicit induction schemes to take into account the inductive structure of \(\mathbb{N}\). In this approach, axioms and schema definitions are considered in a uniform way, which has an important drawback: while the initial statement is a direct encoding of the considered schema, this correspondence will not be preserved by logical inferences, that will completely destroy the “schematic structure” of the problem. Therefore, the obtained derivations will be difficult to understand and follow for a human user, thus making the calculus ill-suited for interactive theorem-proving. For instance, from two formulae \(f(x,c) \simeq d\) and \(p(\tau_n)\), where \(\tau_n\) is the schema of terms defined above, one could expect to derive the formula \(f(a_n(f(a_{n-1},\ldots,f(a_2,d)\ldots)))\), which can be encoded by a new term schema \(p(\tau'_{n-1})\) with the rules:

\[
\begin{align*}
\tau'_0 & \rightarrow d \\
\tau'_{n+1} & \rightarrow f(a_{n+2},\tau'_n)
\end{align*}
\]

Instead, usual proof procedures would simply (at best) derive the equation \(\tau_1 \simeq d\): the previous schema cannot be explicitly generated. Furthermore, if the set of hypotheses contains, e.g., the equation \(\tau'_n \simeq a\), then the user would expect to be able to derive the formula \(p(a)\) in two steps from \(p(\tau_n)\) and \(f(a_1,c) \simeq d\) (since \(f(a_1,c) \simeq d\) entails \(\tau_n \simeq \tau'_n\)); but if the schema definitions are encoded as first-order formula this formula can only be derived by a tedious induction (involving an auxiliary induction invariant).

In this paper, we propose to use another and much more direct approach, which consists in devising inference rules operating directly on schematic objects. This calculus has the advantage that the obtained derivations are much closer to those constructed by humans and thus much more readable and more natural. For instance, from the clauses \(\bigvee_{i=1}^{n} p(a_i)\) and \(\neg p(x) \lor q(x)\) (where \(x\) is a variable),
the calculus is able to derive the clause $\bigvee_{i=1}^{n} q(a_i)$ in one step. In contrast, using an explicit formalisation of schemata yields a much more tedious and complex derivation. Firstly one would have to write explicitly the induction lemmata, here (for instance) $\psi(n, m) \equiv \bigvee_{i=1}^{m} q(a_i) \lor \bigvee_{i=m+1}^{n} p(a_i)$, then one has to prove that $\psi(n, m)$ holds for every $m \in [0, n]$, by induction on $m$, i.e. that the formula $\phi \Rightarrow \psi(n, 0) \land (\forall m \in [0, n-1] \psi(n, m) \Rightarrow \psi(n, m + 1))$ holds.

The rest of the paper is structured as follows. Section 2 defines the syntax and semantics of the logic of schemata. An unusual feature of this logic is that terms and formulae are handled in a uniform way; this is convenient in our context since the unfolding of schemata definitions does not preserve conjunctive normal forms. Section 3 tackles the problem of the replacement of subterms inside a schema. This problem is much more complex than in the standard case, since the replacement can affect the rewrite system describing the schemata and since the unfolding of these definitions can create an infinite number of distinct terms, which have all the same shape, and therefore can be replaced in parallel. Section 4 contains the formal definition of the inference rules. In Section 5, we prove that these rules are sound, and in Section 6, completeness issues are investigated. The unsatisfiability problem is not semi-decidable, but we devise some syntactic conditions ensuring completeness. In Section 7 a publicly available implementation of the calculus is briefly described and an example of application is provided. Finally, Section 8 briefly concludes the paper and gives some lines of future work.

Related work

We refer to [4] for a detailed comparison between schemata and usual logical languages. Logical formalisms to reason on schemata of formulae have been first introduced in [1, 3]. Unlike the present work, these logics are restricted to propositional schemata with a unique parameter. Furthermore, the devised proof procedures (namely tableaux and Davis-Putnam-Logemann-Loveland procedures enriched by delayed instantiation schemes and loop detection mechanisms) are completely different from the calculus described in the present paper. In [5] (see also [21]), a resolution-based calculus is introduced for first-order schemata. This procedure shares some common points with the one presented here, namely the use of the resolution method as a base calculus and the handling of arithmetic parameters, but it is based on very different ideas. In [5], schemata are handled in an implicit way: recursive rules are encoded as axioms and the inference rules operate on these axioms rather than on the schemata themselves. No explicit induction scheme is considered, and the method relies instead on a loop detection mechanism that is able to generate implications of the form $S[n] \Rightarrow n > 0 \land S[n - 1]$, which by descente infinie entail that $S$ is unsatisfiable. These features make the extraction of resolution proofs difficult, first because the inferences performed by the calculus do not necessarily correspond to inferences at the first-order level, and second because the induction invariants are not explicit. In contrast, the construction of the derivations is straightforward in our approach (these derivations are defined as usual by recursion on
the parameter). On the other hand, the procedure in [5] uses a more efficient calculus, with ordering restrictions to prune the search space. Consequently, the method in [5] seems better suited to automated proof search, whereas the calculus defined in the present paper is better adapted to interactive theorem proving. It should also be mentioned that the method in [5] is complete only for propositional schemata with a unique parameter. The completeness result given in the present paper is strictly more general. It is not comparable with the results in [21].

2 Preliminary definitions

Syntax

We assume some familiarity with the usual notions in logic and resolution-or paramodulation-based theorem-proving (we refer to, e.g., [22, 26] for more details).

Let $S$ be a set of sort symbols, containing in particular the symbols bool (booleans) and nat (natural numbers). The set of terms (or schemata) is built inductively using a set of function symbols $\Sigma$ and a set of variables $X$ (with $X \cap \Sigma = \emptyset$). We assume that each variable in $X$ has a unique sort in $S$ and that every function symbol $f$ in $\Sigma$ has a unique profile of the form $s_1, \ldots, s_n \to s$, where $s_1, \ldots, s_n, s \in S$ (with possibly $n = 0$); $s$ is the range of $f$ and $s_1, \ldots, s_n$ is its domain. Variables of sort nat will be denoted by $u, v, w$ and those of a sort different from nat will be denoted by $x, y, z$ (possibly with indices or primes).

The function symbols whose profile is of the form $s_1, \ldots, s_n \to \text{bool}$ are the predicate symbols. We assume that $\Sigma$ contains in particular the usual symbols $0, 1 : \text{nat}, + : \text{nat} \to \text{nat}$ and $< : \text{nat}, \text{nat} \to \text{bool}$ of Presburger arithmetic, the usual logical constant symbols and connectives $\text{true}, \text{false} : \text{bool}, \land, \lor, \Rightarrow : \text{bool}, \text{bool} \to \text{bool}$ and an equality predicate $\approx^s$ for each sort $s \in S$. For simplicity the exponent will be omitted, and $\approx^s$ will be written $\approx$ (the domain will be clear from the context). The function symbols of a range distinct from bool will be denoted by $f, g, h, \ldots$ and predicate symbols are denoted by $p, q, r, \ldots$. The constant symbols of sort nat and distinct from 0 are the parameters; they will be denoted by $n$ or $m$.

Note that, in our setting, formulae and terms are handled in a uniform way: all formulae are taken as terms of sort bool (all variables are implicitly universally quantified). We do not assume that these formulae are in clausal form, because, as we shall see, this property is not necessarily preserved by the rewrite rules specifying the interpretation of the defined symbols. We assume that the symbol $+$ is the only non-constant symbol of range nat. As usual, the term $1 + \ldots + 1 (k \text{ times})$ is denoted by $k$.

A substitution $\sigma$ is a function mapping each variable to a term of the same sort. For any term $t$, $t_\sigma$ denotes the term obtained from $t$ by replacing each occurrence of a variable $x$ by $\sigma(x)$. If $\vec{x} = (x_1, \ldots, x_k)$ is a vector of variables and $\vec{t} = (t_1, \ldots, t_k)$ is a vector of terms, such that for every $i \in [1, k]$, $t_i$ is of the
same sort as \( x_i \), we denote by \( \{ \bar{x} \mapsto \bar{t} \} \) the substitution of domain \( x_1, \ldots, x_k \) such that \( \forall i \in [1, k], \sigma(x_i) = t_i \). A substitution \( \sigma \) is a unifier of two terms \( t \) and \( s \) if \( t \sigma = s \sigma \) (up to the usual arithmetic properties of the symbols \( \mathit{0}, \mathit{1} \) and \( + \)). Note that unification modulo Presburger arithmetic is decidable \([7]\) (however the most general unifier is not unique).

Let \( \mathcal{D} \subseteq \Sigma \) be a set of defined symbols (written with a hat to distinguish them from ordinary symbols). We assume that the profile of every symbol \( \hat{f} \in \mathcal{D} \) is of the form \( \mathtt{nat}, s_1, \ldots, s_n \rightarrow s \), where \( s_1, \ldots, s_n, s \in \mathcal{S} \). A term of the form \( \hat{f}(\bar{t}) \), with \( \hat{f} \in \mathcal{D} \) is a defined term. The symbols in \( \mathcal{D} \) are to be used to construct sequences of terms or formulae, parameterized by natural numbers. For instance, the formula \( \bigvee_{i=0}^n p_i \) will be denoted by a defined term \( \hat{f}(n) \), where the interpretation of \( \hat{f} \) is specified by the following rewrite rules:

\[
\begin{align*}
\hat{f}(0) & \rightarrow \ \text{false} \\
\hat{f}(u + 1) & \rightarrow \ \hat{f}(u) \lor p_{u+1}
\end{align*}
\]

We therefore assume that a set of rewrite rules \( \mathcal{R} \) is given, so that any ground defined term can be reduced to a term not containing defined symbols (the reader can refer to, e.g., \([8]\) for all the basic definitions concerning rewrite systems).

More precisely, let \( \preceq \) be a pre-order on defined symbols. We write \( \hat{f} \sim \hat{g} \) if \( \hat{f} \preceq \hat{g} \) and \( \hat{g} \preceq \hat{f} \) and \( \hat{f} \prec \hat{g} \) if \( \hat{f} \preceq \hat{g} \) and \( \hat{g} \npreceq \hat{f} \). We assume that each symbol \( \hat{f} \in \mathcal{D} \) of profile \( s_1, \ldots, s_n \rightarrow s \) is mapped to a rewrite system \( \mathcal{R}_{\hat{f}} \) containing exactly two rules of the form:

\[
\hat{f}(0, \bar{x}) \rightarrow B
\]

(the base rule of \( \hat{f} \)) and

\[
\hat{f}(u + 1, \bar{x}) \rightarrow I
\]

(the inductive rule of \( \hat{f} \)), satisfying the following properties:

- \( B \) and \( I \) are terms of sort \( s \).
- \( u, \bar{x} \) is a vector of pairwise distinct variables of sorts \( s_1, \ldots, s_n \).
- For every defined term \( \hat{g}(s, \bar{t}) \) occurring in \( B \), we have \( \hat{g} \prec \hat{f} \).
- For every defined term \( \hat{g}(s, \bar{t}) \) occurring in \( I \) we have either \( \hat{g} \sim \hat{f} \) and \( s = u \), or \( \hat{g} \prec \hat{f} \).

Let \( \mathcal{R} = \bigcup_{\hat{f} \in \mathcal{D}} \mathcal{R}_{\hat{f}} \).

**Proposition 1** \( \mathcal{R} \) is convergent.

**Proof.** It is clear that \( \mathcal{R} \) is orthogonal, hence it suffices to prove that it is terminating. We consider the ordering \( \prec \) inductively defined as follows:

- \( f(t_1, \ldots, t_n) \prec f(s_1, \ldots, s_n) \) if \( \{s_1, \ldots, s_n\} \) is greater than \( \{t_1, \ldots, t_n\} \) according to the multiset extension of \( \prec \).
\[ t \prec f(s_1, \ldots, s_n) \text{ if } t \preceq s_i, \text{ for some } i \in [1, n]. \]

\[ f(t_1, \ldots, t_n) \prec g(s_1, \ldots, s_m) \text{ if for every } i \in [1, n], t_i \prec g(s_i, \ldots, s_m) \text{ and either } g \in D \text{ and } f \notin D, \text{ or } f, g \in D \text{ and either } f \prec g \text{ or } (f \sim g \text{ and } s_1 = t_1 + k, \text{ for some } k \in [1, \infty[). \]

It is straightforward to verify that for every rule \( t \rightarrow s \) and for every substitution \( \sigma \), we have \( s \sigma \prec t \sigma \). Furthermore, \( \prec \) can be viewed as a recursive path ordering (on a modified signature, where the defined terms \( f(k, \vec{t}) \) are written of the form \( (f, k)(\vec{t}) \), with the precedence \( (f, k) > (g, l) \) iff \( f \succ g \) or \( (f \sim g \text{ and } k > l) \)). Therefore, it is a reduction ordering (see for instance [14]), hence \( \mathcal{R} \) must be terminating.

The normal form of any expression \( e \) by \( \mathcal{R} \) is denoted by \( e \downarrow \mathcal{R} \). Note that the rules in \( \mathcal{R} \) are not taken into account when computing unifiers. Unification modulo \( \mathcal{R} \) is clearly not decidable in general. Indeed, although only linear arithmetic expressions are allowed in term schemata, all diophantine equations can be encoded as unification problems modulo \( \mathcal{R} \) as shown by the following proposition.

**Proposition 2** The solvability of diophantine equations can be reduced to a unification problem between term schemata.

**Proof.** It suffices to consider the following rewrite rules (with \( s \prec \hat{g}_+ \prec \hat{g}_\times \)):

\[
\begin{align*}
\hat{s}(0) & \rightarrow a \\
\hat{s}(u + 1) & \rightarrow f(\hat{s}(u)) \\
\hat{g}_+(0, v) & \rightarrow v \\
\hat{g}_+(u + 1, v) & \rightarrow f(\hat{g}_+(u, v)) \\
\hat{g}_\times(0) & \rightarrow a \\
\hat{g}_\times(u + 1, v) & \rightarrow \hat{g}_+(v, \hat{g}_\times(u, v))
\end{align*}
\]

It is clear that the defined terms \( \hat{s}(u), \hat{g}_+(u, v) \) and \( \hat{g}_\times(u, v) \) encode the terms \( f^u(a), f^{u+v} \) and \( f^{u \times v}(a) \), respectively. Using these function symbols, encoding diophantine equations is a straightforward task. For instance, an equation such as \( u \times v = u + v + 1 \) can be encoded as the unification problem \( \hat{g}_\times(u, v) \equiv f(\hat{g}_+(u, v)) \).

A literal is a term of sort \( \text{bool} \) that is of the form \( p(t_1, \ldots, t_n) \) or \( \neg p(t_1, \ldots, t_n) \) where \( p \) is distinct from \( \lor, \land, \rightarrow \) and \( \neg \) (\( p \) may be \( \simeq \)). A literal of the form \( \neg(t \simeq s) \) is written \( t \not\simeq s \). If \( L \) is a literal, then \( L^c \) denotes the complement of \( L \). A clause is a finite disjunction of literals. For the sake of clarity, we shall assume that the logical connectives \( \lor, \land, \rightarrow, \neg \) are the only non-defined symbols of domain \( \text{bool} \) (i.e. non interpreted functions operating on booleans are not allowed) and that there is no variables of sort \( \text{bool} \).
Fixed arguments

Before we turn to the definition of the semantics of the language, we introduce the useful notion of a fixed argument. An argument index is fixed for a defined symbol \( \hat{f} \) if its value does not change during the recursive calls to \( \hat{f} \). More formally:

**Definition 3** Let \( \hat{f} \) be a defined symbol of arity \( k+1 \). An index \( i \in [1,k] \) is a fixed argument of \( \hat{f} \) if for every term of the form \( \hat{f}(u,t_1,\ldots,t_k) \) occurring in the right-hand side of the inductive rule \( \hat{f}(u+1,x_1,\ldots,x_k) \to I \) of \( \hat{f} \), we have \( t_i = x_i \). The set of fixed arguments of \( \hat{f} \) is denoted by \( \text{F}(\hat{f}) \).

**Example 4** Assume that \( \hat{f} \) is defined by the rules: \( \hat{f}(0,x,y) \to a \) and \( \hat{f}(u+1,x,y) \to g(y,\hat{f}(u,x,h(y))) \). Then \( \text{F}(\hat{f}) = \{1\} \) is the set of fixed arguments of \( \hat{f} \). Intuitively, this means that at each recursive call to \( \hat{f} \) during the evaluation of a defined term of the form \( \hat{f}(m,t,s) \), the value of the second argument of \( \hat{f} \) will be identical to the initial one (namely \( t \)), whereas the values of the first and third arguments will change (they will be \( m, m-1, \ldots, 0 \) and \( s, h(s), h(h(s)), \ldots, h^m(s) \) respectively).

For every vector \((x_1,\ldots,x_n)\) and for every \( I \subseteq [1,n] \), \((x_{i_1},\ldots,x_{i_k})\) denotes the subvector \((x_{i_1},\ldots,x_{i_k})\) where \( I = \{i_1,\ldots,i_k\} \) and \( i_1 < \ldots < i_k \). In particular, \( \overline{t}_{\text{F}(\hat{f})} \) denotes the subvector of the arguments of \( \hat{f} \) that do not change during the evaluation of a term of the form \( \hat{f}(n,\overline{t}) \) (with \( n \in \mathbb{N} \)).

Semantics

**Definition 5** An interpretation \( I \) is a congruence on ground terms (written \( \equiv_I \)). We assume that:

1. The terms of sort \( \text{nat} \) are interpreted as natural numbers: for every ground term \( t \) of sort \( \text{nat} \), there exists \( k \in \mathbb{N} \) such that \( t \equiv_I k \). Furthermore, 0, 1, + and < are interpreted as usual.

2. The terms of sort \( \text{bool} \) are interpreted as booleans, and \( \text{true}, \text{false}, \lor, \land, \Rightarrow, \neg, \simeq \) have their usual meanings.

3. For every term \( t, t \equiv_I t_{\downarrow_R} \).

If \( \phi \) is a term of sort \( \text{bool} \), we write \( I \models \phi \) (\( I \) is a model of \( \phi \)) if for every ground substitution \( \sigma \) of the variables in \( \phi \), \( \phi \sigma \) is equivalent to \text{true} in \( I \).

**Remark 6** The first property above is the most important one: without it, the considered logic would be merely equivalent to first-order logic (the rewrite rules in \( R \) can be viewed as equations or equivalences). Fixing the interpretation of the parameters to elements on \( \mathbb{N} \) makes the logic non-semi-decidable.
Formulae (i.e., terms of sort `bool`) containing parameters and defined symbols are to be considered as schemata of first-order formulae. If the value of every parameter is fixed and if the variables of sort `nat` are replaced by ground terms then the formula at hand can be reduced to a first-order formula by applying the rewrite rules in \( R \). More formally, a nat-valuation \( \eta \) is a function mapping each parameter to a natural number. For every expression \( e \), \( e_{\eta} \) denotes the formula obtained from \( e \) by replacing every parameter \( n \) by \( \eta(n) \). A formula of the form \( \eta(e) \downarrow_R \) is a nat-evaluation of \( e \). The notions of a nat-valuation and of a nat-evaluation should not be confused with the similar notions of a substitution and of an instance (in which variables are replaced instead of parameters).

**Example 7** Consider the formula

\[
\hat{f}(n,a) \land (v + u \simeq n \Rightarrow \neg p(u,\hat{g}(v,a)))
\]

with the rules:

\[
\begin{align*}
\hat{f}(0,x) & \rightarrow \text{false} \\
\hat{f}(u + 1,x) & \rightarrow p(u + 1,a) \lor \hat{f}(u,f(x)) \\
\hat{g}(0,x) & \rightarrow x \\
\hat{g}(u + 1,x) & \rightarrow f(\hat{g}(u,x))
\end{align*}
\]

This formula encodes the schema

\[
\bigvee_{i=1}^{n} p(i, f(i, f^{n-i}(a))) \land \bigwedge_{i=0}^{n} \neg p(i, f(i, f^{n-i}(a))).
\]

Note that the iterated conjunction could also have been represented by mean of a defined symbol instead of using universal arithmetic variables \( u \) and \( v \).

### 3 Replacement

The first step toward the definition of a paramodulation-based calculus operating on schemata is to extend the notions of subterms and positions to schemata of terms. This definition must take into account not only the subterms occurring inside the considered schema of terms \( t \), but also those occurring in terms obtained from \( t \) by unfolding the defined symbols it contains. For instance, if \( \hat{f} \) is associated with the rules \( \hat{f}(0) \rightarrow a \) and \( \hat{f}(u + 1) \rightarrow g(b, \hat{f}(u)) \), then \( a \) and \( b \) both occur in the unfolded form of \( \hat{f}(n) \), although they do not syntactically occur in \( \hat{f}(m) \). Note that the symbol \( b \) actually occurs several times (possibly 0) in the unfolded form of \( \hat{f}(n) \), whereas \( a \) occurs exactly once (regardless of the value of \( n \)). We shall use two special symbols \( \beta \) and \( \iota \) to specify that a term appears in the base term or inductive term of a given defined symbol. More precisely:

**Definition 8** A generalized position is a finite sequence of elements of \( \mathbb{N} \cup \{ \beta, \iota \} \).

The symbol \( \varepsilon \) denotes the empty position and \( p.q \) is the concatenation of \( p \) and \( q \). A position is standard if it contains no occurrence of \( \iota \) and \( \beta \).

The terms occurring at a position \( \beta \) (resp. \( \iota \)) in a term \( \hat{f}(s,\bar{t}) \) are terms occurring in the right-hand side of the base rule (resp. of the inductive rule) of \( \hat{f} \). In general, such terms will contain variables from the rewrite system \( R \). If these variables correspond to fixed arguments of \( \hat{f} \), it is clear that they can be
replaced by the corresponding argument in the initial term \( \hat{f}(s, t) \). Otherwise, the variables will be instantiated in a different way at each unfolding step, thus they should not be replaced. Consider for instance the rewrite system of Example 4 and the term \( \hat{f}(n, a, b) \). The right-hand side of the inductive rule of \( \hat{f} \) contains two terms \( x \) and \( h(y) \). Since \( x \) corresponds to a fixed argument of \( \hat{f} \), it can be instantiated to \( a \), since its value will not change during recursive calls. However, \( y \) must be left unspecified, since its value depends on the iteration rank. These informal remarks yield the following technical definition, extending the usual notion of a subterm.

**Definition 9** Let \( t \) and \( s \) be two terms and let \( p \) be a position. The term \( t \) is the subterm occurring at position \( p \) in \( s \) (written \( s = t|_p \)) iff one of the following conditions holds:

- \( s = t \) and \( p = \varepsilon \).
- \( s = f(s_1, \ldots, s_n) \), \( t = s_i|_q \) and \( p = i.q \).
- \( s = \hat{f}(a, \vec{s}) \), \( p = \beta.q \), \( \mathcal{R} \) contains a rule \( \hat{f}(0, \vec{x}) \rightarrow B \), such that \( t = B\sigma|_q \) and \( \sigma = \{\vec{x}_{F(\hat{f})} \mapsto \vec{s}_{F(\hat{f})}\} \).
- \( s = \hat{f}(a, \vec{s}) \), \( p = \iota.q \), \( \mathcal{R} \) contains a rule \( \hat{f}(u + 1, \vec{x}) \rightarrow I \), \( t = I\sigma|_q \) where \( \sigma = \{\vec{x}_{F(\hat{f})} \mapsto \vec{s}_{F(\hat{f})}\} \) and for all prefixes \( r \) of \( q \), \( I|_r \) is of a defined head that is strictly lower than \( \hat{f} \).

We denote by \( \text{pos}(t) \) the set of positions \( p \) such that there exists a term at position \( p \) in \( t \).

As explained before, we do not take \( \sigma = \{\vec{x} \mapsto \vec{s}\} \), because the arguments that do not occur in \( F(\hat{f}) \) can change during the evaluation process. This implies that a term \( s \) occurring in \( t \) possibly contains some extra-variables, not occurring in \( t \). The value of these variables is unknown and may actually vary during the evaluation process. As we shall see in Section 4, additional conditions will be added on the unifiers when applying inference rules on terms containing such extra-variables. Note that the condition on the prefixes of \( q \) is essential to ensure that the definition is well-founded (if this condition is omitted then there could be infinitely many positions in a term, which would make the inference rules infinitary).

**Example 10** Let \( t = \hat{f}(n, b, c) \), with the rules: \( \hat{f}(0, x, y) \rightarrow a \) and \( \hat{f}(n + 1, x, y) \rightarrow g(h(x, y), \hat{f}(u, i(x, y))) \). We have \( t|_\beta = a \), \( t|_\iota = g(h(x, c), \hat{f}(u, x, c)) \), \( t|_{i, 1} = h(x, c) \). Note that \( y \) is replaced by \( c \) because it corresponds to a fixed argument of \( \hat{f} \), whereas \( x \) and \( u \) are left unspecified since their values depend on the iteration rank.

The next definition formalizes the replacement of a subterm inside a schema. The essential difference between the usual definition is that this operation possibly affects the rewrite system \( \mathcal{R} \). For instance, consider the term \( \hat{f}(n) \) associated
with the base rule \( \hat{f}(0) \to a \). To replace \( a \) by a new term \( b \) in \( \hat{f}(m) \) we have to change the base rule of \( \hat{f} \); this is done by introducing a new symbol \( \hat{f}^\prime \), whose inductive rule is identical to that of \( \hat{f} \) (except that \( \hat{f} \) is replaced by \( \hat{f}^\prime \)) and whose base term is replaced by \( \hat{f}^\prime(0) \to b \).

**Definition 11** Let \( t, s \) be terms and let \( p \) be a position in \( s \). We let \( s[t]_p \) denote the term defined as follows:

- If \( p = \varepsilon \) then \( s[t]_p \equiv t \).
- If \( s = f(s_1, \ldots, s_n) \) and \( p = i.q \) then:
  \[
  s[t]_p \equiv f(s_1, \ldots, s_{i-1}, s_i[t]_q, s_{i+1}, \ldots, s_n).
  \]
- Assume that \( s \) is of the form \( \hat{f}(a, \vec{s}) \) where \( \hat{f} \) is defined by the rules \( \hat{f}(0, \vec{x}) \to B \) and \( \hat{f}(u+1, \vec{x}) \to I \). Let \( \vec{y} \) be the vector of variables occurring in \( t \) but not in \( \vec{x} \). Then:
  - If \( p = \beta.q \) then \( s[t]_p \equiv \hat{f}^\prime(a, \vec{s}, \vec{y}) \), where \( \hat{f}^\prime \) is a new defined symbol associated with the rules: \( \hat{f}^\prime(0, \vec{x}, \vec{y}) \to B[t]_q \) and \( \hat{f}^\prime(u+1, \vec{x}, \vec{y}) \to I' \), where \( I' \) is obtained from \( I \) by replacing any term of the form \( \hat{f}(u, \vec{i}) \) by \( \hat{f}(u, \vec{i}, \vec{y}) \).
  - If \( \alpha = \iota.r.q \) then \( s[t]_p \equiv \hat{f}^\prime(a, \vec{s}, \vec{y}) \), where \( \hat{f}^\prime \) denotes a new defined symbol defined by the rules: \( \hat{f}^\prime(0, \vec{x}, \vec{y}) \to B \) and \( \hat{f}^\prime(u+1, \vec{x}, \vec{y}) \to I'[u]_{r.q} \), where \( I' \) is obtained from \( I \) by replacing any term of the form \( \hat{f}(u, \vec{i}) \) by \( \hat{f}^\prime(u, \vec{i}, \vec{y}) \).

In order to ensure that the resulting rewrite system fulfills the required conditions, the ordering \( \prec \) is extended to \( \hat{f}^\prime \) in such a way that \( \hat{f}^\prime \) is strictly greater than all defined symbols distinct from \( \hat{f}^\prime \) occurring in \( I' \) or \( B \).

**Example 12** We consider the term \( t \) of Example 10. It is easy to check that \( t[g(x, z)]_{1,1} = \hat{f}^\prime(n, b, c, y) \), where \( \hat{f}^\prime \) is associated with the rules \( \hat{f}^\prime(0, x, y, z) \to a \) and \( \hat{f}^\prime(u + 1, x, y, z) \to g(g(x, z), \hat{f}^\prime(n, i(x), y, z)) \). Note that the variable \( x \) is instantiated differently at each recursive call to \( \hat{f} \). Intuitively, \( t \) denotes the schema \( g(h(b, c), g(h(i(b), c), \ldots, g(h(i^{m-1}(b), c), a), \ldots)) \) and \( t[g(x, z)]_{1,1} \) is \( g(g(b, z), g(g(i(b), z), \ldots, g(h(i^{m-1}(b), z), a), \ldots)) \).

Note that the previous definition is an inductive one: it allows for the replacement of terms occurring at arbitrary deep recursion levels. If \( p \) is standard then the definitions of \( t[p] \) and \( t[s]_p \) above coincide with the usual ones.

We finally introduce the notion of a positive position. Intuitively, a position \( p \) is positive in a term \( t \) if no negation symbol \( \neg \) occurs along \( p \) in \( t \). For instance if \( t = \neg p(a) \lor q(b) \), then the positions 1 and 2 are both positive in \( t \) but 1.1 is not. Again, this definition is meant modulo unfolding: we have to take into account the negation symbols occurring inside the rewrite system. For instance, if \( t =
\( \hat{p}(n, x, y) \), and if the inductive rule of \( \hat{p} \) is \( \hat{p}(u + 1, x, y) \rightarrow q(x) \lor \neg(\hat{p}(u) \land r(y)) \), then the position 2 is positive in \( t \) (because the second argument of \( \hat{p} \), namely \( x \), does not occur in the scope of a negation in the inductive term), but 3 and \( \iota \) are not (because \( y \) and \( \hat{p} \) both occur in the scope of a negation symbol).

**Definition 13** A position \( p \) is positive in a term \( t \) if one of the following conditions holds:

- \( p = \varepsilon \).
- \( p = i.q, t = f(t_1, \ldots, t_n), q \) is positive in \( t_i, f \not\in \mathcal{D}, f \) is not \( \neg \) and either \( f \) is not \( \Rightarrow \) or \( i = 2 \).
- \( p = (i + 1).q, t = \hat{f}(a, t_1, \ldots, t_n), \hat{f} \in \mathcal{D}, q \) is positive in \( t_i \) and for every rule \( \hat{f}(a, x_1, \ldots, x_n) \rightarrow T \) in \( \mathcal{R}, \) \( x_i \) occurs only at positive positions in \( T \).
- \( p = \iota.q, t = \hat{f}(a, \vec{t}) \) and, if \( \hat{f}(n + 1, x_1, \ldots, x_n) \rightarrow I \) denotes the inductive rule of \( \hat{f} \) then:
  - All the terms of head \( \hat{f} \) occur at positive positions in \( I \).
  - \( q \) is positive in \( I\sigma \), where \( \sigma = \{ \vec{x}_{|F(\hat{f})} \mapsto \vec{s}_{|F(\hat{f})} \} \).
- \( p = \beta.q, t = \hat{f}(a, \vec{t}) \) and:
  - If \( \hat{f}(n + 1, x_1, \ldots, x_n) \rightarrow I \) is the inductive rule of \( \hat{f} \) then all the terms of head \( \hat{f} \) in \( I \) occur at positive positions.
  - If \( \hat{f}(0, x_1, \ldots, x_n) \rightarrow B \) is the inductive rule of \( \hat{f} \) then \( q \) is positive in \( B\sigma \), where \( \sigma = \{ \vec{x}_{|F(\hat{f})} \mapsto \vec{s}_{|F(\hat{f})} \} \).

\[ \hat{f}(0) \rightarrow p \]
\[ \hat{f}(u) \rightarrow \hat{f}(u) \Rightarrow q(u) \]
\[ \hat{g}(0, x) \rightarrow p \]
\[ \hat{g}(u, x) \rightarrow q(u, x) \Rightarrow \hat{g}(u, x) \]

\( \hat{f}(u) \) and \( \hat{g}(u, x) \) encode the schemata \((\ldots(p \Rightarrow q(0)) \Rightarrow \ldots) \Rightarrow p(u)) \) and \((q(u, x) \Rightarrow \ldots \Rightarrow q(0, x)) \ldots) \Rightarrow p \), respectively. The positions \( \beta, \iota \) are positive in \( \hat{g}(u, x) \), but not in \( \hat{f}(u) \). The position 2 is not positive in \( \hat{g}(u, x) \) (since \( x \) occurs in the scope of the negation in the inductive rule of \( \hat{g} \)).

\[ \blacklozenge \]

4 The inference system

Using the definitions in the previous section, we now adapt the usual inference rules of the paramodulation calculus. These rules operate on sets of terms of sort \textbf{bool}. As usual, all the rules must be applied modulo the usual AC properties of \( \lor \) and \( \land \). Furthermore, we assume that the initial set of schemata contains one instance of the reflexivity axiom \( x \simeq x \) for each sort symbol in \( \mathcal{S} \).
4.1 Boolean rules

We first use boolean simplification rules which perform an on-the-fly reduction of the terms at hand to negation normal forms. These rules terminate and preserve equivalence, thus they can be applied in a systematic way on all formulae. In the standard case, they would only be applied on the initial formula, as a pre-processing step, but this strategy is not applicable in our case because the application of the rules in $\mathcal{R}$ (using the Unfolding rule in Section 4.2) can create formulae that are not in normal form.

<table>
<thead>
<tr>
<th>boolean rule</th>
<th>simplified form</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg\text{false} \rightarrow \text{true}$</td>
<td></td>
</tr>
<tr>
<td>$\neg\text{true} \rightarrow \text{false}$</td>
<td></td>
</tr>
<tr>
<td>$\neg\neg\phi \rightarrow \phi$</td>
<td></td>
</tr>
<tr>
<td>$\neg(\phi \lor \psi) \rightarrow (\neg\phi) \land (\neg\psi)$</td>
<td></td>
</tr>
<tr>
<td>$\neg(\phi \land \psi) \rightarrow (\neg\phi) \lor (\neg\psi)$</td>
<td></td>
</tr>
<tr>
<td>$\phi \Rightarrow \psi \rightarrow \neg\phi \lor \psi$</td>
<td></td>
</tr>
<tr>
<td>$\text{true} \lor C \rightarrow \text{true}$</td>
<td></td>
</tr>
<tr>
<td>$\text{false} \lor C \rightarrow C$</td>
<td></td>
</tr>
<tr>
<td>$\text{true} \land C \rightarrow C$</td>
<td></td>
</tr>
<tr>
<td>$\text{false} \land C \rightarrow \text{false}$</td>
<td></td>
</tr>
</tbody>
</table>

The Distributivity rule is useful to reduce the obtained formulae into conjunction of clauses. By commutativity of $\lor$, $C \lor E$ can also be derived. The rule can also be applied with $C = \text{false}$, to derive $D$ and $E$ from a formula $D \land E$.

\[
\text{Distributivity: } \frac{C \lor (D \land E)}{C \lor D}
\]

4.2 Unfolding and folding rules

The Unfolding rule applies the rewrite rules in $\mathcal{R}$ to reduce the terms occurring in the formulæ.

\[
\text{Unfolding: } \frac{C[f(\tilde{s})]_p}{C[T\sigma]_p}
\]

If $p$ is a standard position, $\mathcal{R}$ contains a rule of the form $\hat{f}(\tilde{t}) \rightarrow T$, $\tilde{t}\sigma = \tilde{s}$. 
**Example 15** Consider the formula $\neg p(a) \lor \dot{p}(n+1,x)$ together with the rules $\dot{p}(0,y) \rightarrow q(y)$ and $\dot{p}(u+1,y) \rightarrow p(y) \land \dot{p}(u,y)$. The Unfolding rule applies, yielding: $\neg p(a) \lor (p(x) \land \dot{p}(n,x))$. Note that the obtained formula is not in conjunctive normal form. Afterward, the Distributivity rule can be applied to derive $\neg p(a) \lor p(x)$ and $\neg p(a) \lor \dot{p}(n,x)$.

The Folding rule does exactly the opposite: it replaces unfolded definitions by defined terms. At first glance, this rule may seem redundant, since one can assume that all terms are systematically replaced to their normal forms. However, it is sometimes useful to improve readability and conciseness by replacing complex expressions by simple terms. In contrast to the previous case, the variables can be instantiated to enable the folding.

| Folding: | $\frac{C[t|p]}{C[f(a,s)|p\sigma]}$ |
|----------|----------------------------------|
| If $p$ is a standard position, $R$ contains a rule of the form $f(a,s) \rightarrow r$ and $\sigma$ is an m.g.u. of $t$ and $r$. |

**Example 16** Consider the formula $p(a) \land \dot{p}(n,x)$, where $\dot{p}$ is defined as in Example 15. $p(a) \lor \dot{p}(n,x)$ unifies with the right-hand side of the inductive rule of $\dot{p}$, with the unifier $\{x \mapsto a, y \mapsto a, u \mapsto n\}$. Thus the Folding rule applies and we get: $\dot{p}(n+1,a)$ (note that the obtained formula is not equivalent to the initial one).

4.3 **Case analysis rule**

The following rule asserts conditions enabling the application of the Unfolding rule. These conditions can be used to encode the fact that any arithmetic term is either equal to 0 or of the form $u + 1$ to enable the application of one of the rules in $R$. These constraints are simply attached to the consequent clause as literals.

| Case Analysis: | $\frac{C[a|p]}{a \nless a' \lor C[a'|p]}$ |
|----------------|----------------------------------|
| If $a$ is a term of type nat, $p$ is a standard position, $a'$ is either 0 or a term of the form $u + 1$, where $u$ is a fresh arithmetic variable (not occurring in $C$). |
Example 17 Consider the clause \( \hat{p}(n, x) \), where \( \hat{p} \) is defined as in Example 15. The Unfolding rule does not apply on this clause, since \( n \) is neither 0 nor of the form \( u + 1 \). The Case analysis rule yields either \( n \neq 0 \lor \hat{p}(0, x) \) or \( n \neq u + 1 \lor \hat{p}(u + 1, x) \). Afterwards, the Unfolding rules applies, yielding \( n \neq 0 \lor q(x) \) and \( n \neq u + 1 \lor (p(x) \land \hat{p}(u, x)) \).

4.4 Paramodulation rule

We now define the most important rule of the calculus, namely the extended paramodulation rule. The main difference with the standard paramodulation rule (see, e.g., [24]) is that we have to take care of the “extra-variables” occurring in the replaced subterm but not in the parent clause \( C \). Indeed, as explained in Section 3, these variables denote terms whose value is not known, and that will actually vary during the evaluation of \( C \). Thus these variables should not be instantiated during the unification process (since they do not correspond to universally quantified variables, but can be viewed instead as placeholders for arbitrary terms), and, moreover, the other variables in \( C \) should not depend on them. These constraints yield additional application conditions on the rule. Note that the variables from the other parent clause are allowed to depend on these extra-variables because, in this case, one can then consider several instances of this parent clause, one for each value taken by the extra-variable during the unfolding process. The formal definition of the rule is given below (\( C \) is the “into” clause, i.e., the clause in which the replacement is performed and \( r \simeq l \lor E \) is the “from” clause).

<table>
<thead>
<tr>
<th>Paramodulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ C \quad r \simeq l \lor E ]</td>
</tr>
<tr>
<td>[ C[(D</td>
</tr>
</tbody>
</table>

If \( D = C|_p \) is of sort \text{bool}, \( p \) is positive, \( t = D|_q \), \( \sigma \) is an m.g.u. of \( t \) and \( r \) and the following conditions holds:

1. \( \text{dom}(\sigma) \subseteq \text{var}(r) \cup \text{var}(C) \).
2. For every variable \( x \in \text{dom}(\sigma) \) if \( x\sigma \) contains a variable not occurring in \( \text{var}(D) \cup \text{var}(r \simeq l \lor E) \), then \( x \) occurs neither in \( C \), nor in \( E \).
3. For every variable \( x \in \text{dom}(\sigma) \) if \( x\sigma \) contains a variable not occurring in \( \text{var}(C) \cup \text{var}(r \simeq l \lor E) \), then \( x \) does not occur in \( C \).
4. \( r \) and \( t \) are of a sort distinct from \text{nat}.

The definition of the rule is close to that of the usual paramodulation rule but there are important differences, which we now explain in details. The position at which the replacement is performed is written as the concatenation of two positions \( p.q \), where \( p \) denotes some positive position of a subterm \( D \) of sort.
bool and \( q \) the position of the replaced term \( t \) in \( D \). The remaining part \( E\sigma \) of the “from” clause \( r \simeq l \lor E \) is not added at the root level in the conclusion, but rather at the position \( p \). The reason is that this clause possibly contains variables occurring in \( C|_p = D \) but not in \( C \) (the extra-variables). The disjunction must be added at the context in which those variables are defined and not at the root level, as illustrated below.

**Example 18** Consider the clause \( \check{p}(n,a) \) with the rules: \( \check{p}(0,x) \to \text{false} \) and \( \check{p}(u + 1,x) \to p(f(x)) \lor \check{p}(u,x) \). Assume that the “from” clause is: \( f(y) \simeq y \lor q(y) \). \( f(y) \) and \( f(x) \) have a unifier \( \{ y \to x \} \). The replacement of \( f(x) \) by \( x \) in \( p(f(x)) \) produces \( p(x) \). The application of the rule yields: \( \check{p}'(n,a) \), with the rules \( \check{p}'(0,x) \to \text{false} \) and \( \check{p}'(u + 1,x) \to p(x) \lor q(x) \lor \check{p}'(u,x) \). Obviously the literal \( q(x) \) of the from clause must be inserted in the context where \( x \) is introduced, namely in the right-hand side of \( \check{p}' \). Adding it at the root level would be unsound, since the variable \( x \) is not defined at this point (i.e. it does not occur in \( C \)).

The position \( p \) must be positive, otherwise soundness is not preserved.

**Example 19** From \( \neg p(a) \) and \( a \simeq b \lor r \), it is incorrect to deduce \( \neg(p(b) \lor r) \) (the conclusion entails \( \neg r \) which is obviously not a logical consequence of the premises). The correct application is the standard one: \( \neg p(b) \lor r \). Thus in this case \( p \) must be \( \varepsilon \) and not \( 1 \), because \( 1 \) is not a positive position.

Sometimes there are several correct ways of applying the rule, with the same replacement, depending on the way the position is decomposed:

**Example 20** From \( p(a) \land p(b) \lor p(c) \) and \( a \simeq a' \lor p(d) \), one can deduce \( (p(a') \land p(b) \lor p(d)) \lor p(c) \lor (p(a') \land p(b) \lor p(d)) \lor p(c) \lor (p(a') \land p(b) \lor p(c) \lor p(d)) \), depending on how the position \( 1.1.1 \) is decomposed: \( (1.1).1, 1.(1.1) \) or \( \varepsilon.(1.1.1) \) (indeed, in this case \( (1.1), 1 \) and \( \varepsilon \) are all positive).

To avoid the duplication of subformulas (by subsequent applications of the Distributivity rule), the best strategy is to choose the minimal position satisfying the conditions (\( \varepsilon \) in the previous example).

Note that the term \( t \) may occur several times in the unfolded form of \( C \) (with different values of the extra-variables). All these occurrences are simultaneously replaced by \( l \). Thus an application of the paramodulation rule on clause schemata corresponds to an unbounded number of applications of the paramodulation rule on standard clauses, using different copies of the clause \( r \simeq l \lor D \).

**Example 21** Consider the clauses \( \check{f}(n) \simeq a \) and \( g(x) \simeq g'(x) \), where \( \check{f} \) is associated with the rules: \( \check{f}(0) \to a \) and \( \check{f}(u + 1) \to b(g(a(u)), \check{f}(u)) \). The paramodulation rule applies, with \( p = 1 \cdot 1 \cdot t = g(a(u)) \), \( \sigma : \{ x \to a(u) \} \). This yields the clause \( \check{f}'(n) \simeq a \), together with the rules: \( \check{f}'(0) \to a \) and \( \check{f}'(u + 1) \to h(g'(a(u)), \check{f}'(u)) \). This single inference corresponds to a sequence of \( n \) applications of the usual Paramodulation rule on \( h(g(a(n - 1))), \ldots, h(g(a(0)), a) \ldots \) \( \simeq a \), with the “from” clauses \( g(a(n - 1)) \simeq g'(a(n - 1)), \ldots, g(a(0)) \simeq g'(a(0)) \).
A similar version of the rule can be defined, in which the “from” literal is allowed to occur inside an iteration. In this case, several copies of the “into” clauses must be used:

### Paramodulation (2)

\[
\frac{C \quad D}{(D[C[l\sigma]_p\sigma]_q)\sigma}
\]

If \( t = C|_{p}, r \simeq l = D|_{q}, p \) is a standard position, \( \sigma \) is an m.g.u. of \( t \) and \( r \) and the two following conditions holds:

- \( \text{dom}(\sigma) \subseteq \text{var}(C) \cup \text{var}(D) \).
- For every variable \( x \) if \( x\sigma \) contains a variable not occurring in \( \text{var}(C) \cup \text{var}(D) \), then \( x \) does not occur in \( D \).
- \( r \) and \( t \) are of a sort distinct from \text{nat}.

The definition is slightly simpler than in the previous case.

### 4.5 Resolution rule

The Resolution rule is defined similarly. This rule replaces a literal \( M \) (possibly occurring at a “deep” position inside the rewrite system) by a clause \( D \), if a clause \( L \vee D \) occurs in the set of formulæ (up to unification).

### Resolution

\[
\frac{C \quad L \vee D}{(C[D\sigma]_p)_\sigma}
\]

If \( M = C|_{p}, \sigma \) is an m.g.u. of \( L \) and \( M \) and the two following conditions holds:

- \( \text{dom}(\sigma) \subseteq \text{var}(L) \cup \text{var}(D) \).
- For every variable \( x \in \text{dom}(\sigma) \) if \( x\sigma \) contains a variable not occurring in \( \text{var}(C) \cup \text{var}(L) \), then \( x \) does not occur in \( C \).

### 4.6 Factorization rule

The Factorization rule is defined as usual, except that it can be applied at an arbitrary position in the considered formula.
Factorization: \[
\frac{C}{C[L \lor D]_p\sigma}
\]

If \(C\) is of the form \(L \lor L' \lor D\), \(\sigma\) is an m.g.u. of \(L\) and \(L'\), \(\text{dom(}\sigma\text{)} \subseteq \text{var(}C\text{)}\) and for every variable \(x \in \text{dom(}\sigma\text{)}\), \(\text{var(}x\sigma\text{)} \subseteq \text{var(}C\text{)}\).

4.7 Induction rule

We write \(S \vdash C\) (resp. \(S \vdash^* C\)) if \(C\) can be deduced from premises in \(S\) by an application (resp. by a finite sequence of applications) of the inference rules. This relation is (inductively) extended by using a very simple induction scheme:

Induction: \[
\frac{S}{C[u]_p}
\]

If \(u\) is a fresh arithmetic variable, \(S \vdash^* C[0]_p\), \(S \cup \{C[k]\} \vdash^* C[k+1]_p\) for some constant symbol \(k\) of sort \(\text{nat}\) not occurring in \(S\) and \(C\).

5 Soundness

The following theorem states that the calculus is sound.

**Theorem 22** (Soundness) If \(S \vdash C\), then \(S \models C\).

**Proof.** The soundness of the Unfolding and Folding rules is a direct consequence of Condition 3 in Definition 5. The soundness of the Simplification and Distributivity rules stems from Condition 2 in Definition 5 and of the usual properties of the operators \(\lor\) and \(\land\). The soundness of the Case analysis rule is immediate: for every model \(I\) of \(C[k]_p\) and for every substitution \(\sigma\), if \(I \not\models (a \not\equiv a')\sigma\) then we must have \(I(a\sigma) = I(a'\sigma)\), thus by substitutivity \(I \models C[a']_p\sigma\). The soundness of the Induction rule is a straightforward application of the induction principle.

We now consider the case of the key Paramodulation (1) rule (we omit the proofs for the Paramodulation (2), Resolution and Factorization rules that are similar and much simpler). Let \(C\) and \(r \simeq l \lor E\) be two terms of sort \(\text{bool}\) satisfying the application conditions of the rule, with two positions \(p\) and \(q\), a term \(D\), a term \(t\) and an m.g.u. \(\sigma\). Let \(\theta\) be a ground substitution of the variables occurring in \(C[(D[l\sigma]_p \lor E]\sigma]_p\sigma\). We prove that \(C\theta,l \simeq r \lor E \models C[(D[l\sigma]_q \lor E]\sigma]_p\sigma\theta\). The proof is by induction on the formula \(C\). The induction ordering \(\prec\) is defined as in the proof of Proposition 1. We assume that the
symbols \( \hat{f}' \) constructed by the replacement operation are \( \sim \)-equivalent to the corresponding defined symbol \( f \) in the initial term\(^1\) (see Definition 11). We distinguish several cases, according to the form of the positions \( p \) and \( q \) of the formula \( C \). We assume, w.l.o.g., that the range of \( \theta \) contains no defined symbol.

- Assume that \( p = i.p' \), where \( i > 0 \) and that \( C \) is of the form \( \hat{p}(a, t_1, \ldots, t_n) \), with \( \hat{p} \in D \). Then, by unfolding, \( C\sigma\theta \) is equivalent to a formula of the form \( T\gamma\sigma\theta \), where \( \mathcal{R} \) contains a rule \( \hat{p}(b, x_1, \ldots, x_n) \rightarrow T \), where:
  
  - If \( a\theta = 0 \) then \( b = 0 \) and \( \gamma = \{ x_i \mapsto t_i \mid i \in [1, n] \} \).
  
  - If \( a\theta > 0 \) then \( b \) is of the form \( u + 1 \), for some variable \( u \), and \( \gamma = \{ x_i \mapsto t_i \mid i \in [1, n] \} \cup \{ u \mapsto a\theta - 1 \} \).

  Note that we must have \( i > 0 \), since the replaced term cannot be of sort nat. By definition of the replacement operation, \( C([D[\sigma]_q \lor E]_p) \) is of the form \( \hat{p}(b, t_1, \ldots, t_{i-2}, t_i - 1) \). By unfolding, \( C([D[\sigma]_q \lor E]_p) = T\gamma\sigma\theta \), where \( \gamma' \) coincide with \( \gamma \), except on the variable \( x_{i-1} \), for which we have \( x_{i-1}' = t_i - 1 \). Since \( p \) is positive, \( x_i \) only occurs at some positive positions \( q_1, \ldots, q_m \) in \( T \). But then it is clear that \( T\gamma'\sigma \) can be obtained by a sequence of \( m \) applications of the Paramodulation rule from \( l \simeq r \lor E \), into \( T\gamma\sigma \) at positions \( q_j, p' \) (\( 1 \leq j \leq m \)). We have \( T\gamma\sigma\theta \simeq C([D[\sigma]_q \lor E]_p)\theta \) and furthermore, all the formulae deduced during the Paramodulation process are lower than \( T\gamma\sigma\theta \). Thus we may apply the induction hypothesis to deduce that \( T\gamma\sigma\theta, r \simeq l \lor E \models T\gamma'\sigma\theta \), which entails that that \( C\sigma\theta, l \simeq r \lor E \models C([D[\sigma]_q \lor E]_p)\sigma \).

- Assume that \( p = i.p' \), where \( i \in \mathbb{N} \) and that the head symbol of \( C \) is a non-defined symbol. Then (since \( p \) is positive and since the only non-defined symbols of sort bool are \( \lor, \land, \Rightarrow \) and \( \neg \) \( C \) must be of the form \( C_1 \ast C_2 \), where \( \ast \in \{ \lor, \land, \Rightarrow \} \) and if \( \ast \) is \( \Rightarrow \) then \( i = 2 \). By symmetry, we assume that \( i = 2 \). Then \( C([D[\sigma]_q \lor E]_p) = C_1 \ast C_2([D[\sigma]_q \lor E]_p) \). By the induction hypothesis, we have \( C_1\sigma\theta, l \simeq r \lor E \models C_2([D[\sigma]_q \lor E]_p)\sigma \), whence \( C\sigma\theta, l \simeq r \lor E \models C([D[\sigma]_q \lor E]_p)\sigma \).

- Assume that \( p = i.p' \), where \( i \in \{ \beta, \iota \} \). Then \( C \) is of the form \( \hat{p}(a, t_1, \ldots, t_n) \), with \( \hat{p} \in D \). By unfolding, \( C\sigma\theta \) is equivalent to a formula of the form \( T\gamma\sigma\theta \), where \( \mathcal{R} \) contains a rule \( \hat{p}(b, x_1, \ldots, x_n) \rightarrow T \), where \( b \) and \( \gamma \) are defined as in the first case above. By definition of the replacement operation, \( C([D[\sigma]_q \lor E]_p) \) is of the form \( \hat{p}(a, t_1, \ldots, t_n, \hat{y}) \), where \( \hat{y} \) denotes the set of variables occurring in \( (D[\sigma]_q \lor E) \), but not in \( n, x_1, \ldots, x_n \) and \( \mathcal{R} \) contains a rule \( \hat{p}'(b, x_1, \ldots, x_n, \hat{y}) \rightarrow T' \), where \( T' \) is obtained from \( T \) by:

\(^1\)Note that this convention is different from that of Definition 11, which aimed to ensure that \( \mathcal{R} \) is terminating.
Replacing every term of the form $p(u)$ by $p'(u, y)$. By definition, since only the fixed arguments of $p$ are instantiated when computing the subterms, we have $p(u)\gamma[p = p(a, t_1, \ldots, t_n)]p = C[p = D].$ Moreover, $p(u)\gamma[(D[l][\sigma]_q \lor E)]p = p'(u\gamma, y)$.

Possibly replacing a subterm occurring at position $p'$ in $T$ by $(D[l][\sigma]_q \lor E)\gamma$, in the case in which $T\gamma[l]\gamma = D$, where $\gamma'$ is the restriction of $\gamma$ to the fixed arguments of $p$. Then we have $T\gamma[l]\gamma = D\gamma$ and $D\gamma_q = t\gamma$. But by Properties 1 and 3 in the application conditions of the Paramodulation rule, we must have $t\gamma\sigma = t\gamma \sigma$. Thus $D\gamma_q \sigma = t\gamma \sigma$. Moreover, $D[l][\sigma]_q \gamma = D\gamma[l][\sigma]_q \gamma$.

Consequently, each of these replacement steps can be simulated by an application of the Paramodulation rule from the clause $r \simeq l \lor E$ into $T\gamma$. By the induction hypothesis, we deduce that $T\gamma\sigma\theta, r \simeq l \lor E \models T'\gamma\sigma\theta$, whence $C\sigma\theta, r \simeq l \lor E \models C[(D[l][\sigma]_q \lor E)]p \sigma\theta$.

- Assume that $p = \varepsilon$ and that $q = i, j, q'$, where $j \in \{\beta, \iota\}$. In this case $C$ is of the form $p(s_1, \ldots, s_m)$, where $s_i$ is of the form $f(a, t_1, \ldots, t_n)$, with $f \in D$. By unfolding, $C\sigma\theta$ is equivalent to a formula of the form $p(s_1, \ldots, s_{i-1}, T\gamma, s_{i+1}, \ldots, s_m)\sigma\theta$, where $R$ contains a rule $f(b, x_1, \ldots, x_n) \rightarrow T$, with $b$ and $\gamma$ defined as above. Since $p = \varepsilon$, we have $C = D$ and $C[(D[l][\sigma]_q \lor E)]p = (D[l][\sigma]_q \lor E)\sigma$. By definition of the replacement operation, $D[l][\sigma]_q \lor E$ is of the form $p(s_1, \ldots, s_{i-1}, f'(a, t_1, \ldots, t_n), y, s_{i+1}, \ldots, s_n)$, where $y$ denotes the set of variables occurring in $l\sigma$, but not in $b, x_1, \ldots, x_n$. Furthermore, $R$ contains a rule $f'(b, x_1, \ldots, x_n, y) \rightarrow T'$, where $T'$ is obtained from $T$ by:

- Replacing every term of the form $f(u)$ by $f'(u, y)$. By definition, we have $f(u)\gamma[p = D]\gamma = t$. Moreover, $f(u)\gamma[l][\sigma]_q = f'(u\gamma, y)$.

- Possibly replacing a subterm occurring at position $q'$ in $T$ by $v\sigma$, in the case in which $T\gamma[l]\gamma = t$, where $\gamma'$ is the restriction of $\gamma$ to the fixed arguments of $\tilde{f}$. Then we have $T\gamma[l]\gamma = T\gamma$. But by Properties 1 and 2, $t\gamma\sigma = t\gamma \sigma = r\gamma \sigma$. Note also that by Property 2, we have $E\sigma\gamma = E\sigma$.

Consequently, a formula $T'\gamma\sigma \lor E\sigma \lor \ldots \lor E\sigma \equiv T'\gamma\sigma \lor E\sigma$ can be obtained by a sequence of Paramodulation steps from the clause $r \simeq l \lor E$ into $T\gamma$. By the induction hypothesis, we deduce that $T\gamma\sigma\theta, l \simeq r \lor E \models T'\gamma\sigma\theta$, whence $D\sigma\theta, l \simeq r \lor E \models (D[l][\sigma]_q \lor E)\sigma\theta$.

- Assume that $p = \varepsilon$ and that $q = r, q'$, where $r$ is a standard position, and either $r$ is of length greater than 1 or $r$ is of length 1 and the head symbol of $C$ is not the equality predicate (this last case is covered in the next item). Let $t' = C|\gamma$. It is clear that we have $C \models C[x]_r \lor x \neq t'$. By Paramodulation from $l \simeq r \lor E$ into $C[x]_q \lor x \neq t'$, we get: $C[x]_q \lor x \neq t'[l][\sigma]_q \lor E \models (C[t'][l][\sigma]_q)_r \lor E \sigma$. By the induction hypothesis, we have
(C[x] ∨ x \neq t')σθ, l \simeq r \lor E \models (C[t'[σ|q']_r ∨ E)σθ. But C[t'[σ|q']_r = C[σ|q]_r. Thus Cσθ, r \simeq l \lor E \models C[σ|q]_r ∨ E)σθ.

- If the previous conditions are not satisfied, then C must be of the form t \simeq s. Then the rule is equivalent to the usual Paramodulation rule, and the proof follows from well-known results (see, e.g., [24]).

6 Completeness issues

It is clear that the unsatisfiability problem is not semi-decidable for schemata of formulæ, and thus the calculus presented in Section 4 cannot be refutationally complete. Indeed, as shown in Section 2, all diophantine equations can be encoded as equalities between term schemata. From an equation \( t \equiv s \) between non-linear arithmetic expressions, it is possible to construct a schema \( t' \simeq s' \) such that \( t' \simeq s' \) is satisfiable iff \( t \equiv s \) has a solution. The terms \( t' \) and \( s' \) contain no variables: the arithmetic variables occurring in the initial equation are encoded as parameters. Therefore, since diophantine equations are unsolvable, even the ground fragment of the logic is not semi-decidable. A natural restriction is thus to assume that the rewrite system \( R \) is defined in such a way that unification modulo \( R \) is decidable (this is the case for some classes of systems, e.g., primal grammars [20]). However, this restriction is not sufficient for completeness. Indeed, it is shown in [4] that the satisfiability problem is undecidable for purely propositional schemata (i.e., ground schemata in which the signature contains only predicate symbols of sort or \( \text{nat}^k \rightarrow \text{bool} \)), although the unification problem is decidable for such expressions. Since it is clear that the satisfiability problem is semi-decidable for propositional schemata (the set of instances is recursively enumerable), this entails that no refutationally complete proof procedure exists. This result also holds if the predicate symbols are assumed to be monadic (i.e. with \( k \leq 1 \)), provided expressions of the form \( u + u \) (where \( u \) is a variable) are allowed. However, if all the non-ground arithmetic expressions are of the form \( u + k \) (with \( k \in \mathbb{N} \)), then satisfiability can be tested in finite time in the propositional (monadic) case [4]. However, such a condition is still not sufficient for completeness in the non-propositional case: for instance it is shown in [6] that the satisfiability problem is undecidable for schemata of ground (monadic) equational formulæ fulfilling the previous condition.

Despite all these negative results, identifying subclasses of formulæ for which completeness can be ensured is of great theoretical and practical interest.

6.1 A weak completeness result

First, we can remark that the calculus allows one to generate all the implicates of the form \( \vec{n} \neq \vec{k} \), where \( \vec{n} \) is a vector of parameters and \( \vec{k} \) a vector of natural number. Indeed, by using the Case analysis rule it is possible to fix the value of the parameters, and thus to enumerate all possible \( \text{nat} \)-evaluations of a schema.
Then, for each of these nat-evaluations, a refutation can be constructed by using the usual rules. This yields the following result.

**Theorem 23** (Weak Completeness) Let $S$ be a set of formulæ and let $\eta$ be a nat-valuation. If $S\eta$ is unsatisfiable, then there exists a derivation from $S$ of an arithmetic formula $\phi$ such that $\phi\eta$ is unsatisfiable (w.r.t. Presburger arithmetic).

**Proof.** Let $C$ be a formula in $S$, containing an occurrence of a parameter $n$ at a (standard) position $p$. If $\eta(n) = 0$ then an application of the Case Analysis rule yields the clause: $n \not\equiv 0 \lor C[0]_p$. Similarly, if $\eta(n) > 0$ we get: $n \not\equiv u + 1 \lor C[u + 1]_p$. The Case Analysis rule can be applied again on this last formula, on the term $u$, yielding either $n \not\equiv u + 1 \lor u \not\equiv 0 \lor C[1]_p$ or $n \not\equiv u + 1 \lor u \not\equiv v + 1 \lor C[v + 2]_p$.

More generally, by using $\eta(n) + 1$ applications of the Case Analysis rule, we get a formula of the form: $n \not\equiv u_0 + 1 \lor \bigvee_{i=1}^{\eta(n)-1} u_{i-1} \not\equiv u_i + 1 \lor u_{\eta(n)} \not\equiv 0 \lor C[\eta(n)]_p$.

Notice that the formula $n \not\equiv u_0 + 1 \lor \bigvee_{i=1}^{\eta(n)-1} u_{i-1} \not\equiv u_i + 1 \lor u_{\eta(n)} \not\equiv 0$ is equivalent to $n \not\equiv \eta(n)$. Furthermore, it shares no variable with $C[\eta(n)]$ (since the $u_i$’s are fresh variables). By repeating this process on all occurrences of parameters, we get a formula of the form $C' \lor \eta(C)$, where $\eta(C')$ is unsatisfiable and share no variable with $\eta(C)$. $\eta(C)$ contains no parameter, hence can be transformed into a set of standard clauses by the Unfolding and Distributivity rules. The set of clauses $\{\eta(C) \mid C \in S\}$ is unsatisfiable, thus there exists a derivation of the empty clause using the standard Resolution, Factorisation and Paramodulation rules, which can be straightforwardly simulated by our calculus. Since $C'$ share no variable with $\eta(C)$, this refutation can be easily be transformed in a derivation from $\{C' \lor \eta(C) \mid C \in S\}$ of a clause subsuming $\bigvee_{C \in S} C'$. It is straightforward to check that $\bigvee_{C \in S} C'$ fulfills all the conditions of the theorem.

The previous theorem entails that if a set of formulæ $S$ is unsatisfiable, then we can derive from $S$ a set $S'$ that is purely arithmetic and also unsatisfiable. This result does not imply semi-decidability since $S'$ is infinite in general (the set of nat-valuations is infinite), and since Presburger arithmetic is not compact.

### 6.2 A complete class

In this section, we devise syntactic conditions ensuring completeness. The obtained class can be seen as an extension of the regular schemata defined in [4].

The conditions are much more complex to define than those of [4], but they are also much more general: they allow for schemata containing multiple parameters, complex arithmetic expressions and quantifiers. The general principle guiding the definition of the class is that we seek to ensure that a refutation can be obtained from any unsatisfiable schemata by using a sequence of $k$ applications of the induction principle, one for each parameter occurring in the schema. To this aim, the parameters have to be ordered in a sequence $n_1, \ldots, n_k$, in such a way that the truth value of the symbols defined by induction on the parameter $n_i$ only depend on the parameters $n_j$ with $j \geq i$. We then associate to each
parameter a set of atoms depending on a unique arithmetic variable $u$, so that every non-standard atom occurring the formula will be of the form $\pi\{u \rightarrow n + k\}$ or $\pi\{u \rightarrow k\}$, where $\pi$ is an atom associated with parameter $n$ and $k$ is a natural number. We first introduce the notion of a $\pi$-sequence to denote the sequence of parameters and their corresponding atoms.

**Definition 24** A $\pi$-sequence of order $k$ is a finite sequence of pairs $(n_i, \Pi_i)$ $(1 \leq i \leq k)$ satisfying the following properties.

1. $n_1, \ldots, n_k$ are pairwise distinct parameters.

2. For every $i \in [1, k]$, $\Pi_i$ is a finite set of non-equational atoms built on the parameters $n_{i+1}, \ldots, n_k$, a unique variable $u$ of sort $\text{nat}$ and the set of function and predicate symbols $\Sigma$.

If $\pi \in \Pi$ and $t$ is a term of sort $\text{nat}$, then $[\pi[t]$ denotes the atom $\pi\{u \rightarrow t\}$ (note that if $t$ is ground then so is $\pi[t]$).

We impose some additional restrictions on the common instances of the atoms in $\Pi_1, \ldots, \Pi_k$:

**Definition 25** A $\pi$-sequence of order $k$ $(n_i, \Pi_i)$ $(1 \leq i \leq n)$ is local the following condition holds.

1. For every $i \in [1, k]$ for every valuation $\eta$, and for every $n, m \in \mathbb{N}$, if $\pi, \xi \in \Pi_i$ and $\pi[n]\eta = \xi[m]\eta$ then we must have $\pi = \xi$ and $n = m$.

2. There exist $k$ natural numbers $l_1, \ldots, l_k$ such that:

   (a) For every $i, j \in [1, k]$ and for every valuation $\eta$, if $n, m \in \mathbb{N}$, $i < j$, $\pi \in \Pi_i$, $m \leq \eta(n_j) + l_j$, $\xi \in \Pi_j$ and $\pi[n]\eta = \xi[m]\eta$, then $n \leq l_i$.

   (b) For every $i \in [1, k - 1]$, for every $\pi \in \Pi_i$ and for every $n \leq l_i$, there exists $\xi \in \Pi_{i+1}$ and $m \leq l_{i+1}$ such that $\pi[n] = \xi[n_{i+1} + m]$. ⊢

The intuition behind Condition 1 is that all equations between atoms must be reducible to a mere syntactic identity test. Condition 2.a states that the common instances of two atoms $\pi_i$ and $\pi_j$ in $\Pi_i$ and $\Pi_j$ with $i < j$ correspond to small values of the arithmetic variable in $\pi_i$ (i.e., to a value that is lower than some fixed constant $l_i$), whereas Condition 2.b states that all the small instances of the atoms in $\Pi_i$ are also instances of an atom in $\Pi_{i+1}$. Note that these conditions are easy to test, using existing decision procedures for Presburger arithmetic (see for instance [12, 25]).

**Example 26** An example of a local $\pi$-sequence of order 2 is $((n_1, \{p(u + n_2 + 1), q(u)\}), (n_2, \{q(0), \ldots, q(l_1), p(u), q(u, a)\}))$ (with $l_1 \in \mathbb{N}$ and $l_2 = l_1 + 1$). The $\pi$-sequence of order 1 $(n_1, \{p(u), p(u + 1)\})$ is not local, since $p(u)$ and $p(u + 1)$ have a common instance, but are not syntactically equivalent. Similarly, $((n_1, \{p(2,u), p(2,u + 1)\})$ is local (with any $l_1 \in \mathbb{N}$), but $((n_1, \{p(2,u), p(2,u + 1)\}), (n_2, \{p(u)\}))$ is not, because, e.g., $p(2,u)[0] = p(u)[0]$. ♠
Definition 27 A non-equational formula $\phi$ is compatible with a local $\pi$-sequence of order $k$ ($n_i, \Pi_i$) ($1 \leq i \leq k$) (associated with natural numbers $l_1, \ldots, l_i$) if the following conditions hold.

1. For every non-defined atom $A$ occurring in $\phi$, either $A$ contains no parameter and is not unifiable with the atoms in $\Pi_i$ or $A$ is of the form $\pi[n_i + j]$ or $\pi[j]$, for some $i \in [1, k]$, $\pi \in \Pi_i$ and $j \leq l_i$.

2. Every symbol in $D$ is monadic, $\prec$ is empty and there exists a partition $D_1, \ldots, D_k$ of $D$ such that the following conditions hold.

(a) The only defined terms in $\phi$ are of the form $\hat{f}(n_i)$, where $i \in [1, k]$, $\hat{f} \in \Pi_i$.

(b) For all $i \in [1, k]$ and for all symbols $\hat{f} \in D_i$, every atom occurring in the right-hand side of the inductive rule $\hat{f}(u) \rightarrow I$ of $\hat{f}$ is of the form $\pi[u + j]$, for some $i \in [1, k]$, $\pi \in \Pi_i$ and for some $j \leq l_i$.

(c) For all $i \in [1, k]$ and for all symbols $\hat{f} \in D_i$, every atom occurring in the right-hand side of the base rule $\hat{f}(u) \rightarrow B$ of $\hat{f}$ is of the form $\pi[j]$, for some $i \in [1, k]$, $\pi \in \Pi_i$ and $j \leq l_i$.

Definition 28 The formula $\phi$ is weakly regular if it there exists a local $\pi$-sequence $B$ such that $\phi$ is compatible with $B$. \hfill \diamond

A weakly regular formula possibly contains quantifiers (for instance all standard first-order clause sets are weakly regular) but their interaction with schemata is strongly restricted: according to Conditions 1 and 2 in Definition 27, the purely schematic part of the formula must be ground and non-equational. It is easy to see that all regular schemata (in the sense of [4]) are weakly regular, but the same property does not hold for nested-regular or bound-linear schemata (which allow for nested iterated connectives\(^2\)).

The completeness proof is based on a decomposition scheme that is similar to that forming the basis of the usual Davis and Putnam procedure [13]. It is formally defined as follows. For every clause set $S$ and for every ground literal $L$, we denote by $S_L$ the set of clauses $C$ such that $L \notin C$ and $C \lor L^c \in S$. If $A$ is a ground atom, we denote by $S(A)$ the set of clauses $(S_A \otimes S_{\neg A})$, where $S_1 \otimes S_2 \overset{\text{def}}{=} \{ C \lor D \mid C \in S_1, D \in S_2 \}$. From a semantic point of view, $S_L$ denotes the clause set obtained from $S$ by evaluating $L$ to true and $S_1 \otimes S_2$ corresponds to the disjunction of $S_1$ and $S_2$. Therefore $S(A)$ is the disjunction of the clause sets obtained from $S$ by evaluating $A$ to true and false respectively. In the standard (propositional) case, it is well-known that $S$ and $S(A)$ are sat-equivalent. This property does not hold in our context, because the rules in $\mathcal{R}$ can impose additional conditions on the interpretations. For instance the set $S = \{ \hat{f}(n), \neg p(0) \}$, with the rules $\hat{f}(0) \rightarrow \text{true}$ and $\hat{f}(u + 1) \rightarrow p(u) \land \hat{f}(u)$ is

\(^2\)Note that all nested-regular or bound-linear schemata can be reduced to a sat-equivalent regular one, at the cost of an exponential blow-up [4].
unsatisfiable, whereas \( S(p(0)) = \{ f(n) \} \) is satisfiable. Nevertheless, the above relation still holds under some additional conditions:

**Proposition 29** Let \( S \) a set of clauses that is compatible with a local \( \pi \)-sequence of order \( k \) \((n_i, \Pi_i)\), associated with the numbers \( t_1, \ldots, t_k \). Let \( A \) be a ground atom of the form \( \pi[n_1 + l_1 + 1] \), with \( \pi \in \Pi_1 \). Then \( S(A) \) is sat-equivalent to \( S \).

**Proof.** Assume that \( S(A) \) has a model \( I \). This implies that \( I \models S_A \) or \( I \models S_{\neg A} \). We assume by symmetry that \( I \models S_A \). Let \( J \) be an interpretation coinciding with \( I \), except that \( J(A) \equiv \text{true} \). By definition, we have \( J \models S \) iff \( J \models S_A \). Let \( \eta \) be the restriction of \( I \) to the parameters. We have \( J \models S_A \) iff \( J \models S_A^{\eta} \). We remark that every atom \( B \) occurring in \( S_A^{\eta} \) is of the form \( \xi[k] \eta \), where \( \xi \in \Pi_i \) and \( k \leq \eta(n_i) + l_i \). Indeed, Condition 1 in Definition 27 ensures that all the atoms occurring in \( S_A^{\eta} \) have this property, and Condition 2 ensures that this property is preserved when applying the rewrite rules in \( R \). Assume that \( B \) and \( A^{\eta} \) are syntactically equivalent. By Condition 2.a in Definition 25, we must have \( \xi \in \Pi_i \) and by Condition 1, we deduce that \( \pi[n_1 + l_1 + 1] = \xi[k] \). This last property implies that \( B \) occurs in \( S_A^{\eta} \) (it cannot be introduced by applying the rules in \( R \), since these rules cannot increase the argument of the defined symbols), i.e. is of the form \( B^{\prime} \eta \), for some \( B^{\prime} \) occurring in \( \phi_A \). Furthermore, \( B^{\prime} \) and \( A \) are syntactically equivalent, which is absurd (by definition of \( S_A \)). Therefore, \( S_A^{\eta} \) contains no atom equivalent to \( A \) and thus \( I \) and \( J \) coincide on \( S_A^{\eta} \). This entails that \( J \models S_A^{\eta} \), whence \( J \models S \). The converse is straightforward. \( \blacksquare \)

If \( E = \{ A_1, \ldots, A_n \} \) is a set of atoms, then \( S(E) \) denotes the set \( S(A_1) \ldots S(A_n) \) (the order in which the elements are considered can be chosen arbitrarily).

**Theorem 30** Let \( S \) be a weakly regular set of formulæ. If \( S \) is unsatisfiable then there exists a derivation from \( S \) of a purely arithmetic formula \( \phi \) such that \( \phi \) is unsatisfiable (in Presburger arithmetic).

**Proof.** \( S \) is compatible with a local \( \pi \)-sequence \((n_i, \Pi_i)\). The proof is by induction on \( k \). If \( k = 0 \) then \( S \) is a set of standard formulæ (up to the evaluation of ground arithmetic formulæ with no parameter), which can be transformed into an equivalent set of clauses by applying the Distributivity rule, thus the result follows by the usual completeness of the paramodulation calculus on standard clauses [24]. Thus we assume that \( k \geq 1 \). Let \( t_i \) be a sequence of terms such that \( t_0 \equiv n \) and all the \( t_{i+1} \)'s are pairwise distinct variables not occurring in \( S \) or \( R \). We construct inductively a sequence of unsatisfiable sets of clauses \( T_i \) of the form \( \{ n \geq i \} \cup \{ n \neq t_i + i \} \otimes S_i \) (for \( i \in \mathbb{N} \)), in such a way that each set \( T_{i+1} \) can be derived from \( T_i \).

- We take \( T_0 \equiv S_0 \equiv S \) (note that, since \( t_0 = n \), the inequation \( n \neq t_i + i \) is equivalent to \( \text{false} \), furthermore \( n \geq 0 \) is equivalent to \( \text{true} \)).

- Assume that \( T_i = \{ n \geq i \} \cup \{ n \neq t_i + i \} \otimes S_i \) has been derived. We first show how to define the set \( \{ n \neq t_{i+1} + i + 1 \} \otimes S_{i+1} \). By applying the Case
analysis rule on the term $t_i$ (replacing it by $t_{i+1} + 1$), we can derive from
\{n \neq t_{i+1} + 1\} \otimes S_i$ a set of clauses of the form $\{n \neq t_i + i \lor t_i \neq t_{i+1} + 1\} \otimes S'_{i+1}$,
where $S'_{i+1}$ is obtained from $S_i$ by replacing all occurrences of $t_i$ by $t_{i+1} + 1$
and by normalizing the obtained formula with respect to the rules in $\mathcal{R}$
and to the Simplification and Distributivity rules. By resolving with the
reflexivity axiom $x \simeq x$, we get the set $\{n \neq t_{i+1} + (i + 1)\} \otimes S'_i$.
Finally, we construct a set of clauses $\{n \neq t_{i+1} + i + 1\} \otimes S_{i+1}$ by saturating the
set $\{n \neq t_i + i \lor t_i \neq t_{i+1} + 1\} \otimes S'_{i+1}$ w.r.t. all resolution and factorization
inferences operating on the literals of the form $\pi[t_{i+1} + l_i + 1]$, with $\pi \in \Pi$,
and then by discarding the clauses containing such literals (note that the
obtained clause set is compatible with the considered $\pi$-sequence, since
all the literals remaining in the formula necessarily fulfill Condition 1 in
Definition 27). It is clear that $\{n \geq i + 1\} \cup (\{n \neq t_{i+1} + (i + 1)\} \otimes S'_{i+1}$
$= \{n \geq i\} \cup (\{n \neq t_i + i\} \otimes S_i)$, thus since the latter set is unsatisfiable, so
must be $\{n \geq i + 1\} \cup (\{n \neq t_{i+1} + (i + 1)\} \otimes S'_{i+1}$). By definition, $\{n \neq t_{i+1} +$ $i + 1\} \otimes S_{i+1}$ is equivalent to $\{n \geq i + 1\} \cup (\{n \neq t_{i+1} + (i + 1)\} \otimes S'_{i+1})(E)$,
where $E$ is the set of atoms of the form $\pi[t_{i+1} + l_i + 1]$, with $\pi \in \Pi$.
Consequently, by Proposition 29, $\{n \geq i + 1\} \cup (\{n \neq t_{i+1} + i + 1\} \otimes S_{i+1}$)
must be unsatisfiable. Note that, by construction, $\{n \neq t_{i+1} + i + 1\} \otimes S_{i+1}$
can be derived from $T_i$.

We then show how to prove that $n \geq (i + 1)$. By applying again the Case
analysis rule, still on the term $t_i$ but this time replacing it by 0, we can
derive from $\{n \neq t_i\} \otimes S_i$ a set of clauses of the form $\{n \neq t_i \lor t_i \neq 0\} \otimes S''_i$,
where $S''_i$ is obtained from $S_i$ by replacing all occurrences of $t_i$ by 0
and normalizing the obtained formula with respect to the rules in $\mathcal{R}$
and to the Simplification and Distributivity rules. Since $T_i$ is unsatisfiable, so
must be $S''_i$ (since $S''_i$ is an instance of $T_i$, for $n = i$). It is easy to check
that the set $S''_i$ is compatible with the $\pi$-sequence of order $k - 1$ ($n_i, \Pi$),
with $2 \leq i \leq k$. Indeed, all the atoms of the form $\pi[m]$ with $\pi \in \Pi$,
occuring in the formula are such that $m \leq l_i$ (since the parameter $u$
has been instantiated by 0) and thus, by Condition 2.b in Definition 25,
they must also be of the form $\xi[n_i + m']$, for some $i \in [2, k]$ and $m' \leq k_i$.
Consequently, by the induction hypothesis, there exists a refutation of this
set, and therefore there exists a derivation of $n \neq t_i \lor t_i \neq 0$ from $S_i$, i.e.,
(by Reflexivity) of $n \neq i$. But the formula $n \geq i \land n \neq i$ is obviously
equivalent to $n \geq (i + 1)$.

The inferences used to generated the sets $T_i$ do not involve any instantiation
of the variables (other than the variables $t_i$'s). By an easy induction of $i$, one
can thus show that for every $i \in \mathbb{N}$, the clauses occurring in $S_i$ are disjunctions
of clauses (with no variable sharing) occurring in $S$ or in $\mathcal{R}$ (up to a variable
renaming) and of literals of the form $\overline{f}(t_i)$ or $\neg \overline{f}(t_i)$. There exists finitely many
such clauses, up to a renaming of variables (and up to condensing).
Consequently, by the pigeonhole principle, there exist $i, j \in \mathbb{N}$ with $0 < i < j$
such that $S_i$ and $S_j$ are equivalent, modulo a renaming of the variable $t_i$ into $t_j$.
We now prove that, by using the induction rule, we can derive the following
set: \( \{ n \geq i + (j - i) \times v \} \cup \{ \{ n \not\equiv t_i + i + (j - i) \times v \} \otimes S_i \} \). Indeed, the base case is established by deriving \( T_i \) from \( S_0 \) as shown above. Moreover, \( \{ n \geq j \} \cup \{ \{ n \not\equiv t_j + j \} \otimes S_j \} \) can be derived from \( \{ n \geq i \} \cup \{ \{ n \not\equiv t_i + i \} \otimes S_i \} \).

By replacing the number \( i \) by \( i + (j - i) \times v \) in this derivation, we get a derivation from \( \{ n \geq i + (j - i) \times v \} \cup \{ \{ n \not\equiv t_i + i + (j - i) \times v \} \otimes S_i \} \) of \( \{ n \geq i + (j - i) \times v + (j - i) \} \cup \{ \{ n \not\equiv t_j + i + (j - i) \times v + (j - i) \} \otimes S_j \} \), i.e., of \( \{ n \geq i + (j - i) \times (v+1) \} \cup \{ \{ n \not\equiv t_j + i + (j - i) \times (v+1) \} \otimes S_j \} \). By definition of \( S_j \) and \( S_i \), this last expression is equivalent to \( \{ n \geq i + (j - i) \times (v+1) \} \cup \{ \{ n \not\equiv t_i + i + (j - i) \times (v+1) \} \otimes S_i \} \), up to the renaming \( t_i \mapsto t_j \).

In particular, we have thus derived the formula \( \phi \equiv n \geq i + (j - i) \times v \) that is clearly unsatisfiable in Presburger arithmetic (since \( v \) is an universal variable).

**Example 31** We provide an example of a weakly regular schema. This schema is propositional, but does not fall in the scope of the class of regular [2], nested-regular [3] or bound-linear [4] schemata.

\[
p(0) \land (\bigwedge_{i=0}^{n-1} p(i) \Rightarrow p(i+1)) \land (\bigwedge_{j=0}^{m-1} p(n+2,j) \Rightarrow p(n+2,j+2)) \Rightarrow p(n+2m)
\]

This schema is encoded by the term \( p(0) \land \hat{p}(n) \land \hat{q}(m) \Rightarrow p(n + m + m) \), associated with the rules:

\[
\begin{align*}
\hat{p}(0) \quad &\rightarrow \quad \text{true} \\
\hat{p}(u+1) \quad &\rightarrow \quad \hat{p}(u) \land (p(u) \Rightarrow p(u+1)) \\
\hat{q}(0) \quad &\rightarrow \quad \text{true} \\
\hat{q}(u+1) \quad &\rightarrow \quad \hat{q}(u) \land (p_{n+u+u} \Rightarrow p_{n+u+u+2})
\end{align*}
\]

It is compatible with the \( \pi \)-sequence of order 2 \(((n,\{p(n+u+u)\}),(n,\{p(u)\}))\).

**7 Application**

**7.1 Implementation**

The calculus described in Section 4 has been implemented into the proof editor SHRED (standing for Schematic resolution proof editor). The system is written in Prolog and is available at [http://membres-liga.imag.fr/peltier/shred.html](http://membres-liga.imag.fr/peltier/shred.html). For convenience, the program actually handles arithmetic expressions that are slightly more general than those in the previous theoretical definitions, with the additional condition that constraints must be added in the formulae to ensure that all expressions are interpreted as positive numbers (for instance, a term of the form \( n - 1 \) is allowed if \( n \geq 1 \) holds in the context, for instance if the term occurs in a clause containing the literal \( n \not\equiv 1 \)). The implementation also allows for a simplified formulation of schema.
Definition 32 An **iterated schemata** is an expression of the form $C^u=a..b.t$, where $t$ is a term of sort $s$, $u$ is an arithmetic variable, $a$ and $b$ are arithmetic expressions and $C$ is a **context**, i.e. a term of sort $s$ containing some occurrences of a special symbol $\diamond$ of sort $s$ (this symbol is not allowed to occur on the scope of a defined symbol).

Semantically, a formula of the form $D[C^u=a..b.t]$, is equivalent to $a + v \not\equiv b \lor C[\hat{f}(v,a,\vec{x})]$, where $\vec{x}$ is the vector of variables occurring in $C$ or $t$, $v$ is a fresh arithmetic variable, and $\hat{f}$ is a new defined symbol associated with the rules:

$$\begin{align*}
\hat{f}(0, \vec{x}) & \rightarrow t \\
\hat{f}(v + 1, u, \vec{x}) & \rightarrow C[\hat{f}(v, u + 1, \vec{x})/\diamond]
\end{align*}$$

If $C$ is of the form $\diamond \lor C'$, then $C^u=a..b.t$ is written $t \lor \bigvee_{u=a}^{b} C'$. Although strictly less general than the formalism considered in the present paper, the language of iterated schemata is sufficiently expressive to denote most schemata of clause sets occurring in practice. It is also more convenient and easier to use, because it is closer to the language used by mathematicians, and because the semantics of the schemata do not depend on "external" rewrite rules.

The system **Shred** is a proof assistant or a proof verification tool rather than a theorem prover. Indeed, while the inference rules can in principle be applied automatically (in a indeterministic way), in practice the search space is usually very large and some degree of human assistance is required. The system is essentially useful to help in the formalisation of schematic derivations and in the verification of their correctness.

The system takes as an input a list of commands (a short user manual is included in the distribution). These commands can be used to define new formulæ or terms, apply inference rules on existing objects in order to derive new assertions and finally to output the derivation attached to a given formula (in text or LaTeX format). Some features have been automatized in order to ease the writing of the proofs: step by step inferences are performed in a purely automatic way (the system computes the unifiers, checks the conditions on the extra-variables and performs the replacement as described in Section 3), arithmetic reasoning is handled automatically (e.g., arithmetic expressions are compared up to the usual arithmetic properties), and some normalisation steps are performed in an implicit way (such as the unfolding of inductive lemmata). The system has an interactive mode in which it asks the user when several results are possible (for instance if a given rule is applicable in several ways on two premisses) and an automatic mode in which the first found solution is selected.

7.2 The Fürstenberg problem

In [17], a rather strange proof of the infinite of primes is presented, which is based on purely topological arguments, which completely "hide" the construction of the prime numbers. In [9], a semi-automated analysis of this proof is performed: the cut-elimination method CERES (see, e.g., [10, 11]) is applied to get rid
of all topological lemmata, thus reconstructing Euclid's usual argument\(^3\). We applied the calculus defined in the present paper to formalize a crucial part of this analysis. The method CERES works by computing, from the considered proof \(\pi\), an unsatisfiable set of clauses \(S\) called the *characteristic set* of \(\pi\) and a set of proofs of the clauses occurring in \(S\), called the *projection proofs*. A resolution refutation of \(S\) can then be constructed by using any resolution-based theorem-prover and a cut-free proof of the original statement can be obtained by combining this resolution derivation with the projection proofs. However, Fürstenberg's proof is an inductive one, hence it cannot be directly formalized in first-order logic; instead it is formalized [9] as a *family of first-order proofs*, parameterized by an natural number that can be interpreted as the (assumed to be finite) number of prime numbers. In [9], the method CERES has been applied on this proof schema, yielding a schema of clause sets \(S_n\), and a schema of refutations of \(S_n\) has been provided. Both schemata have been constructed by hand. In this section, we provide a formal definition of a refutation of \(S_n\), which has been constructed and verified using the system SHRED. It is a first step toward a the full formalisation of the results in [9] (the other step would be to check that the extraction of the schema of characteristic sets is correct, which is of course a very complex problem that is outside the scope of the present paper, see, e.g., [27, 15, 16])

The derivation below has been depicted automatically by the system in \(\LaTeX\) format, comments have been added manually. It is essentially the same as the refutation provided in [9], although some inference steps have been switched in order to avoid having to consider schemata of terms containing an unbounded number of (indexed) variables. Note that the proof has *not* been constructed automatically: the system is used only to *formalize* and *verify* the sequence of inferences, not to *discover* it. The automatic construction of the proof is a very complex problem: even relatively small instances (e.g. for 3, 4, ... prime numbers) of the considered schema are out of the scope of state-of-the-art resolution and superposition-based provers as such E [28] or Prover9 [23]. It is thus not realistic to expect that a proof can be obtained in a purely automatic way for the parameterized version of the problem. We keep the same notations as in [9]: \(<\>\) is the usual ordering on natural numbers, \(p(i)\) denotes the \(i\)-th prime number (with \(i = 1, \ldots, n\)), \(ts(n) = \prod_{i=1}^n p(i)\) denotes the product of the \(n\) prime numbers formally defined by the iterated schema \((\circ \times p(i))^{i=1,\ldots,n}\), \(s_1(x)\) can be interpreted as the least divisor of \(x\), \(s_4(x)\) is \(\frac{x}{ts(x)}\). Variables are written in Prolog style, using capital letters.

% First step: proof of 0 < ts(r) and 0 < ts(r)+1

\[
\begin{align*}
\text{zero} & : 0 < 1 & \text{(hyp.)} \\
\text{ord} & : 0 \not\in Y - X \lor 0 < (Y - X) + 1 & \text{(hyp.)} \\
p_1 & : 0 < p(Y) - 1 & \text{(hyp.)} \\
p_0 & : 0 < p(Y) & \text{(res, ord, p\(_1\))}
\end{align*}
\]

\(^3\)Assuming that \(p_1, \ldots, p_0\) are the only prime numbers, \(p_1 \times \cdots \times p_0 + 1\) must be prime and distinct from \(p_1, \ldots, p_0\), a contradiction.
$0 \not< (s - r) + 1$  
$0 < ts(s)$  
$0 \not< X \lor 0 \not< Y \lor 0 < X \times Y$  
$0 \not< Y \lor 0 < ts(s) \times Y$  
$x < ts(s) \times p(Y)$  
$x < (o \times p(k))^{k=1 \ldots s} \times p(0)$  
$0 < ts(r)$  
$0 \not< Y \lor 0 < ts(r) \times Y$  
$0 < ts(r) \times 1$

% Second step: proof of
$0 < p(0) \times J \Rightarrow \bigvee_{j=1}^{r}(p(j) \times s_4(1 + (p(0) \times J))) = 1 + (p(0) \times J)$

$0 < -r$  
$X \times Y = Y \times X$  
$0 + X = X$  
$0 \not< -L - K \lor 0 \not< M - K \lor 0 \not< M - L \lor K + (I \times M) \neq (L + (J \times M))$  
$0 \not< L \lor 0 \not< -M \lor 0 \not< L \lor M \neq (I \times M)$  
$M_0 = 1 \lor \bigvee_{j=0}^{r}(s_1(M_0) = p(j))$  
$l \neq K - 1 \lor K \neq 1$  
$M \lor -l \lor \bigvee_{j=0}^{r}(p(j) \times s_4(K) = K)$  
$0 \not< L + (J \times p(0)) - 1 \times 0 < -r \lor 0 \not< L \lor 0 \not< p(0) \lor 0 \not< -L + p(0) \lor \bigvee_{j=0}^{r}(p(j) \times s_4(L + (J \times p(0)))) = L + (J \times p(0))$  
$0 \not< J \times p(0) \lor 0 < -r \lor 0 \not< p(0) - 1 \times \bigvee_{j=1}^{r}(p(j) \times s_4(1 + (J \times p(0)))) = 1 + (J \times p(0))$  
$0 \not< J \times p(0) \lor 0 < -r \lor p(0) \lor \bigvee_{j=1}^{r}(p(j) \times s_4(1 + (J \times p(0)))) = 1 + (J \times p(0))$  
$0 \not< p(0) \times J \lor \bigvee_{j=1}^{r}(p(j) \times s_4(1 + (p(0) \times J))) = 1 + (p(0) \times J)$

% Third step: induction proof of
$s \leq r \Rightarrow \bigvee_{j=0}^{s}(p(j) \times s_4(1 + (ts(s) \times X))) = 1 + (ts(s) \times X)$

$0 \not< (s - r) + 1$  
$0 \not< ts(s) \times X \lor \bigvee_{j=0}^{s}(p(j) \times s_4(1 + (ts(s) \times X))) = 1 + (ts(s) \times X)$  
$X \times (Y \times Z) = (X \times Y) \times Z$
We have presented a calculus for reasoning on infinite sequences of terms or formulae parameterized by natural numbers. The calculus is based on an extension of the resolution and paramodulation rules. Its main feature is that inferences can be applied on atoms or terms occurring at arbitrary deep posi-
tions inside the considered formulae, which allows to factorize many (possibly infinitely many) reasoning steps. The calculus is not complete, since the considered logic is not even semi-decidable, but some complete and/or decidable classes have been identified. As an example of application, we have used this calculus to formalize a refutation of a schema of clause sets introduced in [9] for eliminating the cuts in Fürstenberg’s proof of the infinite of primes.

Several lines of future work deserve to be considered. From a theoretical point of view, it would be interesting to identify other classes of formulae for which the calculus is refutationally complete, in particular formulae whose proofs involve nested induction schemes. A well-known example of schema of propositional formulae is the pigeonhole problem (encoding the fact that for every \( n \in \mathbb{N} \), there is no injective function from \([1, n]\) into \([1, n - 1]\), which plays a central role for the complexity analysis of the resolution method [19]. This problem does not fall in the scope of the class provided in the paper; it would thus be interesting to find a extension of this class which contains such a schema. Extension of the calculus to families of formulae defined on more complex algebraic structures (such as the trees or the lists) can also be considered.

A current limitation of the calculus is that it does not handle indexed variables, such as those in the schema: \( \bigwedge_{i=1}^{n} x_i \approx x_{i+1} \Rightarrow x_1 \approx x_{n+1} \), or unbounded quantifier alternations, such as \( \forall x_1 \exists y_1 \ldots \forall x_n \exists x_{n+1} \phi \). Such variables can be viewed as second-order variables of type \( \text{nat} \to s \). Adding second-order variables to the language is straightforward, but then the unification problem becomes undecidable [18], which entails that some unifiers would have to be provided by the user.

References


