Combining Induction and Saturation-Based Theorem Proving

M. Echenim, Nicolas Peltier

Noname manuscript No.
(will be inserted by the editor)

Combining Induction and Saturation-based Theorem Proving

M. Echenim and N. Peltier

the date of receipt and acceptance should be inserted later

Keywords Inductive Theorem Proving, Saturation-based Theorem Proving

Mathematics Subject Classification (2000) 03B35 · 68T15

CR Subject Classification F.3.1 · F.4.1 · I.2.3

Abstract A method is devised to integrate reasoning by mathematical induction into saturation-based proof procedures based on Resolution or Superposition. The obtained calculi are capable of handling formulas in which some of the quantified variables range over inductively defined domains (which, as is well-known, cannot be expressed in first-order logic). The procedure is defined as a set of inference rules that generate inductive invariants incrementally and prove their validity. Although the considered logic itself is incomplete, it is shown that the invariant generation rules are complete, in the sense that if an invariant (of some specific form) is deducible from the considered clauses, then it is eventually generated.

1 Introduction

The most successful first-order theorem provers nowadays are based on saturation. Such procedures work by refutation: they generate consequences of the initial set of axioms by applying a set of inference rules, usually the Resolution [25] or/and Superposition [2, 27] calculi, until a contradiction is detected or a saturated set is obtained, from which no new (non-redundant) consequences can be derived. Saturation proof procedures based on Resolution or Superposition are sound and complete for first-order logic and form the basis of the most efficient available theorem provers [26, 36, 32, 34]. However they are not able to handle theorems relying on mathematical induction. In the present paper, we extend saturation-based proof procedures to make them capable of refuting inductive properties. Our method is generic in the sense that no assumption is made on the initial proof calculus, except for some basic standard properties such as soundness. This new approach is based on a combination of several features. A constrained proof procedure is used, in which the terms on which induction is performed are abstracted away, while standard inference rules are used to handle the remaining part of the formula.

Univ. Grenoble Alpes, CNRS, LIG, F-38000 Grenoble France

Candidate inductive lemmata are introduced by a controlled application of Tseitin's extension rule, consisting in the introduction of new atoms $\mathbf{T}_\alpha(x_1, \ldots, x_n)$ that encode a formula α with free variables $x_1, \ldots, x_n$. Rules are devised to derive universal formulas when the corresponding inductive scheme has been proven and to generate inductive lemmata automatically and incrementally, exploiting failures of previous attempts.

The principle of our approach is the following. Given a set of clauses S containing a term t to be interpreted on an inductive domain, the goal is to automatically generate an inductive consequence $\forall x\, \alpha(x)$ of S such that the empty clause can be derived from $S \cup \forall x\, \alpha(x)$ with no use of induction, i.e., by a standard inference system that is complete for refutation. For the sake of simplicity, we assume in this example that α contains only one free variable to be interpreted on an inductive domain, and that this inductive domain admits a simple base case a. The goal is to prove that $\alpha(t)$ holds for all ground terms t on the inductive domain. The generation of the inductive invariant is performed as follows. By setting t to be equal to the base case a, a standard refutation of $S \cup \{t \simeq a\}$ is constructed. An analysis of this refutation permits to extract a formula $\alpha(x)$, such that $\neg\alpha(a)$ was used to derive the refutation of $S \cup \{t \simeq a\}$. The formula $\alpha(x)$ is taken as a candidate inductive invariant, and the goal is to prove that $\forall x\, \alpha(x)$ is an inductive consequence of S, so that S is (inductively) unsatisfiable. The verification that α is indeed an inductive invariant is done by infinite descent, i.e., by showing that if $\alpha(t)$ does not hold, then there exists a proper subterm s of t such that $\alpha(s)$ does not hold. If this property cannot be proved, then the candidate inductive invariant is refined and the infinite descent principle is applied again.

Consider for example the following clause set S:

$$\{\neg p(x) \vee p(f(x)), p(a), \neg p(b)\}$$

It is clear that S is satisfiable. However, if we assume that b is interpreted as a ground term of the form $f^n(a)$ for some $n \in \mathbb{N}$ (i.e., that b belongs to the set defined by the constructors a and f), then it is easy to check by induction on n that S is unsatisfiable. If the Resolution calculus (see, e.g., [25,3]) is employed to test the satisfiability of the above clause set then no clause will be derived[1]. If the domain axiom $\forall x\, (x \simeq 0 \vee \exists y\, x \simeq f(y))$ is added, then the only derivable clauses are: $x \simeq 0 \vee x \simeq f(g(x))$ (by Skolemization) and $x \simeq 0 \vee \neg p(g(x)) \vee p(x)$ (by Paramodulation [27] between with the previous clause and the first one). In our framework, the clause $\neg p(b)$ will be encoded as a constrained clause $[\![\neg p(x) \mid b \simeq x]\!]$, meaning that $\neg p(x)$ holds if $x = b$. This technique allows to perform inferences on $\neg p(b)$ without having to replace b by a constructor term, which would require to blindly apply paramodulation inferences into b. By combining the previous constrained clause with $p(a)$ and $\neg p(x) \vee p(f(x))$ it is easy to derive the constrained clauses $[\![\Box \mid b \simeq a]\!]$ — meaning that S is unsatisfiable if $b = a$. Since the only clause involving b that is used to derive this assertion is $[\![\neg p(x) \mid x \simeq b]\!]$, this suggests that this constrained clause represents the negation of an instance of the inductive invariant; here, the candidate inductive invariant is therefore $\forall x\, p(x)$. Using the clauses $\neg p(x) \vee p(f(x))$ and $[\![\neg p(x) \mid b \simeq x]\!]$, we then generate the clause $[\![\neg p(x) \mid b \simeq f(x)]\!]$ — meaning that $\neg p(x)$ is a logical consequence of S if $b = f(x)$. From this property we deduce that $S \models \exists b'\, b' < b \wedge \neg p(b')$, where $<$ denotes the strict subterm ordering. This entails that $\forall y(\neg p(y) \Rightarrow \exists y'\, y' < y \wedge \neg p(y'))$ is a logical consequence of $\{p(a), \neg p(x) \vee p(f(x))\}$ (the only clause involving b is asserted as a

[1] Provided usual ordering restrictions are used, blocking any inference involving the first literal of the first clause since it is not maximal.

hypothesis in the implication so that the term b can be abstracted away and replaced by a universal variable y). By infinite descent, this means that $\{p(a), \neg p(x) \vee p(f(x))\} \models \forall x\, p(x)$, which yields a contradiction with $\neg p(b)$.

In this case, the inductive invariant is easy to obtain and each inductive case is proven by a single inference step. However, in general the property $p(b)$ is not a mere atom, but rather a set of clauses (b may occur in several literals or clauses), and the proofs of the base and inductive cases may be much more complex. Moreover, these clauses possibly contain variables other than the variable x denoting b. The goal is then to determine for which instances of these variables the invariant propagates.

The invariant does not necessarily correspond to the entire set of clauses containing b either, and may contain clauses not occurring in the initial clause set. Consider for instance the following clause set (still on the same inductive domain) that can be handled by our approach:

$$\{\neg p(x, c) \vee p(f(x), c), \neg q(x) \vee q(f(x)), p(a, y) \vee q(a), \neg q(x) \vee r(x), \neg p(b, y), \neg r(b)\}.$$

Here induction can only be applied by considering the set $\{\neg p(b, y), \neg q(b)\}$, where $\neg q(b)$ is derived from $\neg r(b)$ and $\neg q(x) \vee r(x)$ (the clause $\neg r(b)$ must be excluded since it does not propagate), together with the instantiation $y \leftarrow c$. It is clear that blindly enumerating all possible candidate invariants and substitutions is impractical and leaving the burden to the user is also unsatisfactory. We thus devise a calculus that is able to extract such candidate invariants and prove them automatically. The intuitive idea is to compute candidate invariants by collecting minimal sets of hypotheses that are sufficient to prove the base case, and then to exploit failures of previous proof attempts to refine the considered invariant. If, when trying to prove that an invariant propagates, it is discovered that an additional hypothesis is needed, then this hypothesis is added to the current candidate invariant.

Previous and Related Work

The automation of inductive reasoning has been thoroughly investigated in the context of rewriting (see, e.g., [9,7,8,33,12]). The goal is to prove universal queries of the form $\forall x_1, \ldots, x_n\, \phi$, where ϕ is a quantifier-free formula (usually an equation or a disjunction of equations) containing functions defined by rewrite rules and $x_1, \ldots, x_n$ range over the set of ground terms. Heuristics have been proposed to guide the application of the rewriting rules so that the induction hypothesis can be used and techniques have been devised to generate inductive lemmata, for instance by strengthening the initial conjecture or by using enumeration-based algorithms (see, e.g., [13]). The drawback of the rewriting approach is that it does not handle arbitrary first-order formulas (with quantifier alternation). It is worthwhile mentioning that non-validity is usually semi-decidable in the context of rewriting for canonical rewrite systems. Indeed, it is possible to enumerate all the ground instances of the theorem and to test them for validity, provided the rewrite system is terminating. If the theorem is not (inductively) valid then it admits at least one non-valid ground instance, which can be generated in finite time, and this is not the case in first-order logic: combining inductive reasoning and first-order logic makes unsatisfiability testing both non-semi-decidable and non-co-semi-decidable. Inductionless induction [14] reduces inductive validity to a first-order satisfiability test using proof by consistency: a formula F is proven if it is consistent with

a set of axioms. This approach relies on the existence of an axiomatization describing the set of possible counter-examples and is again restricted to purely universal queries.

Inductive reasoning can be performed by using any first-order prover to prove induction schemes. The problem is then how to construct the relevant inductive invariant. For instance, in [18] a technique is proposed to extract inductive invariants from instances of the considered theorems. Related approaches have been considered in the context of Satisfiability Modulo Theories [30]. Although originally designed for handling quantifier-free formulas, SMT solvers are often able to prove first-order formulas by combining quantifier-instantiation with ground decision procedures. In this context, inductive properties can be established by proving inductive schemes: when trying to prove a formula $\forall x\, \phi(x)$, it is possible to assume that $\phi(y)$ is true for every $y < x$. The focus is again on the automated discovery of inductive lemmata, which can been done by enumerating candidate invariants, using filtering techniques to discard irrelevant or falsified candidates. These techniques have been successfully integrated into the theorem prover CVC4 [5]. SMT solving has also been combined with the rewriting approach [19].

Inductive reasoning can also be encoded as cyclic proofs [10]. In this approach, the standard sequent calculus is extended to handle cycles in proof search, i.e., proofs in which the goal itself is used as an axiom. Additional criteria are provided to ensure that soundness is preserved, for instance by showing that some well-founded measure is strictly decreasing along each cycle. This technique is very powerful and general, allowing for example nested or mutually recursive proofs and arbitrary quantifier alternation. However, it does not apply to saturation-based procedures which lack the tree structure of sequent or tableaux calculi. Our method can be seen as a way to generate cycles from a given inference graph. In our context, the cycle must be defined at the level of formulas or clause sets, whereas the inference graph is defined on clauses.

Our procedure is based on the use of a constrained calculus: some of the terms are abstracted away and replaced by variables, while equations are attached as constraints to the clause to specify the values of these variables. This technique is widely used when reasoning modulo theories in the Superposition framework, see for instance the work on hierarchic Superposition calculi [4,6,21].

There have been a few attempts to combine saturation-based procedures with inductive theorem proving. In [21], a Superposition calculus is presented for reasoning on formulas containing constant symbols interpreted over some fixed domain and defined inductively using a set of constructors. As in our approach, such constants are abstracted away and replaced by variables so that they can be instantiated by unification, instead of having to instantiate them explicitly by using domain axioms. An inductive rule is defined, encoding the fact that, when trying to refute a formula $\phi(x)$ where x is interpreted over an inductive domain, one may assume that x is the minimal element such that $\phi(x)$ holds (w.r.t. some well-founded ordering). This entails that $\phi(y)$ is false for all y that are smaller than x. The inductive rule is able to derive instances of this property for the usual subterm ordering. Note however that this inductive scheme only applies to the initial formula, not on auxiliary inductive lemmata: indeed, the rule is clearly unsound if applied on a subformula ϕ that does not contain all occurrences of x. In [20], this calculus is enriched by a technique that is meant to ensure termination in some cases. The idea is to compute substitution expressions (a language constructed by iterations, compositions and disjunctions of substitutions, related to regular expressions and finite automata) representing the set of values of x such that $\phi(x)$ can be refuted. Then the unsatisfiability problem can be solved by de-

tecting that this set is equivalent to the entire domain. Strong conditions are required to ensure termination: the function symbols must be at most unary, the positive literals cannot contain multiple occurrences of the same variable and the Superposition calculus must terminate.

The calculi defined in [1] and [24] are based on similar ideas, but use slightly different approaches for handling inductive reasoning. In [1] the idea is to detect loops in the search space, e.g., if a formula $\phi(y)$ is derived from $\phi(y+k)$, and if there exists an x such that $\phi(x)$ holds (x, y denote natural numbers), then one can assume that x is strictly smaller than k. The addition of the condition $x < k$ thus preserves satisfiability, and if equations of the form $x \neq 0, \ldots, x \neq k-1$ are derived then the unsatisfiability of the initial set is established. [24] uses a similar loop detection rule based on infinite descent: if a formula $\phi(t)$ with $t < x$ is derived from $\phi(x)$ then necessarily $\phi(x)$ is unsatisfiable. In [23] algorithms are provided to explicitly compute the formula ϕ from a given inference graph. This approach is restricted to inductive domains defined by using monadic constructors only. This limitation is overcome by the present work.

In [16,17], a technique is presented to perform inductive reasoning in Superposition calculi. In this approach, inductive invariants are constructed as conjunctions of elementary formulas (called "clause contexts") extracted from the clause set (or submitted by a user), and conditions ensuring that the invariant propagates are encoded as a quantified boolean problem. This approach shares some similarity with ours, mainly the introduction of (albeit purely propositional in this approach) atoms, written $[\![\phi]\!]$, to denote complex formulas ϕ. However it strongly relies on the Avatar architecture [35], where a Superposition theorem prover is tightly integrated with a SAT-solver. Furthermore, it does not handle nested induction or mutually recursive data-structures.

As far as we are aware, the approach described in the present paper is the only one that can reason by induction on complex terms, possibly containing variables (see Example 67) in the context of saturation-based theorem proving. This allows one to prove formulas with arbitrary quantifier alternation, e.g., $\exists x \forall y \exists z\, \phi(x, y, z)$, where y is interpreted in an inductive domain; after negation and skolemization, we get a formula $\forall x, z\, \neg\phi(x, f(x), z)$, and the proof must be done by induction on $f(x)$. Other approaches are restricted to formulas of the form $\forall^* \exists^* \phi$.

Structure of the Paper

In Section 2 we briefly review usual definitions and define the syntax and semantics of the logic we are considering. A calculus for inductive reasoning is described in Section 3, and examples of applications are provided in Section 5. Section 6 contains some restricted completeness results and Section 7 concludes the paper and provides some lines of future work.

2 Preliminaries

2.1 Basic Definitions

We first briefly review and adapt standard definitions. We adopt a multi-sorted framework. The symbols $\mathtt{S}$ and Σ respectively denote the sets of sort and function symbols (with $\mathtt{bool} \in \mathtt{S}$), and each element in Σ is associated with a unique *profile* of the form

$\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$, such that $\mathtt{s}_1, \ldots, \mathtt{s}_n, \mathtt{s} \in \mathtt{S}$, $n \geq 0$ and $\forall i \in [1, n], \mathtt{s}_i \neq \mathtt{bool}$. We write $f : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ to state that f has profile $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$. The *arity* of a symbol f of profile $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ is n. A *predicate symbol* is a function of profile $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{bool}$.

The set of *terms* is built inductively from Σ and a set of variables $\mathcal{V}$. Each variable in $\mathcal{V}$ is mapped to a unique sort and the sort of a term is defined as usual. Given $\Sigma' \subseteq \Sigma$, a *Σ'-term* is a term built from Σ' and $\mathcal{V}$.

The set of *first-order formulas* is built using the set of logical symbols $\vee, \wedge, \neg, \exists, \forall$, from *atoms* of one of the following forms:

- $p(t_1, \ldots, t_n)$, where p is a predicate symbol of profile $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{bool}$, and $t_1, \ldots, t_n$ are terms of sort $\mathtt{s}_1, \ldots, \mathtt{s}_n$ respectively;
- $t \simeq s$ where t and s are terms of the same sort;
- $t \prec s$, where t and s are arbitrary terms, possibly of different sorts.

As we shall see, $\simeq$ is interpreted as the usual equality predicate and $\prec$ as the subterm ordering (the precise interpretations will be defined afterwards). We denote by $\mathrm{var}(e)$ the set of variables freely occurring in the expression (term or formula) e. The formulas $\phi \Rightarrow \psi$ and $\phi \Leftrightarrow \psi$ are taken as shorthands for $\neg\phi \vee \psi$ and $(\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)$ respectively.

The notion of a *position in an expression (term or formula) or sequence of expressions* is defined as usual. The set of positions in an expression e is written $\mathrm{pos}(e)$. If e is an expression, then we denote by $e|_p$ the expression occurring at position p in e and by $e[e']_p$ the expression obtained from e by replacing the expression at position p by e' (implicitly assuming that the sorts of e' and $e|_p$ are identical). The head symbol of the expression e is written $\mathit{head}(e)$; note that $\mathit{head}(e)$ may be a function symbol, a variable or a logical symbol.

A substitution is a total function mapping each variable to a term of the same sort. The image of a term t by a substitution σ is defined inductively and written $t\sigma$. The *domain* of a substitution is the set of variables x such that $x\sigma \neq x$. A substitution σ is a *renaming* if $\forall x \in \mathcal{V}$, $x\sigma \in \mathcal{V}$ and $\forall x, y \in \mathrm{dom}(\sigma)$, $x\sigma = y\sigma \Rightarrow x = y$. We denote by $[t_1/x_1, \ldots, t_n/x_n]$ the substitution mapping x_i to t_i for every $i \in [1, n]$. The notation $t[v/u]$ is extended to the case where u is an arbitrary term, to denote the term obtained from t by replacing every occurrence of u in t by v. A *unifier* of two terms t and s of the same sort is a substitution σ such that $t\sigma = s\sigma$. It is well-known that every unifiable pair of terms admits a most-general unifier (unique up to a renaming).

An *interpretation* is a function $\mathcal{I}$ mapping:

- Each sort symbol $\mathtt{s} \in \mathtt{S}$ to a non-empty set, such that $\mathcal{I}(\mathtt{bool}) = \{\mathtt{true}, \mathtt{false}\}$ and the sets $\mathcal{I}(\mathtt{s})$ are pairwise disjoint.
- Each function symbol f of profile $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ to a function $f^{\mathcal{I}}$ from $\mathcal{I}(\mathtt{s}_1) \times \cdots \times \mathcal{I}(\mathtt{s}_n)$ to $\mathcal{I}(\mathtt{s})$.
- The symbol $\prec$ to a binary relation on the domain $\bigcup_{\mathtt{s} \in \mathtt{S}} \mathcal{I}(\mathtt{s})$.
- Each variable x of sort $\mathtt{s}$ to an element of $\mathcal{I}(\mathtt{s})$.

If $\mathcal{I}$ is an interpretation, $x_1, \ldots, x_n$ are variables of respective sorts $\mathtt{s}_1, \ldots, \mathtt{s}_n$ and $e_1, \ldots, e_n$ are elements of $\mathcal{I}(\mathtt{s}_1), \ldots, \mathcal{I}(\mathtt{s}_n)$, then $\mathcal{I}[e_1/x_1, \ldots, e_n/x_n]$ denotes the interpretation coinciding with $\mathcal{I}$, except that $\mathcal{I}[e_1/x_1, \ldots, e_n/x_n](x_i) \stackrel{\mathrm{def}}{=} e_i$.

The image of a term of sort $\mathtt{s}$ (resp. of a formula) by an interpretation is an element of $\mathcal{I}(\mathtt{s})$ (resp. of $\{\mathtt{true}, \mathtt{false}\}$), defined as usual (where x is a variable of sort $\mathtt{s}$):

$$\begin{array}{rcl}
\mathcal{I}(f(t_1,\ldots,t_n)) & \stackrel{\text{def}}{=} & f^{\mathcal{I}}(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n)) \\
\mathcal{I}(t \prec s) & \stackrel{\text{def}}{=} & \mathtt{true} \text{ iff } (\mathcal{I}(t),\mathcal{I}(s)) \in \mathcal{I}(\prec); \\
\mathcal{I}(\neg\phi) & \stackrel{\text{def}}{=} & \mathtt{true} \text{ iff } \mathcal{I}(\phi) = \mathtt{false}; \\
\mathcal{I}(\phi_1 \vee \phi_2) & \stackrel{\text{def}}{=} & \mathtt{true} \text{ iff } \mathcal{I}(\phi_1) = \mathtt{true} \text{ or } \mathcal{I}(\phi_2) = \mathtt{true}; \\
\mathcal{I}(\phi_1 \wedge \phi_2) & \stackrel{\text{def}}{=} & \mathtt{true} \text{ iff } \mathcal{I}(\phi_1) = \mathtt{true} \text{ and } \mathcal{I}(\phi_2) = \mathtt{true}; \\
\mathcal{I}(t \simeq s) & \stackrel{\text{def}}{=} & \mathtt{true} \text{ iff } \mathcal{I}(t) = \mathcal{I}(s); \\
\mathcal{I}(\exists x\, \phi) & \stackrel{\text{def}}{=} & \mathtt{true} \text{ iff there exists an } e \in \mathcal{I}(\mathtt{s}) \text{ such that } \mathcal{I}[e/x] \models \phi; \\
\mathcal{I}(\forall x\, \phi) & \stackrel{\text{def}}{=} & \mathtt{true} \text{ iff for every } e \in \mathcal{I}(\mathtt{s}),\ \mathcal{I}[e/x] \models \phi.
\end{array}$$

We write $\mathcal{I} \models \phi$ if $\mathcal{I}(\phi) = \mathtt{true}$ and $\phi \models \psi$ if for every interpretation $\mathcal{I}$ such that $\mathcal{I} \models \phi$, we have $\mathcal{I} \models \psi$. Note that for technical convenience and contrarily to the usual practice, we do not assume at this point that free variables are universally quantified; for instance $p(x) \not\models p(y)$, but $\forall x\, p(x) \models \forall y\, p(y)$. Given a formula ϕ such that $\mathrm{var}(\phi) = \{x_1,\ldots,x_n\}$, we denote by $\forall^*\phi$ the formula $\forall x_1,\ldots,x_n\, \phi$.

A *literal* is an atom or the negation of an atom and a *clause* is a finite disjunction of literals. The empty clause is denoted by $\Box$. Clauses are taken as multisets, e.g., we may write $l \in C$ to state that l is a literal occurring in C, or $C = D$ (resp. $C \subseteq D$) if C and D are identical modulo AC (resp. if $D = C \vee C'$).

2.2 Inductive Domains

In order to enable inductive reasoning in our framework, we consider a distinguished set of sort symbols $\mathtt{I} \subseteq \mathtt{S}$, which are interpreted on inductively defined domains; these are domains that are specified by a set of constructors. The sorts in $\mathtt{I}$ are called *inductive sorts*. We thus consider several kinds of function symbols.

- We denote by K the set of constructors, i.e., the symbols defining the domains of inductive sorts. We assume that the range of each symbol of K is in $\mathtt{I}$. For instance, the natural numbers have two constructors $0 : \mathtt{nat}$ and $\mathit{succ} : \mathtt{nat} \to \mathtt{nat}$, lists have two constructors $\mathit{nil} : \mathtt{list}$ and $\mathit{cons} : \mathtt{s} \times \mathtt{list} \to \mathtt{list}$, etc. Note that the arguments of the constructors may be of sorts not necessarily in $\mathtt{I}$; for example, the elements occurring in lists may be of some arbitrary non-inductive sort.
- We denote by Λ the set of function symbols other than constructors that range over an inductive sort. For example, a constant symbol distinct from 0 denoting an unspecified natural number n (to be interpreted as a term of the form $\mathit{succ}^k(0)$, with $k \in \mathbb{N}$) is in Λ, as well as the usual functions $+ : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}$ and $\mathit{append} : \mathtt{list} \times \mathtt{list} \to \mathtt{list}$.
- Each constructor f of profile $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ is associated with n *selectors* f_i^{-1} $(1 \leq i \leq n)$ of profiles $\mathtt{s} \to \mathtt{s}_i$. We denote by $\mathcal{R}$ the set of rules of the form $f_i^{-1}(f(x_1,\ldots,x_n)) \to x_i$ and by $t{\downarrow_{\mathcal{R}}}$ the normal form of any term t by this set of rules.
- The remaining symbols are standard function symbols. Their profiles are of the form $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ where $\mathtt{s}_i \in \mathtt{S} \setminus \{\mathtt{bool}\}$ and $\mathtt{s} \in \mathtt{S} \setminus \mathtt{I}$.

Remark 1 The selector functions are introduced for technical convenience only. They are not intended to occur in the initial clause sets and they will be used only for proving the soundness of the `Domain Decomposition` rule (see Section 3.5). See Examples 28, 47 and Definition 58 for more details.

Given an interpretation $\mathcal{I}$, the set of $\mathcal{I}$-*terms* of sort $\mathtt{s} \in \mathtt{S}$ is the least set containing $\mathcal{I}(\mathtt{s})$ and every expression of the form $f(t_1, \ldots, t_n)$ where f is a constructor of profile $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ and t_i is an $\mathcal{I}$-term of sort $\mathtt{s}_i$, for $i \in [1, n]$. Note that if $\mathtt{s} \notin \mathtt{I}$ then the only $\mathcal{I}$-terms of sort $\mathtt{s}$ are the elements of $\mathcal{I}(\mathtt{s})$. Informally, an $\mathcal{I}$-term is simply a term built on the set of constructors and on elements of the domain of $\mathcal{I}$ (taken as constants). These $\mathcal{I}$-terms are interpreted as usual terms, with $\mathcal{I}(e) \stackrel{\text{def}}{=} e$ if $e \in \mathcal{I}(\mathtt{s})$ for $\mathtt{s} \in \mathtt{S}$. The *subterm ordering* $\lhd$ is defined inductively on $\mathcal{I}$-terms as follows: $t \lhd s$ if s is a term of the form $f(s_1, \ldots, s_n)$, with $f \in K$ and there exists $i \in [1, n]$ such that either $s_i = t$ or $t \lhd s_i$. We write $t \trianglelefteq s$ if $t \lhd s$ or $t = s$.

Example 2 Let $\mathtt{S} = \{\mathtt{list}, \mathtt{real}\}$, with $\mathtt{I} = \{\mathtt{list}\}$. Consider the formula: $\forall x, y\ \mathit{sum}(\mathit{cons}(x, y)) \simeq x + \mathit{sum}(y) \wedge \mathit{sum}(\mathit{nil}) \simeq 0 \wedge \mathit{sum}(l) \simeq 0$. The symbols *nil* and *cons* are constructors of profile $\to \mathtt{list}$ and $\mathtt{real} \times \mathtt{list} \to \mathtt{list}$ respectively; 0, + and *sum* are ordinary function symbols of profile $\to \mathtt{real}$; $\mathtt{real} \times \mathtt{real} \to \mathtt{real}$ and $\mathtt{list} \to \mathtt{real}$ respectively; l is a constant symbol of sort `list` occurring in Λ and x and y are variables of sorts `real` and `list` respectively.

If $\mathcal{I}$ interprets `real` as $\mathbb{R}$, then the $\mathcal{I}$-terms of sort `list` include all terms of the form $\mathit{cons}(r_1, \mathit{cons}(\ldots, \mathit{cons}(r_n, \mathit{nil})\ldots))$ with $n \geq 0$ and $r_1 \ldots, r_n \in \mathbb{R}$, and the $\mathcal{I}$-terms of sort `real` are real numbers.

Definition 3 A term is `I`-*ground* if it contains no variable of a sort in `I` and if all its function symbols are constructors. A substitution σ is `I`-*ground* if $x\sigma$ is `I`-ground, for every variable $x \in \mathrm{dom}(\sigma)$.

A *rooted formula* is an expression of the form $\phi[x_1, \ldots, x_n]$, where ϕ is a formula and $\mathrm{var}(\phi) = \{x_1, \ldots, x_n\}$. A rooted formula is interpreted in the same way as a usual formula, the only difference is that it fixes a total strict ordering on its free variables. If $\alpha = \phi[x_1, \ldots, x_n]$, then we denote by $\alpha(t_1, \ldots, t_n)$ the formula $\phi[t_1/x_1, \ldots, t_n/x_n]$. For simplicity, we may denote a rooted formula $\phi[x, y_1, \ldots, y_n]$ by $\phi[x, \mathbf{y}]$.

Let $\alpha = \phi[x, y_1, \ldots, y_n]$ be a rooted formula, where $x, y_1, \ldots, y_n$ are of sorts $\mathtt{s}, \mathtt{s}_1, \ldots, \mathtt{s}_n$ respectively. We associate α with two $(n+1)$-ary predicate symbols $\mathbf{T}_\alpha$ and $\mathbf{T}_\alpha^\prec$, both of profile $\mathtt{s} \times \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{bool}$. We assume that all the symbols of the forms $\mathbf{T}_\alpha$ and $\mathbf{T}_\alpha^\prec$ are pairwise distinct. The interpretation of these special symbols will be given in Definition 6: intuitively, $\mathbf{T}_\alpha(x, y_1, \ldots, y_n)$ holds if $\alpha(x, y_1, \ldots, y_n)$ holds and $\mathbf{T}_\alpha^\prec(x, y_1, \ldots, y_n)$ holds if $\alpha(x', y_1, \ldots, y_n)$ holds for every proper subterm x' of x.

Remark 4 The special symbols $\mathbf{T}_\alpha$ and $\mathbf{T}_\alpha^\prec$ and the ordering relation $\prec$ are not intended to occur in the initial clause set: they are introduced later on during proof search.

Example 5 The formula $\phi = (p(x, y) \vee p(y, x)) \wedge \neg q(x)$ may be associated with two rooted formulas $\alpha = \phi[x, y]$ and $\beta = \phi[y, x]$, depending on how the variables are ordered. Then $\alpha[a, b]$ and $\beta[a, b]$ denote the formulas $(p(a, b) \vee p(b, a)) \wedge \neg q(a)$ and $(p(b, a) \vee p(a, b)) \wedge \neg q(b)$, respectively.

The following equivalences will hold in the interpretations we consider: $\mathbf{T}_\alpha(x,y) \iff \mathbf{T}_\beta(y,x) \iff \phi$, $\mathbf{T}^{\prec}_\alpha(z,y) \iff \forall x\,(x \prec z \Rightarrow \phi)$ and $\mathbf{T}^{\prec}_\beta(z,x) \iff \forall y\,(y \prec z \Rightarrow \phi)$.

Definition 6 An interpretation is $\mathtt{I}$-*normal* if it satisfies the following conditions:

1. The sets $\mathcal{I}(\mathtt{s})$ for $\mathtt{s} \in \mathtt{I}$ are the least sets satisfying the following property: for each constructor f of profile $\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$, and for each $(e_1, \ldots, e_n) \in \mathcal{I}(\mathtt{s}_1) \times \cdots \times \mathcal{I}(\mathtt{s}_n)$, the $\mathcal{I}$-term $f(e_1, \ldots, e_n)$ occurs in $\mathcal{I}(\mathtt{s})$.
2. If f is a constructor then $f^{\mathcal{I}}$ is the function mapping $(e_1, \ldots, e_n)$ to the $\mathcal{I}$-term $f(e_1, \ldots, e_n)$.
3. $\mathcal{I}$ interprets $\prec$ as $\lhd$.
4. For every rule $t \to s$ in $\mathcal{R}$, $\mathcal{I} \models \forall^*(t \simeq s)$.
5. For each rooted formula $\alpha = \phi[x, \mathbf{y}]$, $\mathcal{I} \models \forall^*(\mathbf{T}_\alpha(x, \mathbf{y}) \Leftrightarrow \alpha(x, \mathbf{y}))$.
6. For each rooted formula $\alpha = \phi[x, \mathbf{y}]$, $\mathcal{I} \models \forall^*(\mathbf{T}^{\prec}_\alpha(x, \mathbf{y}) \Leftrightarrow \forall z.(x \prec y \Rightarrow \alpha(z, \mathbf{y})))$

Remark 7 Note that all constructors are free in every $\mathtt{I}$-normal interpretation.

In the following we implicitly assume – unless specified otherwise – that all interpretations are $\mathtt{I}$-normal: satisfiability and logical entailment are considered w.r.t. $\mathtt{I}$-normal interpretations. The usual entailment relation (i.e., when the interpretations are arbitrary) is denoted by $\models_{fol}$.

Since by Condition 4, an $\mathtt{I}$-normal interpretation is a model of $\forall^*(t \simeq s)$ for every rule $t \to s$, we have the following result:

Proposition 8 *Let $\mathcal{I}$ be an $\mathtt{I}$-normal interpretation and ϕ be a formula. Then $\mathcal{I} \models \phi$ if and only if $\mathcal{I} \models \phi{\downarrow_{\mathcal{R}}}$.*

The next proposition states an immediate consequence of Condition 1 in Definition 6.

Proposition 9 *Let $\mathcal{I}$ be an $\mathtt{I}$-normal interpretation and e be an element of the domain of $\mathcal{I}$. There exists an $\mathcal{I}$-term t containing only symbols in K and elements of $\bigcup_{\mathtt{s} \in \mathtt{S} \setminus \mathtt{I}} \mathcal{I}(\mathtt{s})$ such that $\mathcal{I}(t) = e$.*

Proof The proof is by induction on the ordering $\lhd$. If the sort of e is not in $\mathtt{I}$, then it suffices to take $t = e$ (e is an $\mathcal{I}$-term since it is an element of the domain of $\mathcal{I}$). Otherwise, by Condition 1 of Definition 6, e is of the form $f(e_1, \ldots, e_n)$ where $f : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s} \in K$ and $\forall i \in [1, n]\, e_i \in \mathcal{I}(\mathtt{s}_i)$. For every $i \in [1, n]$, we have $e_i \lhd e$ by definition of $\lhd$, thus there exists a term t_i containing only symbols in K and elements of $\bigcup_{\mathtt{s} \in \mathtt{S} \setminus \mathtt{I}} \mathcal{I}(\mathtt{s})$ such that $\mathcal{I}(t_i) = e_i$. Then $f(t_1, \ldots, t_n)$ fulfills the required property.

2.3 Constrained Clauses

The proof procedure presented in Section 3 operates on constrained clauses of a particular form, formally defined as follows.

Definition 10 (Syntax) A *constrained clause* (or *c-clause*) is an expression of the form $[\![C \mid \mathcal{X}]\!]$, where:

– C is a clause,

– $\mathcal{X}$ is a conjunction of the form $\bigwedge_{i=1}^{n} f_i(\mathbf{u}_i) \simeq v_i \wedge \bigwedge_{i=1}^{m} \mathbf{T}^{\prec}_{\alpha_i}(\mathbf{w}_i)$, where $n, m \geq 0$ and $\forall i \in [1, n]$, $f_i \in \Lambda$.

Empty conjunctions are denoted by $\top$, and a c-clause of the form $[\![C \mid \top]\!]$ may simply be denoted by C. For any set of c-clauses S, we denote by $S_\top$ the set of c-clauses in S with a constraint part equal to $\top$.

Note that, in the above definition, the terms $\mathbf{u}_i$, v_i, $\mathbf{w}_i$ and the clause C may contain symbols in Λ. Constraints are taken as sets: we may write $l \in \mathcal{X}$ to state that l is an atom occurring in $\mathcal{X}$, $\mathcal{X} \subseteq \mathcal{Y}$ if $\mathcal{Y} = \mathcal{X} \wedge \mathcal{X}'$, or $\mathcal{X} = \mathcal{Y} \setminus \mathcal{X}'$ if $\mathcal{X} \wedge \mathcal{X}' = \mathcal{Y}$, modulo AC.

Definition 11 (Semantics) Let $[\![C \mid \mathcal{X}]\!]$ be a c-clause with free variables $x_1, \ldots, x_k$ of respective sorts $\mathtt{s}_1, \ldots, \mathtt{s}_k$. For all interpretations $\mathcal{I}$, we write $\mathcal{I} \models [\![C \mid \mathcal{X}]\!]$ if $\mathcal{I} \models C$ or $\mathcal{I} \not\models \mathcal{X}$. We write $\mathcal{I} \models \forall^* [\![C \mid \mathcal{X}]\!]$ if for all $e_1, \ldots, e_k$ in $\mathcal{I}(\mathtt{s}_1), \ldots, \mathcal{I}(\mathtt{s}_k)$, $\mathcal{I}[e_1/x_1, \ldots, e_k/x_k] \models [\![C \mid \mathcal{X}]\!]$

For every set of c-clauses S, we write $\mathcal{I} \models S$ iff $\mathcal{I} \models C$ holds for every $C \in S$ and $\mathcal{I} \models \forall^* S$ if $\mathcal{I} \models \forall^* C$ for every $C \in S$. We write $S_1 \models S_2$ if $\mathcal{I} \models S_2$ every time $\mathcal{I} \models S_1$.

Constrained Clause Sets and Formulas

We introduce a few shorthands to simplify notations. For every set of clauses S and for every constraint $\mathcal{X}$, we denote by $[\![S \mid \mathcal{X}]\!]$ the set of c-clauses $\{[\![C \mid \mathcal{X}]\!] \mid C \in S\}$. For every formula ϕ not containing quantifiers, $cnf(\phi)$ denotes a formula in conjunctive normal form equivalent[2] to ϕ. For all constraints $\mathcal{X}$, we denote by $[\![\phi \mid \mathcal{X}]\!]$ the set $[\![cnf(\phi) \mid \mathcal{X}]\!]$.

Example 12 If $\phi = (p(a, x) \wedge q(b, y)) \vee r(x, y)$ then:

$$[\![\phi \mid b \simeq y]\!] = \{[\![p(a, x) \vee r(x, y) \mid b \simeq y]\!], [\![q(b, y) \vee r(x, y) \mid b \simeq y]\!]\}.$$

2.4 Inference Trees

The induction rules in Section 3.5 will be defined by taking into account the so-called inference tree of the generated clauses in order to extract candidate inductive invariants.

Before defining the notion of an inference tree, we need to define the notion of an inference. Because our aim is to be as generic as possible, we do not want to assume that a specific set of inference rules is fixed. Instead, we prefer to define a general notion of inference, together with some conditions that will ensure that upcoming induction rules are correct.

Definition 13 An *inference* is a finite nonempty sequence of c-clauses, written $\mathcal{H}_1, \ldots, \mathcal{H}_n \vdash \mathcal{C}$. The c-clauses $\mathcal{H}_1, \ldots, \mathcal{H}_n$ are the *premises* of the inference and $\mathcal{C}$ is its *conclusion*. The inference is *sound* if $\forall^* \mathcal{H}_1, \ldots, \forall^* \mathcal{H}_n \models \mathcal{C}$.

Observe that $\forall^* \mathcal{H}_1, \ldots, \forall^* \mathcal{H}_n \models \mathcal{C}$, entails $\forall^* \mathcal{H}_1, \ldots, \forall^* \mathcal{H}_n \models \forall^* \mathcal{C}$, since by definition $\forall^* \mathcal{H}_1, \ldots, \forall^* \mathcal{H}_n$ contain no variable in $\mathcal{C}$. Note also however that, since free variables are handled rigidly, $\mathcal{H}_1, \ldots, \mathcal{H}_n \models \mathcal{C}$ does not necessarily hold.

[2] Usual structural transformations (see, e.g., [29,28]) can be used to avoid an exponential blow-up – the symbols $\mathbf{T}_\alpha$ can be used for this purpose, together with the axioms in Section 3.2, which ensures that equivalence is preserved in I-normal interpretations.

Definition 14 An *inference tree* δ is a partial function mapping c-clauses to finite sets of c-clauses. For every c-clause $\mathcal{C}$, the c-clauses in $\delta(\mathcal{C})$ are the *parents* of $\mathcal{C}$. The set of *δ-ancestors* (of simply ancestors if δ is fixed in the context) of $\mathcal{C}$ is the least set such that every parent of $\mathcal{C}$ is a δ-ancestor of $\mathcal{C}$ and every parent of a δ-ancestor of $\mathcal{C}$ is also a δ-ancestor of $\mathcal{C}$.

The inference tree δ is:

1. *well-founded* if the δ-ancestor relation is well-founded[3].
2. *sound* if for every $\mathcal{C} \in \mathcal{S}$, if $\delta(\mathcal{C}) = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, then the[4] inference $\mathcal{D}_1, \ldots, \mathcal{D}_n \vdash \mathcal{C}$ is sound.

Intuitively, $\delta(\mathcal{C})$ represents the set of premises that are used to derive $\mathcal{C}$, applying one of the inference rules. The function δ is partial, hence the set $\delta(\mathcal{C})$ may be undefined. This is the case when $\mathcal{C}$ has not been inferred but simply asserted, i.e., when it is a hypothesis occurring in the initial clause set. If $\delta(\mathcal{C}) = \emptyset$ and δ is sound, then this means that $\mathcal{C}$ is valid (it can be deduced from an empty set of premises). This is the case in particular for all the axioms introduced in Section 3.2. The use of the abstract notion of an inference tree makes our results applicable to a wide range of inference systems.

Definition 15 A *derivation* for an inference tree δ from a set of c-clauses S is a finite sequence of c-clauses $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ such that for every $i \in [1, n]$, either $\delta(\mathcal{C}_i)$ is undefined and $\mathcal{C}_i \in S$ or every c-clause in $\delta(\mathcal{C}_i)$ occurs in $\{\mathcal{C}_1, \ldots \mathcal{C}_{i-1}\}$, up to a renaming of variables. A derivation $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ is *ground* if $\mathcal{C}_i$ is ground for every $i \in [1, n]$. A derivation $(\mathcal{C}_1\sigma, \ldots, \mathcal{C}_n\sigma)$ is a *ground instance* of $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ if it is ground and for every $i \in [1, n]$, $\delta(\mathcal{C}_i\sigma) = \delta(\mathcal{C}_i)\sigma$.

In the following, we assume that all the considered inference trees are well-founded and sound. We also assume that for every c-clause $\mathcal{C}$ with $\delta(\mathcal{C}) = \{\mathcal{H}_1, \ldots, \mathcal{H}_n\}$, and for every $i, j \in [1, n]$ such that $i \neq j$, the c-clause $\mathcal{H}_i$ and its ancestors share no variable with the c-clause $\mathcal{H}_j$ and its ancestors this means that if a c-clause is used several times in a derivation then its inference tree may have to be duplicated and renamed[5].

3 Definition of the Calculus

In order to illustrate forthcoming definitions we will consider the following running example.

Example 16 Assume that `I` and `S` contain a unique sort `s`, and that the set of constructors is $\{a :\rightarrow \mathtt{s},\ f : \mathtt{s} \rightarrow \mathtt{s}\}$. Let p and q be predicate symbols with the same profile $\mathtt{s} \times \mathtt{s} \rightarrow \mathtt{bool}$. Let $H_1 = (p(a, a) \vee \forall x\, q(a, x))$, $H_2 = \forall x, x'\, (p(x, x') \Rightarrow p(f(x), x'))$, $H_3 = \forall x\, (q(x, a) \Rightarrow q(f(x), a))$. The goal is to prove that the following entailment holds[6]:

$$H_1 \wedge H_2 \wedge H_2 \models \forall x\, \exists y\, (p(x, y) \vee q(x, y))$$

[3] Because the inference trees are usually finite, it is sufficient to check that no clause is one of its own ancestors.

[4] The order of the premises is arbitrary.

[5] In practice, the inference tree does not have to be computed explicitly: we only have to be able to compute the candidate invariant $\mathfrak{I}^{\delta}(\mathcal{C}, \mathcal{X}, p)$ defined in Section 3.5. This can be done by generating renamings on demand.

[6] The reader can check that the implication does not hold if `s` is not an inductive sort.

By negating the conclusion and eliminating the existential quantification by Skolemization, the problem boils down to proving that the formula:

$$H_1 \wedge H_2 \wedge H_3 \wedge \forall y\,(\neg p(b, y) \wedge \neg q(b, y))$$

is unsatisfiable in $\mathtt{I}$-normal interpretations, where b is a fresh symbol (not a constructor) in Λ of profile $\rightarrow \mathtt{s}$. Since the interpretation is $\mathtt{I}$-normal, b must be interpreted by a term of the form $f^n(a)$ for $n \in \mathbb{N}$.

3.1 Overview of the Inductive Proof Procedure

As informally explained in the Introduction, the goal of our calculus is to derive universal properties $\forall x\,\alpha(x)$ by induction on x. This is done by infinite descent: in order to prove that such a property holds, we show that the implication $\neg\alpha(t) \Rightarrow \exists x\,(x \prec t \wedge \neg\alpha(x))$ is provable from the clause set at hand, for every ground constructed term t on the considered signature. Because $\prec$ is well-founded, this entails that $\alpha(t)$ must hold for every t. Two related issues arise:

1. How to choose the formula α? Our solution is to compute this formula incrementally: we start by collecting a minimal set of hypotheses that is sufficient to refute a base case (when t is minimal w.r.t. $\lhd$). The negation of the conjunction of the collected hypotheses suggests a first candidate invariant. Then other hypotheses are added when necessary to handle the other cases. Moreover, unification is used to compute the instances of the free variables occurring in α for which infinite descent applies.
2. How to check that $\exists x\,(x \prec t \wedge \neg\alpha(x))$ can be derived? This is not obvious because α is a set of clauses[7]. Our solution is to use the special symbol $\mathbf{T}_\alpha^\prec$: we check that $[\![\Box \mid \mathbf{T}_\alpha^\prec(t)]\!]$ can be derived, which will mean that there exists a proper subterm t' of t such that $\neg\alpha(t')$ holds. To this purpose, we need specific axioms that encode the semantics of the symbols $\mathbf{T}_\alpha$, $\mathbf{T}_\alpha^\prec$ and $\prec$. These axioms are defined in Section 3.2. In general $[\![\Box \mid \mathbf{T}_\alpha^\prec(t)]\!]$ can only be derived by distinguishing several cases, depending on the form of the term t. We thus introduce a rule, called the `Domain Decomposition` rule, to perform some kind of case splitting.

In general, the above reasoning is applied under some additional conditions, expressed by constraints or additional literals occurring in the clauses. In other words, a clause of the form $[\![C \mid \mathbf{T}_\alpha^\prec(t) \wedge \mathcal{X}]\!]$ may be obtained instead of $[\![\Box \mid \mathbf{T}_\alpha^\prec(t)]\!]$, meaning that the result holds under conditions $\neg C$ and $\mathcal{X}$, and the formula $[\![C \vee \alpha(x) \mid \mathcal{X}]\!]$ is eventually derived instead of $\alpha(x)$.

Before giving formal definitions, we explain in more details how the procedure works on our running example.

Example 17 (continued). The formula of Example 16 can be encoded by the following set of c-clauses S:

$$\begin{array}{ll} c_1 & p(a,a) \vee q(a,x) \\ c_2 & \neg p(x_2, x_2') \vee p(f(x_2), x_2') \\ c_3 & \neg q(x_3, a) \vee q(f(x_3), a) \\ c_4 & [\![\neg p(y_4, y_4') \mid b \simeq y_4]\!] \\ c_5 & [\![\neg q(y_5, y_5') \mid b \simeq y_5]\!] \end{array}$$

[7] Formula $\exists x\,(x \prec t \wedge \neg\alpha(x))$ may be encoded as a finite disjunction of cnf formulas $\neg\alpha(t_1) \vee \cdots \vee \neg\alpha(t_n)$, with $t_1, \ldots, t_n \lhd t$.

The constant b is abstracted away and respectively replaced by variables y_4 and y_5 in c_4 and c_5, while the constraints $b \simeq y_4$ and $b \simeq y_5$ are added to the clauses (applications of the `Abstraction` rule introduced in Section 3.3 on Page 16). Abstracting b will allow us to apply inference rules without having to replace b by a constructor term (which could be done by using domain axioms). Replacing b with a variable make it possible to use unification to find the values of b for which the above clause set can be refuted. Informally, the procedure will permit to prove that S is unsatisfiable by generating the inductive invariant $\forall x\, p(x,a) \vee q(x,a)$, which, along with c-clauses c_4 and c_5, permits to derive the c-clause $[\![\Box \mid b \simeq y_4]\!]$ stating that S is unsatisfiable, regardless of the value of b. This is done as follows. First, the c-clause $[\![\Box \mid b \simeq a]\!]$ is derived by `Resolution`, using c-clauses c_1, c_4 and c_5, which means that S is unsatisfiable if b is equal to a. The conjunction of instances of the c-clauses involving b that were used to generate $[\![\Box \mid b \simeq a]\!]$ is extracted. In this case, the conjunction consists of instances of c_4 and c_5, and is equal to $(\neg p(b,a) \wedge \neg q(b,y'_5))$. The negation of this conjunction (abstracting away the constant b) suggests a first candidate inductive invariant: $\forall x\, p(x,a) \vee q(x,y'_5)$. Note that this candidate invariant contains a free variable y'_5 and the goal is to determine for which instances of y'_5 this invariant propagates.

It is possible to attempt proving that this candidate is indeed an invariant by well-founded induction and contraposition, by trying to show that the following formula is a logical consequence of S:

$$(\star) : \forall x\, [\neg p(x,a) \wedge \neg q(x,y'_5) \;\Rightarrow\; (\exists x'\, x' \prec x \wedge \neg p(x',a) \wedge \neg q(x',y'_5))].$$

To this aim, a seemingly natural solution would be simply to add the cnf of the negation of $(\star)$ to S in order to derive a contradiction. However this is not satisfactory in our context, for the following reasons:

- First, the invariant contains free variables (here y'_5) that would have to be treated rigidly (they must be instantiated in the same way in every clause).
- Second, if the test $S \models (\star)$ fails (i.e., if $(\star)$ is not provable) then no information would be provided for constructing a new candidate invariant, although a natural way of generating a new candidate invariant is to refine the previous one by adding hypotheses that allow one to prove $(\star)$.

The goal in our context is thus not simply to check that $(\star)$ is a logical consequence of S, but rather to find the conditions under which $S \models (\star)$ holds. If no conditions are necessary, then the candidate is an invariant. Otherwise, it may be necessary to instantiate the free variables in $(\star)$, or to add hypothesis to consider another invariant candidate. Thus, instead of considering the formula $(\star)$ above, we consider the consequent of the implication instantiated by $[b/x]$:

$$(\dagger) :\; (\exists x'\, x' \prec b \wedge \neg p(x',a) \wedge \neg q(x',y'_5))$$

The formula $(\dagger)$ is encoded by the c-clause: $\left[\!\!\left[\Box \;\middle|\; b \simeq z \wedge \mathbf{T}^{\prec}_{p(x,a)\vee q(x,y'_5)}(z,y'_5)\right]\!\!\right]$. We first try to prove that $S \models (\dagger)$, then, by inspecting the clauses involving b used in the proof, we will be able either to deduce that $S \models (\star)$ or to refine the candidate invariant. Note that if $S \models (\star)$ then we necessarily have $S \models (\dagger)$: indeed, by definition, $\neg p(x,a) \wedge \neg q(x,y'_5)$ was constructed by extracting properties of b derivable from S, thus we must have $S \models (\neg p(x,a) \wedge \neg q(x,y'_5))[b/x]$.

The derivation of $[\![\Box \mid b \simeq a]\!]$ mentioned above is a proof that $S \models (\dagger)$ when $b = a$. Using c-clauses c_2, c_3, c_4 and c_5, it is possible to show that $S \models$

$[\![\neg p(y,a) \mid b \simeq f(y)]\!], [\![\neg q(y,a) \mid b \simeq f(y)]\!]$. This means that (†) holds when $b = f(y)$ and y'_5 is instantiated by a. Since a and f are the only constructors, b is either equal to a or of the form $f(y)$. Therefore, $S \models (\dagger)$ regardless of the value of b, but with the instantiation $[a/y']$[8]. At this point we can again collect all the properties of b that were used in this derivation. We get: $(\neg p(b,a) \wedge \neg q(b,a))$. This entails that the following implication is a logical consequence of S:

$$[\neg p(b,a) \wedge \neg q(b,a) \;\Rightarrow\; (\exists x'\, x' \prec b \wedge \neg p(x',a) \wedge \neg q(x',a))]$$

Furthermore, since $\neg p(b,a) \wedge \neg q(b,a)$ contains all the properties of b that are used to prove the consequent (†), the constant b can be abstracted away, which means that the following formula is also a logical consequence of S:

$$\forall x\, [\neg p(x,a) \wedge \neg q(x,a) \;\Rightarrow\; (\exists x'\, x' \prec x \wedge \neg p(x',a) \wedge \neg q(x',a))].$$

We deduce that the invariant does propagate when $y'_5 = a$. This entails that $\forall x\, (p(x,a) \vee q(x,a))$ holds, which contradicts c-clauses c_4 and c_5.

In the next sections, we show how to formalize the procedure sketched in Example 17. Our aim is to make all steps purely automatic and at the same time to avoid fixing the inductive scheme in advance. For example the inductive invariant $p(x,a) \vee q(x,a)$ and the decomposition $b = a \vee b = f(y)$ used in Example 17 should be *discovered automatically.*

3.2 Axioms for the Special Symbols

Definition 18 We define the following sets of c-clauses, where $\mathbf{x}, x, y, y', \mathbf{z}, z, x_i$ denote (vectors of) variables of the appropriate sorts, and α denotes an arbitrary rooted formula:

$$\begin{aligned}
\Gamma_{\mathbf{T}} &: \bigcup \left\{ [\![\alpha(\mathbf{x}) \vee \neg \mathbf{T}_\alpha(\mathbf{x}) \mid \top]\!] \;\middle|\; \alpha = \phi[\mathbf{x}] \right\} \\
\Gamma_{\mathbf{T}^\prec} &: \left\{ \left[\!\!\left[y' \not\prec y \vee \mathbf{T}_\alpha(y', \mathbf{z}) \;\middle|\; \mathbf{T}^\prec_\alpha(y, \mathbf{z}) \right]\!\!\right] \;\middle|\; \alpha = \phi[y, \mathbf{z}] \right\} \\
\Gamma^f_\prec &: \left\{ [\![x_i \prec f(x_1, \ldots, x_n) \mid \top]\!] \;\middle|\; f \in K, \mathit{arity}(f) = n \right\} \\
\Gamma_\prec &: \left\{ [\![x \not\prec y \vee y \not\prec z \vee x \prec z \mid \top]\!] \right\}
\end{aligned}$$

We let $\Gamma \stackrel{\text{def}}{=} \Gamma_{\mathbf{T}} \cup \Gamma_{\mathbf{T}^\prec} \cup \Gamma^f_\prec \cup \Gamma_\prec$.

Intuitively, $\Gamma_{\mathbf{T}}$ states that if $\mathbf{T}_\alpha(y, \mathbf{z})$ holds then $\alpha(y, \mathbf{z})$ holds[9], $\Gamma_{\mathbf{T}^\prec}$ states that if $\mathbf{T}^\prec_\alpha(y, \mathbf{z})$ holds then $\mathbf{T}_\alpha(y', \mathbf{z})$ holds for every strict subterm y' of y; and the sets $\Gamma^f_\prec$ and $\Gamma_\prec$ axiomatize the theory of the subterm ordering $\prec$.

Proposition 19 *The c-clauses in Γ are all valid in* `I`*-normal interpretations.*

[8] As we shall see, this assertion is derived by a form of case splitting, performed by the `Domain Decomposition` rule defined in Section 3.5, Page 24.

[9] This corresponds to an application of Tseitin's extension rule with the formula α. Note that in the definition of $\Gamma_{\mathbf{T}}$, $[\![\alpha(\mathbf{x}) \vee \neg \mathbf{T}_\alpha(\mathbf{x}) \mid \top]\!]$ denotes a set of c-clauses, as explained in Section 2.3.

Proof The validity of the c-clauses in $\Gamma^f_{\prec}$ and $\Gamma_{\prec}$ stems immediately from the definition of the interpretation of $\prec$. The validity of $\Gamma_{\mathbf{T}^{\prec}}$ and $\Gamma_{\mathbf{T}}$ follows from Conditions 5 and 6 of Definition 6.

Remark 20 The axioms for the converse implications (e.g., $\alpha(\mathbf{x}) \Rightarrow \mathbf{T}_\alpha(\mathbf{x})$) are not needed because the symbols $\mathbf{T}_\alpha$ and $\prec$ are used only positively and $\mathbf{T}^{\prec}_\alpha$ appears only in the constraints. Note also that we could get rid of the symbol $\mathbf{T}_\alpha$ by using the alternative axiom:

$$\Gamma_{\mathbf{T}^{\prec}} : \bigcup \left\{ \left[\!\!\left[y' \not\prec y \vee \alpha(y', \mathbf{z}) \mid \mathbf{T}^{\prec}_\alpha(y, \mathbf{z}) \right]\!\!\right] \;\middle|\; \alpha = \phi[y, \mathbf{z}] \right\}.$$

The current presentation is clearer and makes the derivations easier to understand.

Activated Formulas

In practice, using all the axioms defined above is very costly because the set $\Gamma_{\mathbf{T}}$ is infinite and the c-clauses in $\Gamma_{\mathbf{T}}$ and $\Gamma_{\mathbf{T}^{\prec}}$ interfere with each other (e.g., by resolving or superposing on the atoms $\mathbf{T}_\alpha(\mathbf{x})$), which will generate many irrelevant c-clauses. We thus restrict the use of the c-clauses in $\Gamma_{\mathbf{T}}$ and $\Gamma_{\mathbf{T}^{\prec}}$ to a special set of rooted formulas α, called *activated formulas*. Inferences involving c-clauses that correspond to formulas that are not in this set are blocked. Initially, no rooted formula is activated. Formulas are activated during proof search, using the `Trigger` and `Iteration` rules defined in Section 3.5. Informally, a formula is activated when it is considered as a candidate inductive invariant.

Remark 21 Intuitively, activating a rooted formula α makes it possible to use this formula as an inductive hypothesis – more precisely, an instance $\alpha(t, \mathbf{u})$ can be used provided an atom $\mathbf{T}^{\prec}_\alpha(s, \mathbf{u})$ with $t \prec s$ is added in the constraint.

3.3 Core Inference Rules

We consider a first set of rules, called the *core rules*, which contains standard inference rules whose definition is omitted (such as the `Superposition` or `Resolution` rules) together with three additional ones: `Constraint Factorization`, `Abstraction` and `Instantiation`. We assume that each rule in the set of standard rules is of the form $\frac{H_1, \ldots, H_n}{C\sigma}$, where $H_1\sigma, \ldots, H_n\sigma \models_{fol} C\sigma$[10]. For each such rule we introduce the corresponding rule operating on c-clauses, denoted by the same name for simplicity:

$$\frac{\left[\!\!\left[H_1 \mid \mathcal{X}_1 \right]\!\!\right], \ldots, \left[\!\!\left[H_n \mid \mathcal{X}_n \right]\!\!\right]}{\left[\!\!\left[C \mid \mathcal{X}_1 \wedge \cdots \wedge \mathcal{X}_n \right]\!\!\right] \sigma}$$

[10] Note that we do not have $H_1, \ldots, H_n \models_{fol} C\sigma$ in general, because our definition of logical entailment does not assume that free variables are implicitly universally quantified. Instead we have $\forall^* H_1, \ldots, \forall^* H_n \models_{fol} \forall^* C\sigma$.

Example 22 (continued). The following c-clauses can be derived from the set of c-clauses of Example 17.

$$
\begin{array}{lcll}
c_1 & p(a,a) \vee q(a,x) & & \\
c_2 & \neg p(x_2, x_2') \vee p(f(x_2), x_2') & & \\
c_3 & \neg q(x_3, a) \vee q(f(x_3), a) & & \\
c_4 & \left[\!\left[\neg p(y_4, y_4') \;\middle|\; b \simeq y_4 \right]\!\right] & & \\
c_5 & \left[\!\left[\neg q(y_5, y_5') \;\middle|\; b \simeq y_5 \right]\!\right] & & \\
c_6 & [\![q(a,x) \mid b \simeq a]\!] & (\texttt{Resolution}, c_1, c_4) & \frac{\mathrm{ff}}{\mathrm{ff}}[a/y_4,\, a/y_4'] \\
c_7 & [\![\Box \mid b \simeq a]\!] & (\texttt{Resolution}, c_6, c_5) & \frac{\mathrm{ff}}{\mathrm{ff}}[a/y_5,\, x/y_5'] \\
c_8 & \left[\!\left[\neg p(x_2, y_4') \;\middle|\; b \simeq f(x_2) \right]\!\right] & (\texttt{Resolution}, c_2, c_4) & \frac{\mathrm{ff}}{\mathrm{ff}}[f(x_2)/y_4,\, y_4'/x_2'] \\
c_9 & [\![\neg q(x_3, a) \mid b \simeq f(x_3)]\!] & (\texttt{Resolution}, c_3, c_5) & \frac{\mathrm{ff}}{\mathrm{ff}}[f(x_3)/y_5,\, a/y_5']
\end{array}
$$

The following rule permits the factorization of constraints:

`Constraint Factorization`:

$$\frac{[\![C \mid l_1 \wedge l_2 \wedge \mathcal{X}]\!]}{[\![C \mid l_1 \wedge \mathcal{X}]\!]\,\sigma} \quad \text{If } \sigma \text{ is an m.g.u. of } l_1 \text{ and } l_2.$$

A version of the calculus without the `Constraint Factorization` rule could also be defined; however, in the current version, the `Trigger`, `Induction` and `Iteration` rules apply to a single equation that could be obtained by the factorization rule. Without it, it would be necessary to define more complex versions of the former rules, reducing the readability of the paper

We introduce a rule to abstract away terms whose head is in Λ, so that inductive reasoning can be applied on such terms:

`Abstraction`:

$$\frac{[\![C[f(\mathbf{t})]_p \mid \mathcal{X}]\!]}{[\![C[x]_p \mid f(\mathbf{t}) \simeq x \wedge \mathcal{X}]\!]} \quad \text{If } f \in \Lambda,\ x \notin \mathrm{var}(C) \cup \mathrm{var}(\mathbf{t}) \cup \mathrm{var}(\mathcal{X})$$

Remark 23 If C contains several occurrences of $f(\mathbf{t})$ then the same variable x can be employed for each of them (the corresponding rule can be derived by combining the `Abstraction` and `Constraint Factorization` rules).

In all the examples in this paper, the c-clause set at hand is assumed to already be in abstracted form. Simple heuristics can be used to guide the application of the `Abstraction` rule. For instance the rule can be applied when an inference is "blocked" due to a clash in the unification algorithm between an occurrence of a function symbol $f \in \Lambda$ and a constructor. In this case, it is worth trying to replace the term of head f by a variable.

We finally define the dual to the `Abstraction` rule:

`Instantiation`:

$$\frac{[\![C \mid \mathcal{X} \vee t \simeq s]\!]}{[\![C \mid \mathcal{X}]\!]\,\sigma} \quad \text{If } \sigma = \mathrm{mgu}(t, s)$$

The `Instantiation` rule is very close to the usual Equality resolution rule of the Superposition calculus [2], the only difference is that it applies only on constraint

literals. For example the `Instantiation` rule derives the empty clause from $[\![\Box \mid t \simeq x]\!]$, if x does not occur in t. It is only used for this purpose in the present paper.

Redundancy Elimination

Standard redundancy elimination techniques can be easily extended to constrained clauses, provided the constraints are taken into account when checking redundancy. More precisely, a c-clause $[\![C \mid \mathcal{X}]\!]$ will be considered redundant w.r.t. some c-clauses $[\![C_i \mid \mathcal{X}_i]\!]$ with $i = 1, \ldots, n$ if C is redundant w.r.t. $C_1, \ldots, C_n$ (in the usual sense) and if $\mathcal{X}_1, \ldots, \mathcal{X}_n \subseteq \mathcal{X}$. For instance, the subsumption rule is defined as follows:

Definition 24 A c-clause $[\![C \mid \mathcal{X}]\!]$ *subsumes* $[\![D \mid \mathcal{Y}]\!]$ if there exists a substitution θ such that $C\theta \subseteq D$ and $\mathcal{X}\theta \subseteq \mathcal{Y}$.

Other deletion rules (e.g., equational simplification) are omitted, because none of the presented results depend on these. The definition of the induction rule only relies on the existence of an inference tree, regardless of how this tree is constructed. Similarly, all the usual features of the Superposition calculus, such as literal selection, ordering etc. can be integrated into the calculus, because the induction rules do not depend of them. Whether the completeness results in Section 6 extend to inference systems with deletion rules requires further investigation and is left for future work.

3.4 I-Soundness

It turns out that the soundness properties for inferences and inference trees (see Definitions 13 and 14) are not always sufficient for our purpose. We introduce a strictly stronger property, which also takes into account variable instantiations and constraint propagation from the premises to the conclusion[11].

Definition 25 An inference $\mathcal{H}_1, \ldots, \mathcal{H}_n \vdash \mathcal{C}$, where $\mathcal{H}_i = [\![H_i \mid \mathcal{X}_i]\!]$ and $\mathcal{C} = [\![C \mid \mathcal{Y}]\!]$ is I-*sound* w.r.t. a constraint $\mathcal{Y}'$, a function λ and a substitution σ if:

1. $\mathcal{Y}' \subseteq \mathcal{Y}$;
2. λ is a function mapping every occurrence of an equation $l \in \mathcal{Y}$ to a set of occurrences of equations $L \subseteq \bigcup_{i=1}^{n} \mathcal{X}_i$, the elements of which are called the *antecedents* of l;
3. $\mathrm{dom}(\sigma) \cap \mathrm{var}(\mathcal{C}) = \emptyset$;
4. σ is idempotent;
5. for every I-ground substitution θ of the variables in $\mathcal{C}$, there exists a set of indices $I^\theta \subseteq [1, n]$ such that:
   (a) for every $i \in I^\theta$, $\mathcal{X}_i\sigma\theta \downarrow_{\mathcal{R}} \subseteq \mathcal{Y}\theta \downarrow_{\mathcal{R}}$ and for every antecedent $l' \in \mathcal{X}_i$ of an equation $l \in \mathcal{Y}$, we have $l\theta\downarrow_{\mathcal{R}} = l'\sigma\theta\downarrow_{\mathcal{R}}$;
   (b) $\bigwedge_{i \in I^\theta} (H_i\sigma\theta) \models [\![C \mid \mathcal{Y}']\!]\,\theta$.

The substitution σ is called the *inf-substitution* of $\mathcal{H}_1, \ldots, \mathcal{H}_n \vdash \mathcal{C}$, or simply the inf-substitution of $\mathcal{C}$ if the inference is clear in the context. The constraint $\mathcal{Y}'$ is the *inference constraint* of $\mathcal{H}_1, \ldots, \mathcal{H}_n \vdash \mathcal{C}$, or simply the inference constraint of $\mathcal{C}$.

[11] We recall that $\mathcal{R}$ denotes the set of rules associated with the selectors, see Page 7 for details.

Remark 26 In general several substitutions σ may exist, in which case one of them is chosen arbitrarily. Similarly, if an inference is `I`-sound w.r.t. $\mathcal{Y}'$, λ and σ, then it is easy to check that it is also `I`-sound w.r.t. $\mathcal{Y}$ (with the above notations), an empty antecedent function and the same substitution σ. The results below are valid for any choice of σ, $\mathcal{Y}'$ and λ. However, the upcoming inference rules will depend on the choice of $\mathcal{Y}'$ and λ.

Intuitively, the conditions of Definition 25 serve several purposes. Firstly, they are meant to ensure that the conclusion of the inference is indeed a consequence of the premises. Secondly, and more importantly, they split the constraint $\mathcal{Y}$ of the conclusion into two parts[12]. The first part, $\mathcal{Y}'$, contains the constraint ensuring that the inference is indeed sound, whereas the second one contains the literals that are merely inherited from the premises. Those literals play no further role in the inference, in the sense that the entailment is still valid if these conditions are removed (see Condition 5b, only the constraint $\mathcal{Y}'$ is taken into account). Consider for instance an inference $[\![p(a) \mid a \simeq b]\!], \neg p(a') \models [\![\Box \mid a \simeq b \wedge a \simeq a']\!]$. Here the constraint $a \simeq b$ is inherited from the first premise (antecedent), whereas the condition $a \simeq a'$ is necessary to ensure that $\Box$ can indeed be derived from $p(a)$ and $\neg p(a')$. Thirdly, the definition relates every ground instance of the conclusion to a particular set of premises, used to derive this specific instance. This is crucial to handle reasoning by case analysis over the domain. Consider for instance the inference $p(0), p(s(x)) \models p(y)$ over the natural numbers. Here the conclusion is derived from the first clause if $y = 0$ and from the second one otherwise, i.e., $I^\theta = \{1\}$ if $\theta = [0/y]$ and $I^\theta = \{2\}$ otherwise.

Remark 27 In most cases, we will simply have $I^\theta = \{1, \ldots, n\}$, $\mathcal{X}_i\sigma \subseteq \mathcal{Y}$ and $\bigwedge_{i=1}^n (H_i\sigma) \models [\![C \mid \mathcal{Y}']\!]$. However this property does not hold for the `Domain Decomposition` rule introduced on Page 24, which explains why this more general definition is needed. Similarly, we have $\mathcal{Y}' = \emptyset$, except for axioms.

Example 28 Consider the `Resolution` inference:

$$[\![p(a, x) \mid n \simeq x]\!], [\![\neg p(y, b) \vee p(f(y), c) \mid \top]\!] \vdash [\![p(f(a), c) \mid n \simeq b]\!]$$

It is easy to check that $p(a, x)[b/x], (\neg p(y, b) \vee p(f(y), c))[a/y] \models p(f(a), c)$, thus we may set $I^\theta = \{1, 2\}$ regardless of θ, $\sigma = \frac{\mathfrak{fft}}{\mathfrak{fft}}[b/x,\ a/y]$ and $\mathcal{Y}' = \top$. Here the inference constraint $\mathcal{Y}'$ is empty, because the constraint $n \simeq x$ is not needed to ensure the soundness of the inference: it is merely propagated from the first premise to the conclusion. Now consider the `Abstraction` inference: $[\![p(n) \mid \top]\!] \vdash [\![p(x) \mid n \simeq x]\!]$. Here, the entailment $p(n) \models p(x)$ holds only if the constraint $n \simeq x$ is true. Thus for any θ we let: $I^\theta = \{1\}$, $\sigma = id$ and $\mathcal{Y}' = (n \simeq x)$. Similarly, consider the following inference, corresponding to an axiom in $\mathit{\Gamma}_{\mathbf{T}^\prec}$ (with, e.g., $\alpha = p(x)[x]$): $\vdash [\![y' \not\simeq y \vee \mathbf{T}_\alpha(y') \mid \mathbf{T}_\alpha^\prec(y)]\!]$. Here $I^\theta = \emptyset$, $\sigma = id$ and $\mathcal{Y}' = \mathbf{T}_\alpha^\prec(y)$.

In the previous examples the set I^θ does not depend on the substitution θ. Consider the inference $[\![\Box \mid n \simeq 0]\!], [\![\Box \mid n \simeq \mathit{succ}(y)]\!] \vdash [\![\Box \mid n \simeq x]\!]$, where x is a variable of sort $\mathtt{nat} \in \mathtt{I}$ and $0, \mathit{succ}$ are the only constructors of range `nat`. This inference is a particular case of the `Domain Decomposition` rule introduced later; it is sound because, by definition, every term of sort `nat` is equal to 0 or $\mathit{succ}(y)$, for some y. We take $\mathcal{Y}' = \top$. Here I^θ depends on the substitution θ:

$$I^\theta = \begin{cases} \{1\} & \text{if } \theta = [0/x] \\ \{2\} & \text{if } \theta = [\mathit{succ}(t)/x] \end{cases}$$

[12] The two parts are not necessarily disjoint.

Note that we cannot take $I^\theta = \{1, 2\}$, because Condition 5a of Definition 25 would not hold. The definition of the substitution σ is slightly more complicated than in the previous case: we have to find a substitution σ such that $succ(y)\sigma\theta = succ(t)$, if $\theta = [succ(t)/x]$. This is possible thanks to the selector functions and rewrite rules in $\mathcal{R}$: we let $\sigma = [succ_1^{-1}(x)/y]$, so that $\sigma\theta = [succ_1^{-1} \circ succ(t)/y]$, and $succ(y)\sigma\theta\downarrow_{\mathcal{R}} = succ(y\sigma\theta)\downarrow_{\mathcal{R}} = succ(t)$.

Example 29 The derivation of Example 22 corresponds to the following inference tree: $\delta(c_i)$ is undefined for $i = 1, \ldots, 5$, $\delta(c_6) = \{c_1, c_4\}$, $\delta(c_7) = \{c_6, c_5\}$, $\delta(c_8) = \{c_2, c_4\}$ and $\delta(c_9) = \{c_3, c_5\}$. The antecedent of $b \simeq a$ in c_6 is $b \simeq y_4$, the antecedents of $b \simeq a$ in c_7 are $b \simeq a$ (in c_6) and $b \simeq y_5$ (in c_5), the antecedent of $b \simeq f(x_2)$ in c_8 is $b \simeq y_4$ and the antecedent of $b \simeq f(x_3)$ in c_9 is $b \simeq y_5$. The respective inf-substitutions of c_6, c_7, c_8 and c_9 are $\sigma_6 = \frac{\mathtt{fft}}{\mathtt{fft}}[a/y_4,\ a/y_4']$, $\sigma_7 = \frac{\mathtt{fft}}{\mathtt{fft}}[a/y_5,\ x/y_5']$, $\sigma_8 = \frac{\mathtt{fft}}{\mathtt{fft}}[f(x_2)/y_4,\ y_4'/x_2']$ and $\sigma_9 = \frac{\mathtt{fft}}{\mathtt{fft}}[f(x_3)/y_5,\ a/y_5']$. The corresponding inference constraints are all equal to $\top$. These substitutions, inference constraints and inference tree will be used in forthcoming examples.

Proposition 30 *Let $\mathcal{I}$ be an* I*-normal interpretation and ϕ be a formula. There exists an* I*-ground substitution θ and an* I*-normal interpretation $\mathcal{J}$ such that:*

- *$\mathcal{I}$ and $\mathcal{J}$ coincide on every symbol occurring in ϕ.*
- *For every variable x occurring in ϕ, $\mathcal{J}(x) = \mathcal{J}(x\theta)$.*
- *For every $x \in \mathrm{dom}(\theta)$, x occurs in ϕ and $x\theta$ contains no variable occurring in ϕ.*

Proof Let $x_1, \ldots, x_k$ be the set of variables occurring in ϕ. By Proposition 9, there exist k terms $t_1, \ldots, t_k$ containing only symbols in K and elements of $\bigcup_{\mathtt{s} \in \mathtt{S} \setminus \mathtt{I}} \mathcal{I}(\mathtt{s})$ such that $\forall i \in [1, k]$, $\mathcal{I}(t_i) = \mathcal{I}(x_i)$. We associate every element $e \in \bigcup_{\mathtt{S} \setminus \mathtt{I}} \mathcal{I}(\mathtt{s})$ occurring in $t_1, \ldots, t_k$ with pairwise distinct fresh variables y_e not occurring in ϕ and we let $\mathcal{J}$ be the interpretation coinciding with $\mathcal{I}$ except that $\mathcal{J}(y_e) \stackrel{\text{def}}{=} e$. $\mathcal{J}$ is necessarily I-normal since it coincides with $\mathcal{I}$ on every symbol that is not a variable and $\mathcal{I}$ is I-normal. By construction, the first item of the proposition is fulfilled. We let θ be the substitution mapping each variable x_i to the term t_i' obtained from t_i by replacing each element e by y_e. By definition, θ is I-ground and fulfills the third item of the Proposition. Furthermore, we have $\mathcal{J}(x_i) = \mathcal{I}(x_i) = \mathcal{I}(t_i) = \mathcal{J}(t_i) = \mathcal{J}(t_i') = \mathcal{J}(x_i\theta)$ for every $i \in [1, k]$, thus the second item is also fulfilled.

The next lemma states that every I-sound inference is sound.

Lemma 31 *If an inference $\mathcal{H}_1, \ldots, \mathcal{H}_n \vdash \mathcal{C}$ is* I*-sound w.r.t. an inference constraint $\mathcal{Y}'$, an antecedent function λ and an inf-substitution σ, then $\mathcal{H}_1\sigma, \ldots, \mathcal{H}_n\sigma \models \mathcal{C}$.*

Proof We employ the same notations as in Definition 25, i.e., $\mathcal{H}_i = [\![H_i \mid \mathcal{X}_i]\!]$ and $\mathcal{C} = [\![C \mid \mathcal{Y}]\!]$. Let $\mathcal{I}$ be an I-normal interpretation validating $\mathcal{H}_1\sigma, \ldots, \mathcal{H}_n\sigma$ and $\mathcal{Y}$. We have to show that $\mathcal{I} \models C$. By Proposition 30, there exists an I-normal interpretation $\mathcal{J}$ and an I-ground substitution θ such that $\mathcal{I}$ and $\mathcal{J}$ coincide on every symbol in $\mathcal{H}_1\sigma, \ldots, \mathcal{H}_n\sigma, \mathcal{C}$, and $\mathcal{J}(x) = \mathcal{J}(x\theta)$ for every variable x in $\mathrm{var}(\mathcal{C})$. Since $\mathcal{I} \models \mathcal{Y}$, we have $\mathcal{J} \models \mathcal{Y}$ hence $\mathcal{J} \models \mathcal{Y}\theta$. Since $\mathcal{J}$ is I-normal, $\mathcal{J} \models \mathcal{Y}\theta\downarrow_{\mathcal{R}}$ by Proposition 8; furthermore, by Condition 1 of Definition 25, we have $\mathcal{Y}' \subseteq \mathcal{Y}$, so that $\mathcal{J} \models \mathcal{Y}'\theta$. Now by Condition 5a, for every $i \in I^\theta$, we have $\mathcal{X}_i\sigma\theta\downarrow_{\mathcal{R}} \subseteq \mathcal{Y}\theta\downarrow_{\mathcal{R}}$, thus $\mathcal{J} \models \mathcal{X}_i\sigma\theta$. But since $\mathcal{J} \models \mathcal{H}_i\sigma\theta$, necessarily, $\mathcal{J} \models H_i\sigma\theta$. Thus $\mathcal{J} \models \bigwedge_{i \in I^\theta}(H_i\sigma\theta)$ and by Condition 5b we deduce that $\mathcal{J} \models C\theta$. Therefore $\mathcal{J} \models C$, and since $\mathcal{I}$ and $\mathcal{J}$ coincide on all symbols in $\mathcal{C}$, we conclude that $\mathcal{I} \models C$.

Definition 32 An inference rule $\mathfrak{R}$ is `I`-*sound* (resp. sound) if for every instance $\frac{\mathcal{H}_1,\ldots,\mathcal{H}_n}{\mathcal{C}}$ of $\mathfrak{R}$, the inference $\mathcal{H}_1,\ldots,\mathcal{H}_n \vdash \mathcal{C}$ is `I`-sound (resp. sound).

Note that the `Instantiation` rule is not `I`-sound because the premise contains a constraint $t \simeq s$ that does not occur in the conclusion, thus Condition 5a of Definition 25 may not hold.

Proposition 33 *All the core rules except for the* ***`Instantiation`*** *rule, are* `I`*-sound. The* ***`Instantiation`*** *rule is sound.*

Proof The soundness of the `Instantiation` rule is immediate to prove. Let $\frac{\mathcal{H}_i,\ldots,\mathcal{H}_n}{\mathcal{C}}$ be a core rule, with $\mathcal{H}_i = [\![H_i \mid \mathcal{X}_i]\!]$ and $\mathcal{C} = [\![C \mid \mathcal{Y}]\!]$, and let θ be an `I`-ground substitution. It is easy to check that for every rule, except for the `Instantiation` rule, there exists a substitution σ such that $\mathcal{Y} = \bigwedge_{i=1}^n \mathcal{X}_i\sigma \wedge \mathcal{Y}'$ for some constraint $\mathcal{Y}'$ ($\mathcal{Y}'$ is empty for every rule except the `Abstraction` rule, for which it is $f(\mathbf{t}) \simeq x$), with $\bigwedge_{i=1}^n H_i\sigma \models [\![C \mid \mathcal{Y}']\!]$. We let: $I^\theta = \{1,\ldots,n\}$, regardless of θ. The antecedent relation is defined as follows: every atom occurrence $f(\mathbf{u}) \simeq v \in \mathcal{X}_i$ is an antecedent of the corresponding atom occurrence $(f(\mathbf{u}) \simeq v)\sigma \in \mathcal{Y}$, except for the `Abstraction` rule, for which the atom occurrence $f(\mathbf{t}) \simeq x$ has no antecedent. It is straightforward to check that the requirements of Definition 25 are satisfied.

3.5 Inference Rules for Inductive Reasoning

Intuitively, an inductive proof in our framework proceeds as follows. First, the `Abstraction` rule is applied on some term of head $f \in \Lambda$, to replace it by a variable. Afterwards, the usual inference rules may construct a refutation of the clause set for some particular value of this term. Then, information is gathered from this refutation to construct a candidate invariant along with a constraint that will be used to guarantee that the invariant is correct. Additional inference rules test whether the candidate invariant propagates, and if so, then it is generated as a universally quantified set of clauses.

Definition 34 Given $f \in \Lambda$, an f-*constraint* is a constraint of the form $\bigwedge_{i=1}^n f(\mathbf{u}_i) \simeq v_i$ where $n \geq 0$. In particular, $\top$ is an f-constraint. A position p is *eligible for* $\bigwedge_{i=1}^n f(\mathbf{u}_i) \simeq v_i$ *w.r.t. a sort* $\mathtt{s} \in \mathtt{I}$ if it is a position in v_i, for every $i \in [1,n]$, and if the terms $v_1|_p$, …, $v_n|_p$ are of sort $\mathtt{s}$.

The definition will be used in a context where the vectors $\mathbf{u}_i$ and terms v_i are unifiable. In most examples, the position p will be empty. Considering non-empty positions is useful if induction has to be applied on some subterm of $f(\mathbf{u}_i)$.

Example 35 Let $\mathcal{X} = a \simeq f(f(x,y),y) \wedge a \simeq f(z,z')$, where x,y,z,z' are variables and $f \in \Lambda$ is of profile $\mathtt{s} \times \mathtt{s} \to \mathtt{s}$, with $\mathtt{s} \in \mathtt{I}$. $\mathcal{X}$ in an a-constraint, and the positions ε, 1 and 2 are all eligible. The positions 1.1 and 1.2 are not eligible.

The following definition relates a c-clause $\mathcal{C}$, an f-constraint $\mathcal{X}$ and a position p to a candidate invariant, together with a constraint and a set of ancestors of $\mathcal{C}$. The invariant is obtained by considering a disjunction of instances of ancestors of $\mathcal{C}$.

Definition 36 Let δ be an inference tree. Let $\mathcal{X} = \bigwedge_{i=1}^{k} f(\mathbf{u}_i) \simeq v_i$ be an f-constraint and $\mathcal{C}$ be a c-clause of the form $[\![C \mid \mathcal{X} \wedge \mathcal{Y}]\!]$. Let p be an eligible position in $\mathcal{X}$ w.r.t. a sort $\mathtt{s}$ and consider a fresh variable z of sort $\mathtt{s}$, not occurring in the c-clauses under consideration[13]. The set of *auxiliary ancestors* $A^\delta(\mathcal{C},\mathcal{X},p)$, the *invariant* $\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p)$ and the *invariant constraint* $\mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p)$ associated with $\mathcal{C}$, $\mathcal{X}$ and p are inductively defined as follows:

1. If $\mathcal{X} = \top$, then $\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \bot$, $A^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \{\mathcal{C}\}$ and $\mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \mathcal{Y}$.
2. If $\mathcal{X}$ is of the form $f(\mathbf{u}) \simeq v[y]_p$ where y is a variable not occurring in $\mathcal{Y}$, $\mathbf{u}$ or v, then $\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \neg C[z/y]$, $A^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \emptyset$ and $\mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \top$.
3. If $\mathcal{X} \neq \top$, $\delta(\mathcal{C})$ is defined and $\delta(\mathcal{C}) \vdash \mathcal{C}$ is $\mathtt{I}$-sound[14] then:

$$A^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \bigcup_{\mathcal{D}\in\delta(\mathcal{C})} A^\delta(\mathcal{D},\mathcal{X}_\mathcal{D},p),$$

$$\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \bigvee_{\mathcal{D}\in\delta(\mathcal{C})} \mathfrak{I}^\delta(\mathcal{D},\mathcal{X}_\mathcal{D},p)\sigma,$$

$$\mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p) \stackrel{\text{def}}{=} \left(\mathcal{Y}' \wedge \bigwedge_{\mathcal{D}\in\delta(\mathcal{C})} \mathfrak{X}^\delta(\mathcal{D},\mathcal{X}_\mathcal{D},p)\sigma\right)[z/v_1|_p,\ldots,z/v_k|_p],$$

where $\mathcal{X}_\mathcal{D}$ denotes the conjunction of the antecedents of the literals in $\mathcal{X}$ that occur in $\mathcal{D}$, σ is the inf-substitution of $\mathcal{C}$, $\mathcal{Y}'$ is the inference constraint of $\mathcal{C}$, and p is eligible for $\mathcal{X}_D$ w.r.t. $\mathtt{s}$, for every $\mathcal{D} \in \delta(\mathcal{C})$.

The above objects are undefined if $\mathcal{C}$ and $\mathcal{X}$ are not of the required forms, if p is not eligible for $\mathcal{X}_D$ w.r.t. $\mathtt{s}$, if none of the conditions of Items 1–3 hold, or if (in Item 3) there is a $\mathcal{D} \in \delta(\mathcal{C})$ for which the functions are not defined.

Intuitively, $\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p)[z]$ denotes the negation of the properties satisfied by the terms occurring at position p in $v_1,\ldots,v_k$ that are used to derive $\mathcal{C}$; $A^\delta(\mathcal{C},\mathcal{X},p)$ is the set of c-clauses used in the derivation that contain no antecedent of equations in $\mathcal{X}$, and $\mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p)$ is the constraint ensuring that the derivation is sound.

Remark 37 Conditions 2 and 3 in Definition 36 possibly overlap, hence the above functions are actually non-deterministic. To make the functions deterministic, we may assume that Item 3 is applied with the highest priority when possible, to avoid trivial applications of the `Induction` rule.

Note that, in the definition above, $\mathcal{Y}$ possibly contains occurrences of f. Furthermore, if $\delta(\mathcal{C}) = \emptyset$ then $A^\delta(\mathcal{C},\mathcal{X},p) = \emptyset$, $\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p) = \bot$ and $\mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p) = \mathcal{Y}'$. Finally, the same variable z is used in every recursive call, which is possible since, thanks to the eligibility condition, all the occurrences of z have the same sort.

Example 38 (continued). Consider the derivation of Example 22, and the corresponding sets of ancestors constructed in Example 29. We have the following results:

$$\begin{array}{lll}
A^\delta(c_1,\top,\varepsilon) = \{c_1\}, & \mathfrak{X}^\delta(c_1,\top,\varepsilon) = \top, & \mathfrak{I}^\delta(c_1,\top,\varepsilon) = \bot,\\
A^\delta(c_4,b\simeq y_4,\varepsilon) = \emptyset, & \mathfrak{X}^\delta(c_4,b\simeq y_4,\varepsilon) = \top, & \mathfrak{I}^\delta(c_4,b\simeq y_4,\varepsilon) = p(z,y_4'),\\
A^\delta(c_5,b\simeq y_5,\varepsilon) = \emptyset, & \mathfrak{X}^\delta(c_5,b\simeq y_5,\varepsilon) = \top, & \mathfrak{I}^\delta(c_5,b\simeq y_5,\varepsilon) = q(z,y_5'),\\
A^\delta(c_6,b\simeq a,\varepsilon) = \{c_1\}, & \mathfrak{X}^\delta(c_6,b\simeq a,\varepsilon) = \top, & \mathfrak{I}^\delta(c_6,b\simeq a,\varepsilon) = p(z,a).
\end{array}$$

[13] Formally, $A^\delta(\mathcal{C},\mathcal{X},p)$, $\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p)$ and $\mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p)$ are thus also parameterized by the variable z, however this dependency is left implicit for the sake of readability.

[14] In practice, the condition will be checked simply by testing the rule applied to derive $\mathcal{C}$.

$$\begin{array}{cc}
\mathbf{c_1} : [\![\neg \mathbf{p}(\mathbf{x},\mathbf{y}) \vee \mathbf{p}(\mathbf{f}(\mathbf{x}),\mathbf{y}) \mid \mathbf{y} \not\simeq \mathbf{a}]\!] & \mathbf{c_2} : [\![\neg \mathbf{p}(\mathbf{u},\mathbf{b}) \mid \mathbf{n} \simeq \mathbf{u}]\!] \\
A^\delta(c_1, \bot, \varepsilon) = \{c_1\} & A^\delta(c_2, n \simeq u, \varepsilon) = \emptyset \\
\mathfrak{I}^\delta(c_1, \bot, \varepsilon) = \bot & \mathfrak{I}^\delta(c_2, n \simeq u, \varepsilon) = p(z, v) \\
\mathfrak{X}^\delta(c_1, \bot, \varepsilon) = y \not\simeq a & \mathfrak{X}^\delta(c_2, n \simeq u, \varepsilon) = \top
\end{array}$$

$$\begin{array}{c}
\mathbf{c_3} : [\![\neg \mathbf{p}(\mathbf{x},\mathbf{b}) \mid \mathbf{n} \simeq \mathbf{f}(\mathbf{x}) \wedge \mathbf{b} \not\simeq \mathbf{a}]\!] \\
A^\delta(c_3, n \simeq f(x), \varepsilon) = \{c_1\} \\
\mathfrak{I}^\delta(c_3, n \simeq f(x), \varepsilon) = p(z, b) \\
\mathfrak{X}^\delta(c_3, n \simeq f(x), \varepsilon) = b \not\simeq a
\end{array}$$

Fig. 1: Computing a Candidate Invariant from an Inference Tree: Example

From these, we conclude that

$$\begin{array}{l}
A^\delta(c_7, b \simeq a, \varepsilon) = \{c_1\}, \\
\mathfrak{X}^\delta(c_7, b \simeq a, \varepsilon) = \top, \\
\mathfrak{I}^\delta(c_7, b \simeq a, \varepsilon) \ = (p(z,a) \vee q(z, y_5'))\sigma_7 = q(z,x) \vee p(z,a).
\end{array}$$

Intuitively, this indicates that $\Box$ can be derived from $\neg(q(z,x) \vee p(z,a))[a/z]$ using c-clause $\{c_1\}$ under constraint $\top$. The procedure will then try to prove the formula $\forall z\,(q(z,x) \vee p(z,a))$ by induction on z. Note that the extracted inductive invariant contains a free variable x, that remains to be instantiated. In this example, the invariant is actually provable, as we shall see in Example 52, in the case where $x = a$.

Continuing our constructions, we also have

$$\begin{array}{l}
A^\delta(c_2, \top, \varepsilon) = \{c_2\},\ \mathfrak{X}^\delta(c_2, \top, \varepsilon) = \top,\ \mathfrak{I}^\delta(c_2, \top, \varepsilon) = \bot, \\
A^\delta(c_3, \top, \varepsilon) = \{c_3\},\ \mathfrak{X}^\delta(c_3, \top, \varepsilon) = \top,\ \mathfrak{I}^\delta(c_3, \top, \varepsilon) = \bot.
\end{array}$$

Therefore,

$$\begin{array}{lll}
A^\delta(c_8, b \simeq f(x_2), \varepsilon) = \{c_2\} & \text{and } A^\delta(c_9, b \simeq f(x_3), \varepsilon) = \{c_3\}, \\
\mathfrak{X}^\delta(c_8, b \simeq f(x_2), \varepsilon) = \top & \text{and } \mathfrak{X}^\delta(c_9, b \simeq f(x_3), \varepsilon) = \top, \\
\mathfrak{I}^\delta(c_8, b \simeq f(x_2), \varepsilon) = p(z, y_4') & \text{and } \mathfrak{I}^\delta(c_9, b \simeq f(x_3), \varepsilon) = q(z,a).
\end{array}$$

Example 39 Consider the inference: $[\![p(x) \mid n \simeq x]\!], [\![\neg p(f(f(y))) \vee p(y) \mid \top]\!] \vdash [\![p(y) \mid n \simeq f(f(y))]\!]$. Let $\mathcal{C} = [\![p(y) \mid n \simeq f(f(y))]\!]$, $\mathcal{X} = n \simeq f(f(y))$, and $p = 1$. Then $A^\delta(\mathcal{C}, \mathcal{X}, p)$, $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)$ and $\mathfrak{X}^\delta(\mathcal{C}, \mathcal{X}, p)$ are undefined since p is not a position in the term x in the constraint of the first clause. Indeed, $[\![p(y) \mid n \simeq f(f(y))]\!]$ is not derived from c-clauses of the form $[\![D \mid n \simeq f(y)]\!]$. If $q = \varepsilon$, then $A^\delta(\mathcal{C}, \mathcal{X}, q) = \{\neg p(f(f(y))) \vee p(y)\}$, $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p) = p(z)$ and $\mathfrak{X}^\delta(\mathcal{C}, \mathcal{X}, p) = \top$.

Because Definition 32 takes into account the normal form of some terms modulo $\mathcal{R}$, we need to introduce a notion of unifier modulo $\mathcal{R}$. This notion is used for theoretical purposes only: in practice, the selector functions will never occur in the clauses under consideration and standard unification will be used, but terms containing selectors appear in the induction in the proof of Lemma 56. Selectors are useful only to ensure that substitutions satisfying the conditions of Definition 32 always exist (see in particular the proof of Lemma 61).

Definition 40 A substitution σ is a *unifier of a set of terms* $t_1, \ldots, t_n$ *modulo* $\mathcal{R}$ if for every $i, j \in [1, n]$ $t_i\sigma{\downarrow_\mathcal{R}} = t_j\sigma{\downarrow_\mathcal{R}}$.

Example 41 Assume $\mathtt{s}$ is an inductive sort with constructors $a :\rightarrow \mathtt{s}$ and $f : \mathtt{s} \rightarrow \mathtt{s}$. Then the terms $g(a, x)$ and $g(f_1^{-1}(f(y)), b)$ admit the substitution $\sigma \stackrel{\text{def}}{=} [b/x, a/y]$ as a unifier modulo $\mathcal{R}$.

We now define the inference rules for performing inductive reasoning. Two of them, the so-called `Trigger` and `Iteration` rules are actually not inference rules: they do not infer any c-clause, but merely activate some formulas, as explained in Section 3.2, when specific conditions are satisfied. Thus, a set of rooted formulas is maintained during proof search, distinct from the set of clauses at hand, and "Activate formula α" states that α is added into this latter set. The formulas that are activated are candidate invariants. This activation will in turn trigger new inferences, since an inference is allowed on an axiom in $\Gamma_{\mathbf{T}}$ or $\Gamma_{\mathbf{T}^{\prec}}$ iff the corresponding formula α is activated.

The formulas that are activated are candidate invariants.

The `Trigger` rule.

We first define the `Trigger` rule, which activates a formula when a refutation is obtained for a particular inductive term, possibly under some additional conditions, expressed as constraints or ordinary literals. The rule is defined as follows:

`Trigger`:

$$\frac{\mathcal{C} : [\![C \mid f(\mathbf{u}) \simeq v[t]_p \wedge \mathcal{X}]\!]}{\text{Activate formula } \mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[t]_p, p)[z, \mathbf{y}]}$$

If:

- $\mathrm{var}(t) \cap (\mathrm{var}(C) \cup \mathrm{var}(\mathcal{X}) \cup \mathrm{var}(\mathbf{u}) \cup \mathrm{var}(v)) = \emptyset$.
- $\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)$ is defined and contains no selector function[a].
- $\mathbf{y}$ is the vector of the variables occurring in $\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[t]_p, p)$ and distinct from z (the order is chosen arbitrarily).

[a] The latter condition is meant to avoid introducing selector functions in the c-clause sets – since these functions are introduced only for theoretical purposes, they should not occur in the clauses.

The premise $\mathcal{C}$ states that $f(\mathbf{u})$ is distinct from $v[t]_p$, if the condition $\neg C \wedge \mathcal{X}$ holds. The purpose of the `Trigger` rule is to generate a candidate inductive invariant formula, taking the derivation generating $\mathcal{C}$ as the proof of this invariant for the base case t. The candidate inductive invariant is the formula $\alpha(z, \mathbf{y}) \stackrel{\text{def}}{=} \mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[t]_p, p)[z, \mathbf{y}]$. In order to prove that this is indeed an invariant, we have to show that it propagates, i.e., that $\alpha(z, \mathbf{y})$ holds if $\alpha(z', \mathbf{y})$ is true for every proper subterm z' of z. To this purpose, we activate the formula α, and let the procedure infer consequences of the axioms corresponding to α, using in particular the axioms specifying the interpretation of $\prec$, $\mathbf{T}_{\alpha}$ and $\mathbf{T}_{\alpha}^{\prec}$ that were introduced in Section 3.2. If the above implication holds, then the c-clause $[\![\Box \mid f(\mathbf{u}) \simeq v[z]_p \wedge \mathbf{T}_{\alpha}^{\prec}(z, \mathbf{y})]\!]$ will be derived eventually from S and the axioms $[\![\alpha \vee \neg \mathbf{T}_{\alpha}(x, \mathbf{y}) \mid \top]\!]$, $[\![x' \not\prec x \vee \mathbf{T}_{\alpha}(x, \mathbf{y}) \mid \mathbf{T}_{\alpha}^{\prec}(x, \mathbf{y})]\!]$ and $\Gamma_{\prec}^{f} \cup \Gamma_{\prec}$. This c-clause means that if $f(\mathbf{u})$ is of the form $v[z]_p$ then $\mathbf{T}_{\alpha}^{\prec}(z, \mathbf{y})$ is false, i.e., that there exists a proper subterm z' of z such that $\alpha(z', \mathbf{y})$ does not hold. This information will be used in turn by the `Induction` rule below to derive the desired conclusion.

In most examples, the `Trigger` rule will be applied with $C = \Box$ (see Example 69 for a case where $C \neq \Box$). In practice, restricting the application of the rule to $C = \Box$ seems like a reasonable choice, because otherwise the search space would be huge.

Example 42 (continued from Example 38). In our running example, the `Trigger` rule applies on $c_7 = [\![\Box \mid b \simeq a]\!]$, where α is the rooted formula $\mathfrak{I}^\delta(c_7, b \simeq a, \varepsilon)[z, x] = (q(z,x) \vee p(z,a))[z,x]$.

Once this formula is activated, other c-clauses can be derived, using the axioms corresponding to α together with the c-clauses $c_8 = [\![\neg p(x_2, y_4') \mid b \simeq f(x_2)]\!]$ and $c_9 = [\![\neg q(x_3, a) \mid b \simeq f(x_3)]\!]$ generated in Example 22 and which are reproduced here for readability:

$$\begin{array}{lcl}
c_{10} & [\![z_1 \not\prec z' \vee q(z_1,x_1) \vee p(z_1,a) \mid \mathbf{T}_\alpha^\prec(z', x_1)]\!] & (\Gamma_{\mathbf{T}^\prec}, \Gamma_{\mathbf{T}}) \\
c_{11} & [\![z_1 \not\prec z' \vee q(z_1,x_1) \mid b \simeq f(z_1) \wedge \mathbf{T}_\alpha^\prec(z', x_1)]\!] & (c_{10}, c_8) \\
c_{12} & [\![z_1 \not\prec z' \mid b \simeq f(z_1) \wedge \mathbf{T}_\alpha^\prec(z', a)]\!] & (c_{11}, c_9) \\
c_{13} & [\![\Box \mid b \simeq f(z_1) \wedge \mathbf{T}_\alpha^\prec(f(z_1), a)]\!] & (c_{12}, \Gamma_\prec^f)
\end{array}$$

The inf-substitutions corresponding to c_{11} c_{12} and c_{13} are respectively $\sigma_{11} = \frac{\text{fff}}{\text{fff}}[z_1/x_2, a/y_4']$, $\sigma_{12} = \frac{\text{fff}}{\text{fff}}[z_1/x_3, a/x_1]$ and $\sigma_{13} = [f(z_1)/z']$. Associated with the c-clauses above are the following elements, constructed using those of Example 38 and the fact that inference constraints for the `Resolution` rule are always empty:

$$\begin{array}{ll}
\mathfrak{X}^\delta(c_{10}, \top, \varepsilon) = \mathbf{T}_\alpha^\prec(z', x_1), & \mathfrak{I}^\delta(c_{10}, \top, \varepsilon) = \bot \\
\mathfrak{X}^\delta(c_{11}, b \simeq f(z_1), \varepsilon) \equiv \mathbf{T}_\alpha^\prec(z', x_1), & \mathfrak{I}^\delta(c_{11}, b \simeq f(z_1), \varepsilon) \equiv p(z,a) \\
\mathfrak{X}^\delta(c_{12}, b \simeq f(z_1), \varepsilon) \equiv \mathbf{T}_\alpha^\prec(z', a), & \mathfrak{I}^\delta(c_{12}, b \simeq f(z_1), \varepsilon) \equiv p(z,a) \vee q(z,a)
\end{array}$$

We also have $\mathfrak{I}^\delta(c_{13}, b \simeq f(z_1), \varepsilon) \equiv p(z,a) \vee q(z,a)$, and

$$\begin{aligned}
\mathfrak{X}^\delta(c_{13}, b \simeq f(z_1), \varepsilon) &= \Big(\big(\mathfrak{X}^\delta(c_{12}, b \simeq f(z_1), \varepsilon) \wedge \mathfrak{X}^\delta(c_9, \top, \varepsilon) \big) \sigma_{13} \Big) [z/f(z_1)] \\
&= \mathbf{T}_\alpha^\prec(z', a) \sigma_{13} [z/f(z_1)] \\
&= \mathbf{T}_\alpha^\prec(z, a).
\end{aligned}$$

The c-clause c_{13} states that when the head symbol of b is the constructor f, there exists a proper subterm z' of b such that $p(z',a) \vee q(z',a)$ does not hold. In order to be able to apply the infinite descent principle, it will be necessary to generate the c-clause $[\![\Box \mid b \simeq x' \wedge \mathbf{T}_\alpha^\prec(x', a)]\!]$; we will show how to derive this c-clause using another rule, called the `Domain Decomposition` rule, see the definition below and Example 46.

The `Domain Decomposition` rule.

We define a rule that performs a form of case analysis. The idea is to derive a general property from a set of instances that covers all possible cases. We introduce the following:

Definition 43 A *constructor term* is a possibly non-ground term built on the set of constructors. A set of constructor terms T of a given sort `s` is *covering* if for every `I`-ground term t of sort `s`, there exists a term $s \in T$ such that t is an instance of s.

Note that all `I`-ground terms are constructor terms, but the converse does not hold: for instance a variable of a sort in `I` is a constructor term but it is not `I`-ground. There exist several algorithms for detecting whether a given set of terms is covering, see, e.g., [15].

Example 44 Assume that $\texttt{nat}, \texttt{list} \in \texttt{I}$, where the only constructors are $0 :\rightarrow \texttt{nat}$, $succ : \texttt{nat} \rightarrow \texttt{nat}$, $nil :\rightarrow \texttt{list}$ and $cons : \texttt{nat} \times \texttt{list} \rightarrow \texttt{list}$. Let x, y be variables of respective sorts `nat` and `list`. The sets $\{x\}$, $\{0, succ(x)\}$ and $\{0, succ(0), succ(succ(x))\}$ are covering for the sort `nat`; the sets $\{nil, cons(x, y)\}$ and $\{nil, cons(0, y), cons(succ(x), y)\}$ are covering for the sort `list`.

`Domain Decomposition`:

$$\frac{[\![C_1 \mid \beta_1(t_1) \wedge \mathcal{X}_1]\!] \quad \ldots \quad [\![C_n \mid \beta_n(t_n) \wedge \mathcal{X}_n]\!]}{[\![C_1 \vee \cdots \vee C_n \mid \mathcal{X}_1 \cdots \wedge \mathcal{X}_n \wedge \beta_1(x)]\!]\,\eta}$$

If $\beta_1, \ldots, \beta_n$ are rooted formulas, η is a most general idempotent substitution such that $\beta_j\eta \subseteq \beta_1\eta$, for every $j = 2, \ldots, n$, $\{t_1, \ldots, t_n\}$ is covering, x is a fresh variable of the same sort as $t_1, \ldots, t_n$, and the variables occurring in t_i do not occur in C_i, β_i and $\mathcal{X}_i$.

The `Domain Decomposition` rule performs a case analysis on the terms of the same sort as $t_1, \ldots, t_n$. Intuitively, if a set $\{t_1, \ldots, t_n\}$ is covering for some sort $\texttt{s} \in \texttt{I}$, then every term t of sort `I` is an instance of some t_i, for $i \in [1, n]$. Thus if $\forall^* p(t_1), \ldots, \forall^* p(t_n)$ hold for some property p then $\forall^* p(x)$ also holds, where x is fresh variable. The rule essentially applies this scheme with $p(x) \equiv [\![\Box \mid \beta_1(x)]\!]\,\eta$.

Example 45 Assume that the only constructors of `nat` are 0 and $succ$. The `Domain Decomposition` rule infers the c-clause $[\![\Box \mid n \simeq x]\!]$ from the set of premises $\{[\![\Box \mid n \simeq 0]\!]\,,\ [\![\Box \mid n \simeq succ(0)]\!]\,,\ [\![\Box \mid n \simeq succ(succ(x))]\!]\}$. The covering set is $\{0, succ(0), succ(succ(x))\}$ and the rooted formulas β_1, β_2 and β_3 are: $\beta_1 = \beta_2 = \beta_3 = (n \simeq x)[x]$. From $[\![\Box \mid n \simeq x]\!]$, $\Box$ can be derived by `Instantiation`.

Example 46 In Examples 22 and 42, the following c-clauses were derived:

$$\begin{array}{ll} c_7 & [\![\Box \mid b \simeq a]\!] \\ c_{13} & [\![\Box \;\big|\; b \simeq f(z_1) \wedge \mathbf{T}^{\prec}_{\alpha}(f(z_1), a)]\!] \end{array}$$

Intuitively, c_7 means that b cannot be equal to a and c_{13} means that if b is equal to $f(z_1)$, then there exists a proper subterm z' of b such that $(p(z', a) \vee q(z', a))$ does not hold. We apply the `Domain Decomposition` rule on c_{13} and c_7, with $\beta_1 = (b \simeq x' \wedge \mathbf{T}^{\prec}_{\alpha}(x', a))[x']$ and $\beta_2 = (b \simeq x')[x']$, $t_1 \stackrel{\text{def}}{=} f(z_1)$, and $t_2 \stackrel{\text{def}}{=} a$. The set $\{f(z_1), a\}$ is covering and $\beta_2 \subseteq \beta_1$, thus we may derive:

$$c_{14} \quad [\![\Box \;\big|\; b \simeq x' \wedge \mathbf{T}^{\prec}_{\alpha}(x', a)]\!] \quad (\texttt{Domain Decomposition}, c_7, c_{13})$$

The c-clause c_{14} states that *regardless of the value of* b, there exists a proper subterm z of b such that $p(z, a) \vee q(z, a)$ does not hold.

Example 47 In Example 46, we derived $c_{14} : [\![\Box \;\big|\; b \simeq x' \wedge \mathbf{T}^{\prec}_{\alpha}(x', a)]\!]$ from $c_7 : [\![\Box \mid b \simeq a]\!]$ and $c_{13} : [\![\Box \;\big|\; b \simeq f(z_1) \wedge \mathbf{T}^{\prec}_{\alpha}(f(z_1), a)]\!]$. Assume for instance that $x'\theta = f(s)$. To fulfill Condition 5a of Definition 25, we need to find a premise $[\![C \mid \mathcal{X}]\!]$ and a substitution σ_{14} independent of θ such that $\mathcal{X}\sigma_{14}\theta{\downarrow_{\mathcal{R}}} \subseteq (b \simeq x' \wedge \mathbf{T}^{\prec}_{\alpha}(x', a))\theta{\downarrow_{\mathcal{R}}} = b \simeq f(s) \wedge \mathbf{T}^{\prec}_{\alpha}(f(s), a)$. It is clear that the only premise that can fulfill this property is c_{13}, which corresponds to the set $I^{\theta} = \{2\}$ (since c_{13} is the second premise). We must have $(b \simeq f(z_1))\sigma_{14}\theta{\downarrow_{\mathcal{R}}} \subseteq b \simeq f(s)$, i.e., $z_1\sigma_{14}\theta{\downarrow_{\mathcal{R}}} = s$. We cannot take $z_1\sigma_{14} = s$, since θ is arbitrary, thus this property can only hold if $z_1\sigma_{14} = f_1^{-1}(x')$ (see also Example

28). The inf-substitution of c_{14} is thus $\sigma_{14} \stackrel{\text{def}}{=} [f_1^{-1}(x')/z_1]$, and because x' occurs in both literals in the constraint, we have:

$$\begin{aligned}\mathfrak{X}^\delta(c_{14}, b \simeq x', \varepsilon) &= \mathfrak{X}^\delta(c_{13}, b \simeq f(z_1), \varepsilon)\sigma_{14} \;\equiv\; \mathbf{T}_\alpha^\prec(z, a),\\ \mathfrak{I}^\delta(c_{14}, b \simeq x', \varepsilon) &= (\mathfrak{I}^\delta(c_7, b \simeq a, \varepsilon) \vee \mathfrak{I}^\delta(c_{13}, b \simeq f(z_1), \varepsilon))\sigma_{14}\\ &\equiv q(z, x) \vee p(z, a) \vee q(z, a).\end{aligned}$$

The ***Induction*** *rule.* The `Induction` rule is the key rule for inductive reasoning. We first introduce the relation $\models_s$, which denotes a form of subsumption between formulas. Intuitively, $\models_s$ is a restricted (decidable) entailment relation that will be used to check whether all the hypotheses used in the inductive proof are entailed by the the considered invariant, see Condition 1 in the definition of the rule.

Definition 48 Let ϕ and ψ be two quantifier-free formulas in nnf. We write $\psi \models_s \phi$ if one of the following conditions holds:

- $\phi = \top$.
- $\psi = \bot$.
- $\phi = \psi$.
- $\psi = (\psi_1 \vee \psi_2)$ and $\psi_i \models_s \phi$ for all $i \in \{1, 2\}$.
- $\phi = (\phi_1 \wedge \phi_2)$ and $\psi \models_s \phi_i$ for all $i \in \{1, 2\}$.
- $\phi = (\phi_1 \vee \phi_2)$ and $\psi \models_s \phi_i$ for some $i \in \{1, 2\}$.
- $\psi = (\psi_1 \wedge \psi_2)$ and $\psi_i \models_s \phi$ for some $i \in \{1, 2\}$.

The definition extends to arbitrary quantifier-free formulas by transforming them into nnf. We write $\forall \mathbf{x}\, \phi \models_s \psi$ if there exists a vector of terms $\mathbf{t}$ such that $\phi[\mathbf{t}/\mathbf{x}] \models_s \psi$.

Proposition 49 *If $\psi \models_s \phi$ then $\psi \models \phi$.*

Proof By a straightforward induction on the set of formulas.

Remark 50 The relation $\models_s$ in Definition 48 is given here as an example. Any other relation could be used instead, as long as Proposition 49 is satisfied, and provided an algorithm is available to compute, given two formulas ϕ and ψ, a substitution σ such that $\phi\sigma \models_s \psi\sigma$ is satisfied (if any).

The `Induction` rule is defined in Figure 2. The substitution σ can be computed by combining the rules in Definition 48 (modulo AC) with any unification algorithm.

Remark 51 In most examples, the vector $\mathbf{y}$ will be empty. Considering nonempty vectors of variables $\mathbf{y}$ allows one to handle inductive invariants containing universally quantified variables, other than the variable on which induction is applied. See Example 68 for such an application.

It should be noted that the conclusion of the `Induction` rule is a set of c-clauses, not a single c-clause because $\alpha(x, \mathbf{t})$ is a formula, see Section 2.3 for more details.

`Induction`:

$$\frac{\mathcal{C} : [\![C \mid f(\mathbf{u}) \simeq v[x]_p \wedge \mathcal{X} \wedge \mathbf{T}^{\prec}_{\alpha}(x, \mathbf{t})]\!]}{[\![C \vee \alpha(x, \mathbf{t})\sigma \mid \mathcal{Y}\sigma]\!]}$$

If there exists a vector of variables $\mathbf{y}$ such that:

1. $\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)$ is defined and the substitution σ is a most general idempotent substitution such that $(\forall \mathbf{y}\, \mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z])\sigma \models_s \alpha(x, \mathbf{t})\sigma$, $x\sigma = x$ and $\forall y \in \mathrm{dom}(\sigma)$, $x \notin \mathrm{var}(y\sigma)$.
2. x does not occur in C.
3. $\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)$ and $\mathfrak{X}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)$ contain no occurrence of x and no selector function[a].
4. $\mathcal{Y} = \mathfrak{X}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z] \setminus \mathbf{T}^{\prec}_{\alpha}(x, \mathbf{t})$.
5. The variables in $\mathbf{y}$ do not occur in $C\sigma$, $\mathcal{Y}\sigma$.

[a] The latter condition is not necessary for soundness, but avoids introducing selector functions in the c-clause sets.

Fig. 2: The `Induction` rule

The `Induction` rule applies when the candidate inductive invariant propagates. This is the case when a c-clause of the form $[\![\Box \mid \mathbf{T}^{\prec}_{\alpha}(x, \mathbf{t})]\!] \equiv \neg\mathbf{T}^{\prec}_{\alpha}(x, \mathbf{t})$ can be derived, possibly with some additional conditions in the general case, here, $f(\mathbf{u}) \simeq v[x]_p$, $\mathcal{X}$ and $\neg C$. By definition of the interpretation of $\mathbf{T}^{\prec}_{\alpha}(x, \mathbf{t})$, this means that there exists a proper subterm x' of x such that $\neg\alpha(x', \mathbf{t})$ holds. If the above c-clause is derived from a set of hypotheses $\neg\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z]$ that is – as stated by the converse of Condition 1 – a logical consequence of $\neg\alpha(x, \mathbf{t})$, then the implication $\neg\alpha(x, \mathbf{t}) \Rightarrow \exists x'\, x' \prec x \wedge \neg\alpha(x', \mathbf{t})$ is a logical consequence of the considered clause set. By contrapositive, this means that $\alpha(x, \mathbf{t})$ is true if $\alpha(x', \mathbf{t})$ is true for every proper subterm of x (this is the usual Noetherian induction scheme). Thus $\alpha(x, \mathbf{t})$ must be true for every x.

Example 52 In Example 46 we derived:

$$c_{14} \quad [\![\Box \mid b \simeq x' \wedge \mathbf{T}^{\prec}_{\alpha}(x', a)]\!],$$

where α is the rooted formula $\mathfrak{I}^{\delta}(c_7, b \simeq a, \varepsilon)[z, x] = (q(z, x) \vee p(z, a))[z, x]$, and we computed:

$$\mathfrak{I}^{\delta}(c_{14}, b \simeq x', \varepsilon) = q(z, x) \vee p(z, a) \vee q(z, a) \stackrel{\text{def}}{=} \alpha'(z, x),$$

$$\mathfrak{X}^{\delta}(c_{14}, b \simeq x', \varepsilon) = \mathbf{T}^{\prec}_{\alpha}(z, a).$$

We observe that, according to Definition 48, $\alpha'(z, a) \models_s \alpha(z, a)$. This indicates that the invariant $\forall z\, \alpha(z, a)$ indeed propagates, and since $\mathfrak{X}^{\delta}(c_{14}, b \simeq x', \varepsilon)[x'/z] = \mathbf{T}^{\prec}_{\alpha}(x', a)$, the `Induction` rule derives the c-clause $[\![p(z, a) \vee q(z, a) \mid b \simeq x']\!]$. Afterwards, the unsatisfiable c-clause $[\![\Box \mid b \simeq x']\!]$ can be inferred by `Resolution` with $c_4 = [\![\neg p(y_4, y'_4) \mid b \simeq y_4]\!]$ and $c_5 = [\![\neg q(y_5, y'_5) \mid b \simeq y_5]\!]$.

The reasoning justifying this inference can be summarized as follows. As we shall see in Lemma 56, $S, \neg\alpha'(z, a) \models [\![\Box \mid \mathbf{T}^{\prec}_{\alpha}(z, a)]\!]$, meaning that if $\alpha'(z, a)$ is false in an I-normal model of S, then there exists a proper subterm z' of z such that $\alpha(z', a)$ does not hold in the model. But by Proposition 49 we also have $\alpha'(z, a) \models \alpha(z, a)$ hence $\neg\alpha(z, a) \models \neg\alpha'(z, a)$. We deduce that if $\alpha(z, a)$ is false in a model of S, then there exists a proper subterm z' of z such that $\alpha(z', a)$ is also false in that model. Since the

subterm relation is well-founded, this is possible only if $\alpha(z, a)$ is true for every term z, in every model of S.

Remark 53 As we shall see in the proof of Theorem 62, the conclusion of the `Induction` inference is a logical consequence not of the given premise, but rather of a set of its ancestors which are guaranteed to exist since the premise is derived from an initial clause set. However, the rule applies only when the premise is derived.

The `Iteration` rule.

We finally introduce the following rule, which, similarly to the `Trigger` rule, only aims at activating formulas. The purpose of this rule is to refine candidate invariants in the cases where the `Induction` rule does not apply.

`Iteration`:

$$\frac{\mathcal{C} : [\![C \mid f(\mathbf{u}) \simeq v[x]_p \wedge \mathcal{X} \wedge \mathbf{T}^{\prec}_{\alpha}(x, \mathbf{t})]\!]}{\text{Activate formula } \beta}$$

If:

1. $\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)$ is defined, x does not occur in C, $\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)$ and $\mathfrak{X}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)$ contain no occurrence of x and no selector function.
2. There is no substitution σ such that $\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[z]_p, p)[x/z]\sigma \models_s \alpha(x, \mathbf{t})\sigma$ and $x\sigma = x$.
3. $\beta = (\mathfrak{I}^{\delta}(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p) \vee \alpha(z, \mathbf{t}))[z, \mathbf{y}]$, where z is the fresh variable introduced in Definition 36 and $\mathbf{y}$ denotes the other variables occurring in β.

The `Iteration` rule applies when Condition 1 of the `Induction` rule does not hold, i.e., when property $\neg\mathbf{T}^{\prec}_{\alpha}(x, \mathbf{t})$ has been proven, but using additional hypotheses that do not occur in the invariant. In this case, the rule generates a new candidate invariant β by adding these hypotheses to the previous candidate. The rule applies iteratively until some fixpoint is reached (if any). See Example 65 for an example of the application of the `Iteration` rule.

Definition 54 We denote by CORE the set of core rules and by $\mathcal{IC}$ the calculus defined by the set of core rules enriched with the rules `Trigger`, `Domain Decomposition`, `Induction` and `Iteration`

Remark 55 In practice, the `Trigger`, `Induction` and `Iteration` rules should be applied only on c-clauses derived by the `Domain Decomposition` rule (as it is done in all examples in the paper and also in the completeness proof in Section 6). Actually, we could replace in a practical implementation the previous rules by derived inference rules combining one application of the `Domain Decomposition` rule with one application of the `Trigger`, `Induction` or `Iteration` rule (the current presentation is clearer and simpler to deal with from a theoretical point of view).

To summarize, a set of clauses S can be proved to be unsatisfiable by induction by:

- Deriving $\square$ under some assumptions on an inductive term.
- Generating a candidate invariant and applying the `Trigger` rule to attempt constructing a derivation showing this candidate propagates.

– If the candidate propagates, the `Domain Decomposition` and `Induction` rules permit to generate a universally quantified formula that is an inductive invariant of S; the core inference rules can then be applied to generate a refutation.

Since all the inference rules are sound (see Section 4), the refutation proves that S is indeed unsatisfiable.

4 Soundness of the Calculus

In this section, we show that $\mathcal{IC}$ is sound. To this purpose, we have to prove that the `Induction` rule is sound and that the `Domain Decomposition` rule is `I`-sound. For the `Domain Decomposition` rule, proving that it is sound is not sufficient, because this rule is intended to be used inside cycles: to make `Domain Decomposition` usable in the inferences corresponding to Case 3 of Definition 36, we must check that it is `I`-sound. Note that we do not need to consider the rules `Trigger` and `Iteration` because they only activate formulas. Moreover, `Induction` is *not* `I`-sound.

We first establish the following lemma, which states a key property of the candidate invariant $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)$:

Lemma 56 *Let δ be an inference tree. Let $\mathcal{C} = [\![C \mid \mathcal{X} \wedge \mathcal{Y}]\!]$ be a c-clause, where $\mathcal{X} = \bigwedge_{i=1}^{n} f(\mathbf{u}_i) \simeq v_i[t_i]_p$, and assume that η is an idempotent unifier of $(\mathbf{u}_1, v_1[t_1]_p), \ldots, (\mathbf{u}_n, v_n[t_n]_p)$ modulo $\mathcal{R}$, such that $x\eta$ is `I`-ground, for every variable x occurring in $\mathcal{C}$. Also let $t \stackrel{def}{=} t_1\eta{\downarrow_{\mathcal{R}}}$. If $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)$ is defined then:*

$$\forall^* A^\delta(\mathcal{C}, \mathcal{X}, p),\ \neg\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)[t/z]\eta \ \models\ \left[\!\!\left[C \;\middle|\; \mathfrak{X}^\delta(\mathcal{C}, \mathcal{X}, p)[t/z] \right]\!\!\right] \eta.$$

Proof First note that we can assume that the c-clauses occurring in the inference tree δ share no variables with t, except for variables occurring in either $\mathcal{C}$ or $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)$. Indeed, if this condition is not satisfied, then all the variables occurring in $\mathrm{var}(t) \setminus (\mathrm{var}(\mathcal{C}) \cup \mathrm{var}(\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)))$ can be replaced in δ by fresh, pairwise distinct, variables. The resulting inference tree δ' satisfies the required condition; it is clear that $\mathfrak{I}^{\delta'}(\mathcal{C}, \mathcal{X}, p) = \mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)$ since variables occurring in $\mathcal{C}$ and $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)$ are not renamed, and $\forall^* A^{\delta'}_{\top}(\mathcal{C}, \mathcal{X}, p) \equiv \forall^* A^\delta(\mathcal{C}, \mathcal{X}, p)$ because the c-clauses in $A^{\delta'}_{\top}(\mathcal{C}, \mathcal{X}, p)$ are renamings of those in $A^\delta(\mathcal{C}, \mathcal{X}, p)$.

We prove the result by induction on δ. Let $v \stackrel{\text{def}}{=} v_1[t_1]_p\eta{\downarrow_{\mathcal{R}}}$ and $\mathcal{I}$ be an `I`-normal model of $\forall^* A^\delta(\mathcal{C}, \mathcal{X}, p)$ and $\neg\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)[t/z]\eta$. According to Definition 36, we distinguish 3 cases.

1. Assume that $\mathcal{X} = \top$ (i.e., that $n = 0$). Then by Case 1 of Definition 36, we have $A^\delta(\mathcal{C}, \mathcal{X}, p) = \{\mathcal{C}\} = \{[\![C \mid \mathcal{Y}]\!]\}$ and $\mathfrak{X}^\delta(\mathcal{C}, \mathcal{X}, p) = \mathcal{Y}$. Since $\mathcal{I} \models \forall^* A^\delta(\mathcal{C}, \mathcal{X}, p)$, and $\mathcal{Y}$ contains no occurrence of z, necessarily $\mathcal{I} \models \left[\!\!\left[C \;\middle|\; \mathfrak{X}^\delta(\mathcal{C}, \mathcal{X}, p) \right]\!\!\right] [t/z]\eta$.
2. Assume that $\mathcal{X}$ is of the form $f(\mathbf{u}) \simeq v[y]_p$ (i.e., that $n = 1$ and $t_1 = y$), where y is a variable not occurring in $\mathcal{Y}$, $\mathbf{u}$ or v. Then by Case 2 of Definition 36, $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p) = \neg C[z/y]$ and $\mathfrak{X}^\delta(\mathcal{C}, \mathcal{X}, p) = \top$, hence $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)[t/z]\eta = \neg C[t/y]\eta = \neg C[y\eta{\downarrow_{\mathcal{R}}}/y]\eta$. Since η is idempotent, we have $y\eta{\downarrow_{\mathcal{R}}}\ \eta = y\eta{\downarrow_{\mathcal{R}}}$. Furthermore, since $\mathcal{I}$ is `I`-normal, by Condition 4 of Definition 6, we have $\mathcal{I} \models y\eta{\downarrow_{\mathcal{R}}} \simeq y\eta$. Thus $\mathcal{I} \models C\eta$, so that $\mathcal{I} \models \left[\!\!\left[C \;\middle|\; \mathfrak{X}^\delta(\mathcal{C}, \mathcal{X}, p) \right]\!\!\right] \eta$.

3. Otherwise, we have:

$$\begin{array}{lcl} A^\delta(\mathcal{C},\mathcal{X},p) & = & \bigcup_{D\in\delta(\mathcal{C})} A^\delta(\mathcal{D},\mathcal{X}_\mathcal{D},p), \\ \mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p) & = & \bigvee_{D\in\delta(\mathcal{C})} \mathfrak{I}^\delta(\mathcal{D},\mathcal{X}_\mathcal{D},p)\sigma, \\ \mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p) & = & \left(\mathcal{Y}' \wedge \bigwedge_{D\in\delta(\mathcal{C})} \mathfrak{X}^\delta(\mathcal{D},\mathcal{X}_\mathcal{D},p)\sigma\right)[z/v_1|_p,\ldots,z/v_k|_p], \end{array}$$

where σ is the inf-substitution of $\mathcal{C}$ and $\mathcal{Y}'$ is the inference constraint of $\mathcal{C}$. By Condition 3 of Definition 25, $\mathrm{dom}(\sigma)\cap\mathrm{var}(C)=\emptyset$, hence $x\sigma = x$ holds for every variable x occurring in $\mathcal{C}$. Furthermore, if $x \in \mathrm{var}(\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p))$ then necessarily $x\sigma = x$, because σ is idempotent by Condition 4 of Definition 25. Since we have assumed that the c-clauses occurring in the inference tree δ share no variables with t except those occurring in $\mathcal{C}$ or $\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p)$, we deduce that $t\sigma = t$. Let $\delta(\mathcal{C}) = \{\mathcal{H}_1,\ldots,\mathcal{H}_n\}$, where $\mathcal{H}_i = [\![H_i \mid \mathcal{X}_i]\!]$. By Condition 5 of Definition 25, there exists $I^\eta \subseteq [1,n]$ such that:

$$\bigwedge_{i\in I^\eta} H_i\sigma\eta \models [\![C \mid \mathcal{Y}']\!]\,\eta, \tag{1}$$

and if an atom $l \in \mathcal{X}_i$ with $i \in I^\eta$ is an antecedent of an atom $l' \in (\mathcal{X}\wedge\mathcal{Y})$ then:

$$l\sigma\eta{\downarrow_\mathcal{R}} = l'\eta{\downarrow_\mathcal{R}}\ . \tag{2}$$

Let $i \in I^\eta$. The constraint $\mathcal{X}_i$ is of the form $\mathcal{X}'_i \wedge \mathcal{Z}$, where $\mathcal{X}'_i$ is the conjunction of antecedents of atoms in $\mathcal{X}$. By definition of an antecedent, $\mathcal{X}'_i$ is necessarily an f-constraint. Since $\mathcal{I} \models \neg\mathfrak{I}^\delta(\mathcal{C},\mathcal{X},p)[t/z]\eta$, $\mathcal{H}_i \in \delta(\mathcal{C})$ and $\mathcal{X}'_i = \mathcal{X}_{\mathcal{H}_i}$, necessarily $\mathcal{I} \models \neg\mathfrak{I}^\delta(\mathcal{H}_i,\mathcal{X}'_i,p)\sigma[t/z]\eta$.
For every literal $f(\mathbf{u}') \simeq v'$ in $\mathcal{X}'_i$, we have by (2): $(f(\mathbf{u}') \simeq v')\sigma\eta\downarrow_\mathcal{R} \in \mathcal{X}\eta\downarrow_\mathcal{R}$, thus, if j is an arbitrary index in $[1,k]$ such that $f(\mathbf{u}') \simeq v'$ is an antecedent of $f(\mathbf{u}_j) \simeq v_j$, then $v'\sigma\eta\downarrow_\mathcal{R} = v_j[t_j]\eta\downarrow_\mathcal{R} = v$. This means that $\mathcal{X}'_i$ satisfies the same conditions as $\mathcal{X}$ (with the same term v and unifier $\sigma\eta$ modulo $\mathcal{R}$). Since $t\sigma = t$ and $\mathcal{I} \models \neg\mathfrak{I}^\delta(\mathcal{H}_i,\mathcal{X}'_i,p)\sigma[t/z]\eta$, we deduce that $\mathcal{I} \models \neg\mathfrak{I}^\delta(\mathcal{H}_i,\mathcal{X}'_i,p)[t/z]\sigma\eta$. But we also have $\mathcal{I} \models \forall^* A^\delta(\mathcal{H}_i,\mathcal{X}'_i,p)$ because $A^\delta(\mathcal{H}_i,\mathcal{X}'_i,p) \subseteq A^\delta(\mathcal{C},\mathcal{X},p)$, and therefore, by the induction hypothesis, $\mathcal{I} \models \left[\!\!\left[H_i \;\middle|\; \mathfrak{X}^\delta(\mathcal{H}_i,\mathcal{X}'_i,p)[t/z] \right]\!\!\right]\sigma\eta$. Now, since $\mathfrak{X}^\delta(\mathcal{H}_i,\mathcal{X}'_i,p)[t/z]\sigma \subseteq \mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p)[t/z]$, we have $\mathcal{I} \models \left[\!\!\left[H_i\sigma \;\middle|\; \mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p) \right]\!\!\right][t/z]\eta$. This relation holds for every $i \in I^\eta$, we thus deduce:

$$\mathcal{I} \models \left[\!\!\left[\bigwedge_{i\in I^\eta} H_i\sigma \;\middle|\; \mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p)[t/z] \right]\!\!\right]\eta. \tag{3}$$

Since $t_1\eta\ {\downarrow_\mathcal{R}} = \cdots = t_n\eta\ {\downarrow_\mathcal{R}} = t$, $\mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p)[t/z]\eta$ is equivalent to $\mathcal{Y}' \wedge \bigwedge_{D\in\delta(\mathcal{C})} \mathfrak{X}^\delta(\mathcal{D},\mathcal{X}_\mathcal{D},p)\sigma\eta$. We deduce, from (3) and (1) that $\mathcal{I} \models \left[\!\!\left[C \;\middle|\; \mathfrak{X}^\delta(\mathcal{C},\mathcal{X},p)[t/z] \right]\!\!\right]\eta$.

We now prove that the `Domain Decomposition` rule is `I`-sound. In order to do so, we need to exhibit the inf-substitution associated with the conclusion, according to Definition 25. But the variables in the premises represent subterms of the terms denoted by the variables in the conclusion of the rule, thus this substitution cannot be defined using standard terms: this is why we use the selector functions f_i^{-1}, defined in Section 2.2.

Definition 57 A position p is a *constructor-position* in a term t if p is a position in t and for every proper prefix q of p, the head symbol of $t|_q$ is a constructor.

In particular, if t is $\mathtt{I}$-ground, then any position in t is a constructor-position.

Definition 58 Let t be an $\mathtt{I}$-ground term, let p be a constructor-position in t and let s be a term. The term $SubT(t,p,s)$ is inductively defined as follows.

- If $p = \epsilon$ then $SubT(t,p,s) \stackrel{\text{def}}{=} s$.
- If $p = q.i$ and $t|_q = f(t_1,\ldots,t_n)$ then $SubT(t,p,s) \stackrel{\text{def}}{=} f_i^{-1}(SubT(t,q,s))$.

Example 59 Consider the terms $t \stackrel{\text{def}}{=} f(g(x))$ and $s \stackrel{\text{def}}{=} f(g(a))$. Then $SubT(t,1.1,s) = g_1^{-1}(f_1^{-1}(f(g(a))))$.

Proposition 60 *For any term t, constructor-position p of t and instance s of t, $SubT(t,p,s)\downarrow_{\mathcal{R}} = s|_p\downarrow_{\mathcal{R}}$.*

Proof The proof is by induction on p. If $p = \epsilon$ then $SubT(t,p,s) = s$ and the proof is immediate. If $p = q.i$ then $SubT(t,p,s) = f_i^{-1}(SubT(t,q,s))$ with $t|_q = f(t_1,\ldots,t_n)$ hence by the induction hypothesis $SubT(t,p,s)\downarrow_{\mathcal{R}} = f_i^{-1}(s|_q)\downarrow_{\mathcal{R}}$. Since s is an instance of t we have $s|_q = f(s_1,\ldots,s_n)$ and $s|_p = s_i$. Hence $SubT(t,p,s)\downarrow_{\mathcal{R}} = f_i^{-1}(f(s_1,\ldots,s_n)\downarrow_{\mathcal{R}} = s_i\downarrow_{\mathcal{R}}$, since by definition of $\mathcal{R}$, $f_i^{-1}(f(x_1,\ldots,x_n))\downarrow_{\mathcal{R}} = x_i$.

Lemma 61 *The* `Domain Decomposition` *rule is* $\mathtt{I}$*-sound.*

Proof We employ the same notations as in the definition of the rule page 25. Let $\mathcal{C}_i \stackrel{\text{def}}{=} [\![C_i \mid \mathcal{X}_i \wedge \beta_i(t_i)]\!]$ for $i = 1,\ldots,n$, and $\mathcal{D} \stackrel{\text{def}}{=} [\![C_1 \vee \cdots \vee C_n \mid \mathcal{X}_1 \wedge \cdots \wedge \mathcal{X}_n \wedge \beta_1(x)]\!]\,\eta$. We assume that the $\mathcal{C}_1,\ldots,\mathcal{C}_n$ (hence the terms $t_1,\ldots,t_n$) are variable-disjoint.

The inference constraint $\mathcal{Y}'$ is $\top$ and the antecedent relation is defined in a natural way, i.e., any equation $l \in \mathcal{X}_i \wedge \beta_i(t_i)$ is an antecedent of the corresponding equation $l\eta \in \mathcal{X}_i\eta \wedge \beta_i(x)\eta$. The inf-substitution is defined as follows. Let $j \in [1,n]$, and σ_j be the substitution mapping each variable y in t_j to the term $SubT(t_j,p_j,x)$, where p_j is an arbitrarily chosen position such that $t_j|_{p_j} = y$. Note that p_j is necessarily a constructor position since t_j is a constructor term. We let: $\sigma \stackrel{\text{def}}{=} \eta(\bigcup_{j=1}^{n}\sigma_j)$.

Let θ be an $\mathtt{I}$-ground substitution of the variables in $\mathcal{D}$. Since $\{t_1,\ldots,t_n\}$ is covering, there exists an element i of $[1,n]$ and a substitution γ such that $x\theta = t_i\gamma$. We let: $I^\theta \stackrel{\text{def}}{=} \{i\}$. Then for every variable y occurring at a position p_i in t_i, we have $y\sigma = y\eta\sigma_i = y\sigma_i$. Indeed, by definition of the `Domain Decomposition` rule, since y is a variable of t_i, it cannot occur in D_j or $\mathcal{X}_j$ for $j = 1,\ldots,n$, thus $y\eta = y$. We deduce that $y\sigma = SubT(t_i,p_i,x)$, and $y\sigma\theta = SubT(t_i,p_i,x\theta) = SubT(t_i,p_i,t_i\gamma)$. By Proposition 60, $y\sigma\theta\downarrow_{\mathcal{R}} = t_i\gamma|_{p_i}\downarrow_{\mathcal{R}} = (t_i|_{p_i})\gamma\downarrow_{\mathcal{R}} = y\gamma\downarrow_{\mathcal{R}}$ We deduce that $y\sigma\theta\downarrow_{\mathcal{R}} = y\gamma\downarrow_{\mathcal{R}}$ for every $y \in \mathrm{var}(t_i)$, so that $t_i\sigma\theta\downarrow_{\mathcal{R}} = t_i\gamma\downarrow_{\mathcal{R}} = x\theta\downarrow_{\mathcal{R}}$. Since the variables in t_i do not occur in C_j, β_j or $\mathcal{X}_j$ (for $j = 1,\ldots,n$), by construction of σ we have $\beta_i\sigma\theta = \beta_i\eta\theta$ and $\mathcal{X}_i\sigma\theta = \mathcal{X}_i\eta\theta$ (this also entails that the conclusion of the rule cannot contain variables in $\mathrm{dom}(\sigma)$ and that σ is idempotent, since η is idempotent). Since by definition of the rule we have $\beta_i\eta \subseteq \beta_1\eta$, we deduce that $\beta_i(t_i)\sigma\theta\downarrow_{\mathcal{R}} \subseteq \beta_1(x)\eta\theta\downarrow_{\mathcal{R}}$. Therefore, all the conditions of Definition 25 are satisfied.

Theorem 62 *The* `Induction` *rule is sound.*

Proof Let δ be an inference tree, $\mathcal{C}$ be a c-clause, $\mathcal{S}'$ be a set of c-clauses deduced from $\mathcal{C}$ by applying the `Induction` rule and $\mathcal{S} = \left[\!\!\left[C \vee \mathfrak{I}^\delta(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z]\sigma \;\middle|\; \mathcal{Y}\sigma \right]\!\!\right]$. By definition of the rule, $\mathcal{C}$ and $\mathcal{S}'$ are respectively of the form $[\![C \mid f(\mathbf{u}) \simeq v[x]_p \wedge \mathcal{X} \wedge \mathbf{T}_\alpha^{\prec}(x, \mathbf{t})]\!]$ and $[\![C \vee \alpha(x, \mathbf{t})\sigma \mid \mathcal{Y}\sigma]\!]$. Recall that, by definition, $A^\delta(\mathcal{C}, f(\mathbf{u}), p)$ is a set of ancestors of $\mathcal{C}$. Assume there exists a model $\mathcal{I}$ of $\forall^* A^\delta(\mathcal{C}, f(\mathbf{u}), p)$ that is a counter-model of $\mathcal{S}'$. By Condition 1 of the `Induction` rule and Proposition 49, we have $(\forall \mathbf{y}\, \mathfrak{I}^\delta(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p))[x/z]\sigma \models \alpha(x, \mathbf{t})\sigma$, and since the variables in $\mathbf{y}$ do not occur in $C\sigma$, this entails that $\mathcal{I}$ is also a counter-model of $\mathcal{S}$. By Proposition 30, there exists an $\mathtt{I}$-ground substitution θ and an interpretation $\mathcal{J}$ coinciding with $\mathcal{I}$ on every symbol occurring in $\forall^* A^\delta(\mathcal{C}, f(\mathbf{u}), p)$ and $\mathcal{S}$, such that $\mathcal{J}(y) = \mathcal{J}(y\theta)$ for every variable y occurring in $\mathcal{S}$, and for every $y \in \mathrm{dom}(\theta)$, $y\theta$ contains no variables occurring in $\forall^* A^\delta(\mathcal{C}, f(\mathbf{u}), p)$ or in $\mathcal{S}$. We have $\mathcal{J} \models \forall^* A^\delta(\mathcal{C}, f(\mathbf{u}), p)$ and $\mathcal{J} \not\models \mathcal{S}\theta$. Consequently, $\mathcal{J} \models \mathcal{Y}\sigma\theta$, $\mathcal{J} \not\models \mathfrak{I}^\delta(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z]\sigma\theta$ and $\mathcal{J} \not\models C\sigma\theta$.

Assume that there is an element m that is minimal w.r.t. $\lhd$ such that there exists a vector of elements $\mathbf{e}$ of the same sorts as $\mathbf{y}$ with $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m/x, \mathbf{e}/\mathbf{y}] \models \neg\mathfrak{I}^\delta(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z]\sigma\theta$. Since $\mathcal{J} \models \forall^* A^\delta(\mathcal{C}, f(\mathbf{u}), p)$, we have $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m/x, \mathbf{e}/\mathbf{y}] \models \forall^* A^\delta(\mathcal{C}, f(\mathbf{u}), p)$ and it is clear that $\sigma\theta$ is an idempotent unifier of $\{x\}$ modulo $\mathcal{R}$. By Lemma 56, this entails that $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m/x, \mathbf{e}/\mathbf{y}] \models \left[\!\!\left[C \;\middle|\; \mathfrak{X}^\delta(\mathcal{C}, \mathcal{X}, p)[x/z] \right]\!\!\right]\sigma\theta$, and therefore $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m/x, \mathbf{e}/\mathbf{y}] \models [\![C\sigma\theta \mid \mathcal{Y}\sigma\theta \wedge \mathbf{T}_\alpha^{\prec}(x, \mathbf{t})\sigma\theta]\!]$ by definition of the constraint $\mathcal{Y}$. By Condition 5, the variables in $\mathbf{y}$ cannot occur in $C\sigma\theta$ or $\mathcal{Y}\sigma\theta$. By Conditions 1, 2 and 3, the same property holds for the variable x. Since $\mathcal{J} \models \neg C\sigma\theta, \mathcal{Y}\sigma\theta$, we deduce that $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m/x, \mathbf{e}/\mathbf{y}] \models \neg C\sigma\theta, \mathcal{Y}\sigma\theta$. Necessarily, $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m/x, \mathbf{e}/\mathbf{y}] \models [\![\Box \mid \mathbf{T}_\alpha^{\prec}(x, \mathbf{t})]\!]\sigma\theta$, which means that there exists a term $m' \lhd m$ such that $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m'/x, \mathbf{e}/\mathbf{y}] \models \neg\mathbf{T}_\alpha(x, \mathbf{t})\sigma\theta$, hence such that $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m'/x, \mathbf{e}/\mathbf{y}] \models \neg\alpha(x, \mathbf{t})\sigma\theta$. By Condition 1 and Proposition 49, $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m'/x, \mathbf{e}/\mathbf{y}] \models \neg(\forall \mathbf{y}\, \mathfrak{I}^\delta(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p))[x/z]\sigma\theta$, thus there exists $\mathbf{e'}$ such that $\mathcal{J}\frac{\mathbf{m}}{\mathbf{m}}[m'/x, \mathbf{e'}/\mathbf{y}] \models \neg\mathfrak{I}^\delta(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z]\sigma\theta$ which contradicts the minimality of m.

Consequently $\mathcal{J} \models \forall x, \mathbf{y}\ \mathfrak{I}^\delta(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z]\sigma\theta$, and in particular, $\mathcal{J} \models \mathfrak{I}^\delta(\mathcal{C}, f(\mathbf{u}) \simeq v[x]_p, p)[x/z]\sigma\theta$, which contradicts our hypotheses.

Remark 63 It is clear that the calculus has a huge search space, and the unrestricted application of the rules is not practical for automated proof search. A first natural way to reduce the branching is to apply the rules only on empty clauses (i.e., add the condition $C = \Box$ (resp. $C_i = \Box$) to the `Trigger`, `Iteration`, `Induction` (resp. `Domain Decomposition`) rules. This entails a loss of generality only if induction is also required to refute C, otherwise the refutation of C could be included into the considered inference tree. A second way to strongly reduce the search space is to restrict the application of `Abstraction`, that selects the term on which induction can be performed. For instance, one may select a single constant or Skolem term in the initial formula, or apply the rule only if the function f is not completely defined (for instance in Example 66, $x + y$ is defined for all ground terms x and y whereas n and m are undefined). We may further restrict the induction terms by taking $p = \varepsilon$ in the `Trigger`, `Iteration` and `Induction` rules. Another option consists in applying the `Trigger` rule only if the same candidate invariant is repeatedly generated. This typically happens when multiple instances of the same formula can be proven, which naturally suggests that it could be

proven by induction (if $p(t_1), \ldots, p(t_n)$ can be proven, then it is worth trying to derive $\forall x\, p(x)$).

Finally, in the context of interactive theorem proving, one could let a human user suggest and select candidate invariants. This means that `Trigger` would be dismissed, leaving to the user the burden of activating some formulas. The procedure would then check that the proposed formula is provable and refine it is needed.

The presented calculus uses the core inference rules for two distinct purposes: first to *prove* that a formula is an inductive invariant, and second to *generate* this formula. For the first part, any refutationally complete calculus is (theoretically) sufficient, whereas for the second part, less restrictive calculi may prove useful: if more clauses are generated, it is more likely that an inductive invariant will be obtained. This suggests that the two aspects could be disconnected: very restrictive core rules could be employed, coupled with any non-restrictive deductive procedure to generate input clauses (but such non-restrictive inferences would not occur inside inference trees).

5 Examples

We illustrate how the calculus works on some simple examples, that have been chosen to illustrate various features of the method. The first example is explained in full detail and involves a binary constructor. Although quite simple, it is thus out of the scope of the method described in [24,23], which only handles unary functions.

Example 64 Consider the clause set:

$$\begin{array}{ll} 1 & p(a) \\ 2 & \neg p(x_1) \vee \neg p(x_2) \vee p(f(x_1, x_2)) \end{array}$$

where $a : \tau$ and $f : \tau \times \tau \to \tau$ are the only constructors. The goal is to prove that $\forall x\, p(x)$ holds. To this purpose, we add the c-clause:

$$3 \quad [\![\neg p(y) \mid b \simeq y]\!]$$

to the set (where b is a fresh – non-constructor – symbol of profile τ) in order to derive a contradiction. First, the base case is handled by applying the `Resolution` rule:

$$4 \quad [\![\Box \mid b \simeq a]\!] \quad \text{(res, 1, 3)}$$

meaning that the set is unsatisfiable if b is equal to a. Then, the `Trigger` rule applies on 4: C and $\mathcal{X}$ are empty, $f(\mathbf{u})$ is b, v is arbitrary and p is ε. We compute the set $\mathfrak{I}^\delta(4, b \simeq a, \varepsilon)$. The two parents of 4 are 1 and 3, and the only antecedent of $b \simeq a$ is the literal $b \simeq y$ in 3, thus: $\mathfrak{I}^\delta(4, b \simeq a, \varepsilon) = \neg\neg p(z) \equiv p(z)$ and $\mathfrak{X}^\delta(4, b \simeq a, \varepsilon) = \top$. Consequently, the rooted formula $\alpha = p(z)[z]$ is activated by the `Trigger` rule. By applying the `Resolution` rule again, we get:

5	$[\![\neg p(x_1) \vee \neg p(x_2) \mid b \simeq f(x_1, x_2)]\!]$	(res, 2, 3)
6	$[\![\neg p(x_1) \vee \neg \mathbf{T}_\alpha(x_2) \mid b \simeq f(x_1, x_2)]\!]$	(res, $\Gamma_{\mathbf{T}}$, 5)
7	$[\![\neg \mathbf{T}_\alpha(x_1) \vee \neg \mathbf{T}_\alpha(x_2) \mid b \simeq f(x_1, x_2)]\!]$	(res, $\Gamma_{\mathbf{T}}$, 6)
8	$[\![\neg \mathbf{T}_\alpha(x_1) \vee x_2 \not\prec y_2 \mid b \simeq f(x_1, x_2) \wedge \mathbf{T}^\prec_\alpha(y_2)]\!]$	(res, 7, $\Gamma_{\mathbf{T}^\prec}$)
9	$[\![x_1 \not\prec y_1 \vee x_2 \not\prec y_2 \mid b \simeq f(x_1, x_2) \wedge \mathbf{T}^\prec_\alpha(y_1) \wedge \mathbf{T}^\prec_\alpha(y_2)]\!]$	(res, 8, $\Gamma_{\mathbf{T}^\prec}$)
10	$[\![x_1 \not\prec y_1 \mid b \simeq f(x_1, x_2) \wedge \mathbf{T}^\prec_\alpha(y_1) \wedge \mathbf{T}^\prec_\alpha(f(x'_1, x_2))]\!]$	(res, 9, $\Gamma^f_\prec$)
11	$[\![\Box \mid b \simeq f(x_1, x_2) \wedge \mathbf{T}^\prec_\alpha(f(x_1, x_2))]\!]$	(res, 10, $\Gamma^f_\prec$ + c-fact)

In the constraint of c-clause 11, the conjunction $\mathbf{T}^{\prec}_{\alpha}(f(x'_1, x_2)) \wedge \mathbf{T}^{\prec}_{\alpha}(f(x_1, x'_2))$ is factorized into $\mathbf{T}^{\prec}_{\alpha}(f(x_1, x_2))$. At this point, the `Domain Decomposition` rule applies on c-clauses 11 and 4, with the covering set $\{f(x_1, x_2), a\}$ and the formulas $\beta_1 = (b \simeq x \wedge \mathbf{T}^{\prec}_{\alpha}(x))[x]$ and $\beta_2 = (b \simeq x)[x]$, yielding:

$$13 \quad [\![\Box \mid b \simeq x \wedge \mathbf{T}^{\prec}_{\alpha}(x)]\!]$$

The set $\mathfrak{I}^{\delta}(13, b \simeq x, \varepsilon)$ is computed by collecting ancestors of 13 that contain antecedents of $b \simeq x$ of the form $b \simeq x'$, for some variable x', and considering the disjunction of the negation of the clause part of these ancestors (substituting x' by a fresh variable z). The only such ancestor is 3 (the details of the computation are omitted for conciseness and readability), thus $\mathfrak{I}^{\delta}(13, b \simeq x, \varepsilon) = \neg\neg p(z) \equiv p(z)$ and $\mathfrak{X}^{\delta}(13, b \simeq x, \varepsilon) = \top$. The conditions of the `Induction` rule are thus trivially satisfied, and the rule can be applied to generate

$$14 \quad [\![p(z) \mid \top]\!] \quad (\texttt{Induction}, 13)$$

By `Resolution` with c-clause 3 and `Instantiation`, we get $[\![\Box \mid b \simeq z]\!]$ and $\Box$, which shows that the initial clause set is indeed unsatisfiable.

We now give a example in which the initial induction invariant does not propagate, but can be refined by applying the `Iteration` rule.

Example 65 Consider the following clause set, in which a and f are the only constructors:

$$\begin{array}{ll} 1 & p(a) \\ 2 & \neg p(x) \vee p(f(x)) \vee q(f(x)) \\ 3 & \neg q(x) \vee p(f(x)) \vee q(f(x)) \\ 4 & [\![\neg p(y) \mid b \simeq y]\!] \\ 5 & [\![\neg q(y) \mid b \simeq y]\!] \\ 6 & [\![\neg r(y) \mid b \simeq y]\!] \end{array}$$

The last three c-clauses correspond to the skolemized form of the negation of the goal $\forall y\,(p(y) \vee q(y) \vee r(y))$. The c-clause $7 : [\![\Box \mid b \simeq a]\!]$ can be derived, and, exactly as in the previous case, the `Trigger` rule is applied on the formula $p(z)$ meaning that the calculus attempts to prove that $\forall z\, p(z)$ holds – which is clearly unfeasible and will fail as will shall see. The c-clause $8 : \left[\!\!\left[\Box \;\middle|\; b \simeq x \wedge \mathbf{T}^{\prec}_{p(z)[z]}(x)\right]\!\!\right]$ can be derived in a similar way to the c-clause 13 in the previous example, however, this time, c-clause 5 must be used, which entails that $\mathfrak{I}^{\delta}(8, b \simeq x, \varepsilon) = (\neg\neg p(z) \vee \neg\neg q(z)) \equiv (p(z) \vee q(z)) \not\models_s p(z)$. Therefore, Condition 1 in the definition of the `Induction` rule does not hold: the invariant does not propagate and the previous attempt fails. We must thus apply the `Iteration` rule to generate a new candidate invariant. The rule activates the formula $\beta = (p(z) \vee q(z) \vee p(z))[z] \equiv (p(z) \vee q(z))[z]$, meaning that the calculus attempts to prove that $\forall z\,(p(z) \vee q(z))$ holds. Afterwards the clause $[\![\Box \mid b \simeq x]\!]$ can be derived in the same way as in the previous example. Note that the correct invariant can only derived by using information extracted from the failure of the first one (it is clear that trying to apply the induction scheme on the initial formula $\forall y\,(p(y) \vee q(y) \vee r(y))$ would also fail).

We now show how to prove the commutativity of addition using the superposition calculus [2,27]. Although simple, this example is interesting for two reasons: first it

requires nested induction (induction is required both in the base and inductive cases of the main induction scheme), and second its uses an auxiliary lemma, namely $x + succ(y) \simeq succ(x + y)$, which is generated automatically by the calculus.

Example 66 We consider the following clause set:

$$\begin{array}{ll} 1 & 0 + x \simeq x \\ 2 & succ(x) + y \simeq succ(x + y) \\ 3 & [\![x + y \not\simeq y + x \mid n \simeq x \wedge m \simeq y]\!] \end{array}$$

We first derive clause $4 : [\![x + 0 \not\simeq x \mid n \simeq x \wedge m \simeq 0]\!]$, from c-clauses 1 and 3. Then the clause $5 : [\![\Box \mid n \simeq x \vee m \simeq 0]\!]$ can be derived by simple induction, as in the previous examples: this is done by first generating $[\![\Box \mid n \simeq 0 \wedge m \simeq 0]\!]$ using the usual rules, then applying the `Trigger` rule to activate formula $x + 0 \simeq x$, and finally showing that the invariant indeed propagates and applying the `Induction` rule. The `Trigger` rule applies on 5, with the formula $\neg(x+y \not\simeq y+x) \equiv x+y \simeq y+x$, activating the formula $\alpha \stackrel{\text{def}}{=} (x + y \simeq y + x)[y, x]$. Intuitively, this means that the calculus tries to prove that $x + y \simeq y + x$ holds by induction on y (having already proven the base case). By combining the axiom $[\![x + y \simeq y + x \vee \neg \mathbf{T}_\alpha(y, x) \mid \top]\!]$ of $\Gamma_{\mathbf{T}}$ associated with the newly activated formula α with c-clauses 2 and 3, we derive by superposition, unifying y with $succ(y')$:

$$6 : [\![x + succ(y') \not\simeq succ(x + y') \vee \neg \mathbf{T}_\alpha(y', x) \;\big|\; n \simeq x \wedge m \simeq succ(y')]\!]$$

From 6, $\Gamma^f_\prec$ and the axiom $[\![y' \not\prec y \vee \mathbf{T}_\beta(y', x) \;\big|\; \mathbf{T}^\prec_\alpha(y, x)]\!]$ of $\Gamma_{\mathbf{T}^\prec}$, we obtain:

$$7 : [\![x + succ(y') \not\simeq succ(x + y') \;\big|\; n \simeq x \wedge m \simeq succ(y') \wedge \mathbf{T}^\prec_\alpha(succ(y'), x)]\!]$$

Using c-clause 1 twice and unifying x with 0, we get:

$$8 : \left[\!\!\left[\Box \;\middle|\; n \simeq 0 \wedge m \simeq succ(y') \wedge \mathbf{T}^\prec_\alpha(succ(y'), 0)\right]\!\!\right]$$

Again, the `Trigger` rule applies, this time with the formula $\beta \stackrel{\text{def}}{=} (x + succ(y') \simeq succ(x + y'))[x, y']$. This formula can be viewed as a lemma, that will be proven by induction on x. From the axiom $x' + succ(y') \simeq succ(x' + y') \vee \neg \mathbf{T}_\beta(x', y')$ of $\Gamma_{\mathbf{T}}$ and from c-clauses 2 and 7, we eventually derive (unifying x with $succ(x')$):

$$9 : [\![\neg \mathbf{T}_\beta(x', y') \;\big|\; \wedge n \simeq succ(x') \wedge m \simeq succ(y') \wedge \mathbf{T}^\prec_\alpha(succ(y'), succ(x'))]\!]$$

Using the axioms in $\Gamma^f_\prec$ and $\Gamma_{\mathbf{T}^\prec}$, this yields:

$$10 : \left[\!\!\left[\Box \;\middle|\; n \simeq succ(x') \wedge m \simeq succ(y') \wedge \mathbf{T}^\prec_\alpha(succ(y'), succ(x')) \wedge \mathbf{T}^\prec_\beta(succ(x'), y')\right]\!\!\right]$$

Together with clause 8, this entails (by `Domain Decomposition`, with the covering set $\{0, succ(x')\}$):

$$11 : \left[\!\!\left[\Box \;\middle|\; n \simeq x \wedge m \simeq succ(y') \wedge \mathbf{T}^\prec_\alpha(succ(y'), x) \wedge \mathbf{T}^\prec_\beta(x, y)\right]\!\!\right]$$

and the `Induction` rule applies, yielding:

$$12 : [\![x + succ(y') \simeq succ(x + y') \;\big|\; \top]\!]$$

From 12 and 7, we derive:

$$13 : \left[\!\left[\Box \;\middle|\; n \simeq x \wedge m \simeq succ(y') \wedge \mathbf{T}_\alpha^{\prec}(succ(y'), x) \right]\!\right]$$

Using the c-clauses 13 and 5, the `Domain Decomposition` rule can be applied with the covering set $\{0, succ(y')\}$, yielding $14 : [\![\Box \mid n \simeq x \wedge m \simeq y \wedge \mathbf{T}_\alpha^{\prec}(y, x)]\!]$. The corresponding candidate invariant $\mathfrak{I}^\delta(14, m \simeq y, \varepsilon)$ is α. Thus the `Induction` rule applies, yielding: $[\![x + y \simeq y + x \mid \top]\!]$. Together with c-clause 3, this yields the unsatisfiable c-clause: $[\![\Box \mid n \simeq x \wedge m \simeq y]\!]$.

It is interesting to note that this example is handled without having to supply or synthesize any additional lemmata (as it is done for instance in [22,13]). All the required lemmata are automatically generated.

We also give an example involving quantifier alternation. This example is interesting because the correct instantiation can only be obtained from the inductive step. Trying to prove the initial property by induction would fail.

Example 67 Let S be the following clause set, defining an operator $\oplus$ adding a natural number to all elements in a list (the additional axioms for addition are irrelevant and omitted):

$$\begin{array}{ll} 1 & x \oplus nil \simeq nil \\ 2 & x \oplus cons(x', y) \simeq cons(x + x', x \oplus y) \\ 3 & 0 + x \simeq x \end{array}$$

The set of constructors of range `list` is $\{nil : \texttt{list}, cons : \texttt{nat} \times \texttt{list} \to \texttt{list}\}$. Note that in this example `nat` does not necessarily belong to `I`, thus 0 can be either a constructor or a standard symbol. We want to prove that that $\oplus$ admits a left-neutral element, i.e., that $\exists x \forall y\ x \oplus y \simeq y$. Proceeding by contradiction, we add the c-clause $4 : [\![x \oplus y \not\simeq y \mid f(x) \simeq y]\!]$ and try to derive an unsatisfiable c-clause of the form $[\![\Box \mid f(x) \simeq y]\!]$ (x then denotes the left-neutral element). The symbol f denotes a new, non-constructor, symbol of profile $\texttt{nat} \to \texttt{list}$. Using 4 and 1 we derive: $5 : [\![\Box \mid f(x) \simeq nil]\!]$. This means that the base case always holds, regardless of the value of x. At this point the `Trigger` rule applies, activating the formula $\alpha = (x \oplus y \simeq y)[y, x]$. This indicates that the calculus tries to prove that $\forall y\ x \oplus y \simeq y$ holds, which, clearly, should succeed only if $x = 0$. By several steps of superposition using the axioms $\Gamma_{\mathbf{T}}$, $\Gamma_{\mathbf{T}^\prec}$ and $\Gamma^f_{\prec}$, we eventually derive:

$$6 : \left[\!\left[\Box \;\middle|\; f(0) \simeq cons(x', y) \wedge \mathbf{T}_\alpha^{\prec}(cons(x', y), 0) \right]\!\right]$$

Together with 5, this yields, by the `Domain Decomposition` rule:

$$7 : \left[\!\left[\Box \;\middle|\; f(0) \simeq z \wedge \mathbf{T}_\alpha^{\prec}(z, 0) \right]\!\right]$$

Intuitively, this means that the inductive invariant propagates, but only in the case where $x = 0$. Then the `Induction` rule applies as in the previous cases, yielding:

$$8 : [\![0 \oplus z \simeq z \mid \top]\!]$$

and by `Resolution` with c-clause 4:

$$9 : [\![\Box \mid f(0) \simeq z]\!]$$

This c-clause provides the answer to the original question, namely that the left-neutral element is $x = 0$.

The following example is taken[15] from [11]. It is interesting because it involves an inductive invariant containing a universal quantifier (see also Remark 51).

Example 68 We consider the clause set:

$$\begin{array}{ll} 1 & r(0,y) \\ 2 & \neg r(x,0) \vee r(s(x),0) \\ 3 & \neg r(s(s(x)),y) \vee r(s(x),s(y)) \\ 4 & [\![\neg r(x,y) \mid n \simeq x \wedge m \simeq y]\!] \end{array}$$

It is easy to check that clause $5 : r(x,0)$ can be generated from clauses 1, 2 and 4, as in the previous examples. Afterwards, we may derive $6 : [\![\Box \mid n \simeq x \wedge m \simeq 0]\!]$ from 5 and 4, and the `Trigger` rule thus applies on the candidate invariant $\alpha = r(x,y)[y,x]$. Then the c-clause $7 : [\![\Box \mid \mathbf{T}^{\prec}_{\alpha}(y, s(s(x))) \wedge n \simeq s(s(x)) \vee m \simeq y]\!]$ can be obtained using clauses 3, 4 and 5, with $\mathfrak{I}^{\delta}(7, m \simeq y, \varepsilon) = r(s(x), z)$ and $\mathfrak{X}^{\delta}(7, m \simeq y, \varepsilon) = \mathbf{T}^{\prec}_{\alpha}(z, s(s(x)))$. We have $\forall x\, r(s(x), y) \models_s r(s(s(x)), y)$, hence the `Induction` rules applies (with $\mathbf{y} = (x)$), yielding $8 : r(s(x), y)$. Finally, the empty clause can be generated from 1, 8 and 4 by straightforward applications of the `Resolution` and `Domain Decomposition` rules. Note that there is no substitution σ such that $r(s(x), y)\sigma \models_s r(s(s(x)), y)\sigma$, which means that the `Induction` rule cannot be applied on 7 with a non-empty vector $\mathbf{y}$. Intuitively, the formula $\forall x\, r(s(x), y)$ can be proven by induction on y but *not* the formula $r(s(x), y)$.

The final example illustrates how the `Induction` rule can be applied under some conditions (stated as literals).

Example 69 We consider the clause set (taken from [37]):

$$\begin{array}{ll} 1 & p(0) \\ 2 & p(s(x)) \vee \neg p(x) \vee \neg q(x, s(x)) \\ 3 & q(x,0) \\ 4 & q(x, s(y)) \vee \neg q(x,y) \vee \neg p(x) \\ 5 & [\![\neg p(x) \vee \neg q(x,y) \mid n \simeq x \wedge m \simeq y]\!] \end{array}$$

First the clause $6 : [\![\neg p(x) \mid n \simeq x \wedge m \simeq 0]\!]$ is generated from 3 and 5, which activates the formula $\alpha = (p(x) \wedge q(x,y))[y,x]$. Then the c-clause $[\![\neg p(x) \mid \mathbf{T}^{\prec}_{\alpha}(y,x)]\!]$ can be derived from 3, 4 and 5. From this, the c-clause $7 : \neg p(x) \vee q(x,y)$ can be derived[16] by `Induction`. Afterwards, the clause $8 : p(s(x)) \vee \neg p(x)$ can be derived from 7 and 2, and the clause $p(x)$ is easily derived from 8, 1, 7 and 5, yielding an immediate contradiction.

6 A Completeness Result

The calculus $\mathcal{IC}$ cannot be refutationally complete. Indeed, the collection of sets of c-clauses that are unsatisfiable is not recursively enumerable, even if Λ only contains

[15] For the sake of readability the predicate N describing natural numbers in the original formulation is omitted.

[16] Note that we also generate the clause $\neg p(x) \vee p(x)$, which is a tautology. Intuitively, the invariant $p(x) \wedge q(x,y)$, is proven by induction on y, under the condition $p(x)$.

a unique constant symbol (see, e.g., [24]). In this section we provide restricted completeness results. More precisely, we assume (informally speaking) that the clausal representation of a particular kind of inductive invariant exists in the considered set of c-clauses, either because it already occurs in the initial set or because it is generated at some point by the inference rules. We show that, under this condition, and assuming that the core inference rules are *liftable* (see below) and refutationally complete in the usual sense, the rules in $\mathcal{IC}$ can be used to extract this invariant from the set and use it to derive the empty clause. Of course, in general there is no guarantee that such an invariant can be generated, and even the *existence* of such an invariant cannot be ensured.

Definition 70 A substitution σ is an $\mathtt{I}$-*substitution* if every $x \in \mathrm{dom}(\sigma)$ is of a sort in $\mathtt{I}$. We denote by $\mathit{Inst}(S)$ (resp. $\mathit{Inst}_{\mathtt{I}}(S)$) the set of c-clauses of the form $\mathcal{C}\sigma$, where $\mathcal{C} \in S$ and σ is a substitution (resp. an $\mathtt{I}$-substitution).

Definition 71 For all objects (terms, formulas, substitutions, ...) t and s, we write $s \succeq_g t$ and say that s is *more general than* t if there exists a substitution η such that $t = s\eta$.

Proposition 72 *Let t, s be terms of the same sort and σ be a substitution mapping all variables in s to pairwise distinct constant symbols (of the appropriate sorts) not occurring in t or s. If $t \succeq_g s\sigma$, then $t \succeq_g s$.*

Proof For every term u, we denote by $\gamma(u)$ the term obtained from u by replacing any constant symbol of the form $x\sigma$ by the variable x. Note that the function γ is well-defined because by hypothesis σ is injective. Since $t \succeq_g s\sigma$, we must have $t\theta = s\sigma$, for some substitution θ. Let θ' be the substitution defined as follows: for every x, $x\theta' \stackrel{\text{def}}{=} \gamma(x\theta)$. Since t contains no constant symbol of the form $x\sigma$, it is clear that $\gamma(t\theta) = t\theta'$, hence $\gamma(s\sigma) = t\theta'$. By definition of γ, $\gamma(s\sigma) = s$, thus $t\theta' = s$ and $t \succeq_g s$.

Definition 73 For every set of inference rules $\mathfrak{R}$, we write $S \vdash_{\mathfrak{R}} \mathcal{C}$ if $\mathcal{C}$ is derivable from S by the rules in $\mathfrak{R}$ in any number of steps. A set of rules $\mathfrak{R}$ is *liftable* if the following property holds: for every set of c-clauses S and for every c-clause $\mathcal{C}$, if $\mathit{Inst}(S) \vdash_{\mathfrak{R}} \mathcal{C}$ then there exists a c-clause $\mathcal{C}'$ and a substitution σ such that $S \vdash_{\mathfrak{R}} \mathcal{C}'$ and $\mathcal{C}'\sigma = \mathcal{C}$.

The following proposition states a well-known property of the standard Resolution calculus:

Proposition 74 *The set consisting of the* ***Resolution****,* ***Factorization*** *and* ***Constraint Factorization*** *rules is liftable.*

Proof By an immediate induction on the derivation.

Remark 75 The superposition calculus is not liftable. For example, $p(a, b)$ is derivable from $\{b \simeq a,\ p(b, b)\}$ if $b > a$, but no c-clause more general than $p(a, b)$ is derivable from $\{b \simeq a,\ p(x, x)\}$.

Definition 76 Let α be a rooted formula containing a unique free variable x of sort $\mathtt{s} \in \mathtt{I}$. A *simple induction scheme* for α is a formula of the form:

$$\bigwedge_{i=1}^{n} (\alpha(t_i) \Rightarrow \exists y\, (y \prec t_i \wedge \alpha(y)))$$

where $\{t_1, \ldots, t_n\}$ is a covering set of terms of sort $\mathtt{s}$.

Definition 77 A *simple inductive invariant* for a set of c-clauses S is a rooted formula α such that there exists a simple induction scheme ψ for α and[17]:

$$\forall^* \Gamma \cup \forall^* S_\top \models_{fol} \psi.$$

Intuitively, if a set of c-clauses admits a simple inductive invariant, then, by the infinite descent principle, this set cannot be satisfiable. We assume in the remainder of the section that the core rules are liftable and refutationnally complete w.r.t. first-order satisfiability, i.e., that if S is a set of clauses such that $S \models_{fol} \square$, then $S \vdash_{core} \square$, and that they contain the `Constraint Factorization` rule (the `Abstraction` and `Instantiation` rules are not needed). The results apply, e.g., to the Resolution or Hyperresolution calculi, but not to the Superposition calculus. The following theorem states a form of completeness for $\mathcal{IC}$. Informally, it ensures that, if a simple inductive invariant exists for a set of clauses, then the empty clause is derivable.

Theorem 78 *Let S be a set of c-clauses. Let t, s be two terms of a sort in* `I` *and let p be a position in t. Consider a variable x that does not occur in t or s and assume that there exists a simple inductive invariant α for S such that every c-clause in $[\![\alpha(x) \mid s \simeq t[x]_p]\!]$ occurs in S (possibly up to a renaming of variables). Then there exists a c-clause $\mathcal{C}$ derivable from $S \cup \Gamma$ by $\mathcal{IC}$ such that $\mathcal{C}$ subsumes $[\![\square \mid s \simeq t[x]_p]\!]$.*

The proof of Theorem 78 is based on the following definition and results:

Definition 79 A *propagation-constraint* is a conjunction of atoms of the form $\mathbf{T}^{\prec}_{\beta}(\mathbf{t})$.

Proposition 80 *Consider a set of c-clauses S, two terms t, s of a sort in* `I` *and let p be a position in t. Assume that every c-clause in S is of the form $[\![C \mid \top]\!]$ or $[\![C \mid s \simeq t[x]_p]\!]$, where $\mathrm{var}(C) \cap (\mathrm{var}(s) \cup \mathrm{var}(t)) = \emptyset$ and $x \notin \mathrm{var}(t)$. If $[\![D \mid \mathcal{X}]\!]$ is a c-clause derivable from $S \cup \Gamma$ by the core rules then $\mathcal{X}$ is of the form*

$$(s \simeq t[x])\eta_1[u_1/x] \wedge \cdots \wedge (s \simeq t[x])\eta_n[u_n/x] \wedge \mathcal{X}',$$

where:

- *each η_i is a variable-disjoint renaming of the variables in s, t, and $x \notin \mathrm{dom}(\eta_i)$;*
- *$(\mathrm{var}(u_i) \cup \mathrm{var}(D) \cup \mathrm{var}(\mathcal{X}')) \cap (\mathrm{var}(s\eta_i) \cup \mathrm{var}(t\eta_i)) = \emptyset$;*
- *$\mathcal{X}'$ is a propagation-constraint.*

Proof First note that, although $x \notin \mathrm{var}(t)$ by hypothesis, it is possible to have $x \in \mathrm{var}(C)$. Any constraint in $\mathcal{X}$ comes either from the axioms in Γ or from the c-clauses in S. In the former case the constraint must be a propagation-constraint, and in the latter case, it must be of the form $(s \simeq t[x]_p)\eta\gamma$, where η is a renaming (applied to ensure that the hypotheses are variable-disjoint) and γ denotes the union of all unifiers in the derivation. Since t and s share no variable with the clausal part of the c-clauses, the renamed terms $t\eta$ and $s\eta$ cannot be affected by the substitution γ and cannot share variables with the other atoms occurring in the c-clause. This entails that $(s \simeq t[x]_p)\eta\gamma$ must be of the form $(s \simeq t[x]_p)\eta[u/x]$, for some term u. The detailed proof is by an immediate induction on the length of the derivation.

[17] See Section 3.2 for the definition of Γ and Definition 10 on Page 9 for the definition of $S_\top$.

Proposition 81 *Let u be a ground term and $\mathcal{X}$ be a propagation-constraint. Consider an extended formula α with a unique free variable x, and let t, s be terms such that $x \notin \mathrm{var}(s) \cup \mathrm{var}(t)$ and every clause in $[\![\alpha(x) \mid s \simeq t[x]_p]\!]$ occurs in S possibly up to a renaming of variables. If $[\![\Box \mid \mathcal{X}]\!]$ is derivable from $\Gamma \cup S_\top \cup cnf(\alpha(u))$ by the core rules then there exist a c-clause $\mathcal{C}$ and a substitution θ such that:*

- *$\mathcal{C}$ is of the form $[\![\Box \;\big|\; \mathcal{X}']\!]$ or $[\![\Box \;\big|\; s \simeq t[u']_p \wedge \mathcal{X}']\!]$, where $u'\theta = u$ and $\mathcal{X}'\theta = \mathcal{X}$;*
- *$\mathcal{C}$ is derivable from $S \cup \Gamma$ by the core rules.*

Proof Let s', t' be arbitrarily chosen ground instances of s and t respectively. Consider the derivation yielding $[\![\Box \mid \mathcal{X}]\!]$ from $\Gamma \cup S_\top \cup cnf(\alpha(u))$. By adding the constraint $s' \simeq t'[u]_p$ to every clause in $cnf(\alpha(u))$ and all their descendants we obtain a derivation of $[\![\Box \mid \mathcal{Y} \wedge \mathcal{X}]\!]$ from $\Gamma \cup S_\top \cup [\![\alpha(u) \;\big|\; s' \simeq t'[u]_p]\!]$, where $\mathcal{Y}$ is either $s' \simeq t'[u]_p$ (if a clause in $cnf(\alpha(u))$ is used in the above derivation) or $\top$ (if no clause in $cnf(\alpha(u))$ is used). By hypothesis, every c-clause in $[\![\alpha(u) \;\big|\; s' \simeq t'[u]_p]\!]$ is an instance of a c-clause in S because x is the only free variable in α. Since the core rules are liftable, we deduce that there exist a c-clause $\mathcal{C}$ and a substitution μ such that $\Gamma \cup S \vdash_{\text{CORE}} \mathcal{C}$ and $\mathcal{C}\mu = [\![\Box \mid \mathcal{Y} \wedge \mathcal{X}]\!]$. By Proposition 80, $\mathcal{C}$ must be of the form $[\![\Box \;\big|\; \bigwedge_{i=1}^m s_i \simeq t_i[u_i]_p \wedge \mathcal{X}'']\!]$, where $(\mathrm{var}(s_i) \cup \mathrm{var}(t_i)) \cap (\mathrm{var}(u_i) \cup \mathrm{var}(\mathcal{X}'')) = \emptyset$ for all $i = 1, \ldots, m$, and $\mathcal{X}''$ is a propagation-constraint. Since $\mathcal{C}\mu = [\![\Box \mid \mathcal{Y} \wedge \mathcal{X}]\!]$, we must have $t_i\mu = t'$, $s_i\mu = s'$, $u_i\mu = u$ for all $i = 1, \ldots, m$, and $\mathcal{X}''\mu = \mathcal{X}$. Let σ be the m.g.u. of $\{(t_i, s_i, u_i) \mid i = 1, \ldots, m\}$, so that $\mu = \sigma\theta$ for some substitution θ. By applying at most m times the `Constraint Factorization` rule, we derive a c-clause that is of the form $[\![\Box \;\big|\; s_1\sigma \simeq t_1\sigma[u_1\sigma]_p \wedge \mathcal{X}''\sigma]\!]$, up to a renaming. Since each t_i (resp. s_i) is a renaming of t (resp. s) and shares no variables with u_i or $\mathcal{X}'$, we can assume by renaming that $t_1\sigma = t$ and $s_1\sigma = s$. Then the result follows by taking $u' = u_1\sigma$ and $\mathcal{X}' = \mathcal{X}''\sigma$.

Proposition 82 *Consider a formula $\alpha(y)$ and a c-clause $\mathcal{C} = [\![C \mid \mathcal{X}]\!]$, let $S' = [\![\alpha(y) \mid s \simeq t[y]_p]\!]$ and let S be a set of standard clauses. If $\Gamma \cup S \cup S' \vdash_{\text{CORE}} \mathcal{C}$, then $\mathfrak{I}^\delta(\mathcal{C}, \mathcal{X}, p)$ is defined and is of the form $(\neg \bigwedge_{C \in E} C)$, for some $E \subseteq cnf(\alpha(z))$.*

Proof By definition of $[\![\alpha(y) \mid s \simeq t[y]_p]\!]$, all clauses in S' are of the form $[\![D \mid s \simeq t[y]_p]\!]$, where $D \in cnf(\alpha(y))$. We prove the result by induction. If $\mathcal{C}$ is a standard clause, then by Case 1 of Definition 36, $\mathfrak{I}^\delta(\mathcal{C}, s \simeq t[u']_p, p) = \bot$, and the result holds. If $\mathcal{C}$ occurs in $\mathcal{S}'$, then by Case 2 of Definition 36, $\mathfrak{I}^\delta(\mathcal{C}, s \simeq t[u']_p, p) = \neg C[z/y]$ and the result holds again, since $C \in cnf(\alpha(x))$ in this case. Otherwise, $\delta(\mathcal{C})$ must be defined, and by Case 3 of Definition 36, $\mathfrak{I}^\delta(\mathcal{C}, s \simeq t[u']_p, p)$ is of the form $\bigvee_{\mathcal{D}' \in \delta(\mathcal{C})} \mathfrak{I}^\delta(\mathcal{D}', \mathcal{X}_{\mathcal{D}'}, p)\sigma$. By the induction hypothesis, $\mathfrak{I}^\delta(\mathcal{D}', \mathcal{X}_{\mathcal{D}'}, p)\sigma = (\neg \bigwedge_{C \in E_{\mathcal{D}'}} C)[x_{\mathcal{D}'}]$ for some $E_{\mathcal{D}'} \subseteq cnf(\alpha(z))$. It is then straightforward to verify that the result also holds for ψ, with $E \stackrel{\text{def}}{=} \bigcup_{\mathcal{D}' \in \delta(\mathcal{C})} E_{\mathcal{D}'}$.

We are now in the position to give the proof of Theorem 78:

Proof We assume that $S \cup \Gamma \not\models_{fol} \Box$; otherwise the proof follows from the refutational completeness of the core rules. Let $S' = [\![\alpha(x) \mid s \simeq t[x]_p]\!]$, and u be a $\lhd$-minimal ground term of the same sort as $t|_p$ (such a term necessarily exists since the domain of a sort cannot be empty and $\lhd$ is well-founded[18]). By hypothesis, α admits a simple induction

[18] Note that u is not necessarily a constant ($\lhd$-minimal constants of the same sort as $t|_p$ do not always exist). We may, however, assume w.l.o.g. that the signature contains a constant of every sort, but this constant is not necessarily a constructor.

scheme ψ of the form

$$(\dagger) \quad \bigwedge_{i=1}^{n} (\alpha(t_i) \Rightarrow \exists x\, (x \prec t_i \wedge \alpha(x))),$$

for which the conditions of Definitions 76 and 77 are satisfied. In particular, u is an instance of t_i, for some $i = 1, \ldots, n$, and we have: $\forall^* \Gamma \cup \forall^* S_\top \models_{fol} \alpha(u) \Rightarrow \exists x\, (x \prec u \wedge \alpha(x))$. Since u is minimal w.r.t. $\lhd$, this entails that $\forall^* \Gamma \cup \forall^* S_\top \cup cnf(\alpha(u)) \models_{fol} \Box$, and because the core rules are refutationally complete, $\Gamma \cup S_\top \cup cnf(\alpha(u)) \vdash_{\text{CORE}} \Box$. By Proposition 81, we deduce that either $S \cup \Gamma \vdash_{\text{CORE}} \Box$ or $\Gamma \cup S_\top \cup S' \vdash_{\text{CORE}} [\![\Box \mid s \simeq t[u']_p]\!]$ where $u' \succeq_g u$. The former case cannot occur because $S \cup \Gamma \not\models_{fol} \Box$, thus $\Gamma \cup S_\top \cup S' \vdash_{\text{CORE}} [\![\Box \mid s \simeq t[u']_p]\!]$. Let $\mathcal{C} = [\![\Box \mid s \simeq t[u']_p]\!]$. By definition, all the hypotheses in the corresponding inference tree are either standard clauses in $S_\top$ or c-clauses occurring in S'. Therefore, by Proposition 82, $\mathfrak{I}^\delta(\mathcal{C}, s \simeq t[u']_p, p)$ is defined and is of the form $\neg \bigwedge_{C \in E} C$, where $E \subseteq cnf(\alpha(z))$. The `Trigger` rule applies and activates the extended formula $\beta \stackrel{\text{def}}{=} (\neg \bigwedge_{C \in E} C)[z]$.

Now, consider a $\subseteq$-maximal subset E' of $cnf(\alpha(z))$ such that an extended formula $\beta' \equiv (\neg \bigwedge_{C \in E'} C)[z]$ is activated. Since β and E satisfy these properties, E' necessarily exists. Let $j \in [1, n]$ and v be a ground term obtained from t_j by replacing all variables in t_j by fresh pairwise distinct non-constructor constant symbols of the appropriate sort. This entails that v is an instance of t_j, and by (†) we have: $\forall^* \Gamma \cup \forall^* S_\top \models_{fol} \alpha(v) \Rightarrow \exists x\, (x \prec v \wedge \alpha(x))$. Since $E' \subseteq cnf(\alpha(z))$, we have $\alpha(x) \models_{fol} \neg\beta'(x)$, and therefore,

$$\forall^* \Gamma \cup \forall^* S_\top \cup [\![\alpha(v) \mid \top]\!] \models_{fol} \exists x\, (x \prec v \wedge \alpha(x)) \models_{fol} \exists x\, (x \prec v \wedge \neg\beta'(x)),$$

so that $\forall^* \Gamma \cup \forall^* S_\top \cup [\![\alpha(v) \mid \top]\!] \cup [\![x \not\prec v \vee \beta'(x) \mid \top]\!] \models_{fol} \Box$. By Definition 18, the set of axioms $\Gamma_{\mathbf{T}}$ contains all the c-clauses in $cnf(\beta'(x) \vee \neg \mathbf{T}_{\beta'}(x))$, hence we deduce that

$$\forall^* \Gamma \cup \forall^* S_\top \cup [\![\alpha(v) \mid \top]\!] \cup \{[\![x \not\prec v \vee \mathbf{T}_{\beta'}(x) \mid \top]\!]\} \models_{fol} \Box.$$

By the completeness of the core rules, this entails that:

$$\Gamma \cup S_\top \cup [\![\alpha(v) \mid \top]\!] \cup \{[\![x \not\prec v \vee \mathbf{T}_{\beta'}(x) \mid \top]\!]\} \vdash_{core} \Box.$$

By attaching the constraint $\mathbf{T}^{\prec}_{\beta'}(v)$ to the c-clause $[\![x \not\prec v \vee \mathbf{T}_{\beta'(x)} \mid \top]\!]$ and all its descendants, we deduce that

$$\Gamma \cup S_\top \cup [\![\alpha(v) \mid \top]\!] \cup \{\left[\!\!\left[x \not\prec v \vee \mathbf{T}_{\beta'}(x) \;\middle|\; \mathbf{T}^{\prec}_{\beta'}(v)\right]\!\!\right]\} \vdash_{core} [\![\Box \mid \mathcal{Y}_j]\!],$$

where $\mathcal{Y}_j$ is either $\mathbf{T}^{\prec}_{\beta'}(v)$, if the c-clause $[\![x \not\prec v \vee \mathbf{T}_{\beta'(x)} \mid \top]\!]$ is used at least once in the derivation, or $\top$ otherwise. By definition, the set $\Gamma_{\mathbf{T}^\prec}$ contains the axiom: $\left[\!\!\left[y' \not\prec y \vee \mathbf{T}_{\beta'}(y') \;\middle|\; \mathbf{T}^{\prec}_{\beta'}(y)\right]\!\!\right]$, that is more general than $\left[\!\!\left[x \not\prec v \vee \mathbf{T}_{\beta'}(x) \;\middle|\; \mathbf{T}^{\prec}_{\beta'}(v)\right]\!\!\right]$. Since the core rules are liftable, it is possible to derive from $\Gamma \cup S_\top \cup [\![\alpha(v) \mid \top]\!]$ a c-clause $[\![\Box \mid \mathcal{X}]\!] \succeq_g [\![\Box \mid \mathcal{Y}_j]\!]$. By Proposition 81, there exist a c-clause $\mathcal{D}_j$ and a substitution θ_j such that $\Gamma \cup S_\top \cup S' \vdash_{\text{CORE}} \mathcal{D}_j$, where $\mathcal{D}_j$ is of the form $[\![\Box \mid \mathcal{Y}'_j]\!]$, $\mathcal{Y}'_j$ is either $\mathcal{X}'$ or $s \simeq t[v']_p \wedge \mathcal{X}'$, with $v'\theta = v$ and $\mathcal{X}'\theta = \mathcal{X}$. By hypothesis v is obtained from t_j by replacing each variable by new, pairwise distinct, constant symbols, thus by Proposition 72, we deduce that $v' \succeq_g t_j$, and $\mathcal{X}' \succeq_g \mathcal{Y}_j$. We deduce that for all $j \in [1, n]$, we can derive a c-clause D_j that is more general than either

$[\![\Box \mid s \simeq t[t_j]_p \wedge \mathcal{Y}_j]\!]$ or $[\![\Box \mid \mathcal{Y}_j]\!]$. Furthermore, the set $\{t_1, \ldots, t_n\}$ is covering, which entails that the `Domain Decomposition` rule can be applied on $\mathcal{D}_1, \ldots, \mathcal{D}_n$, yielding a c-clause $\mathcal{D}$ with either $\mathcal{D} = \left[\!\!\left[\Box \;\middle|\; s \simeq t[x]_p \vee \mathbf{T}^{\prec}_{\beta'}(x)\right]\!\!\right]$ or $\mathcal{D} = [\![\Box \mid s \simeq t[x]_p]\!]$. In the latter case, the proof is completed, hence we assume that the former condition holds. The hypotheses of this derivation are either clauses in $S_\top$ or c-clauses occurring in S'. Therefore, by Proposition 82, $\mathfrak{I}^\delta(\mathcal{D}, s \simeq t[x]_p, p)$ is defined, and is of the form $\neg \bigwedge_{C \in E''} C$, where $E'' \subseteq cnf(\alpha(z))$. If the `Induction` rule applies, then it yields $[\![\beta'(z) \mid \top]\!]$, from which $[\![\Box \mid s \simeq t[x]_p]\!]$ can be derived, and the proof is completed. Otherwise, we have $\neg \bigwedge_{C \in E''} C \not\models_s \beta'$ thus $E'' \not\subseteq E'$. By applying the rule `Iteration`, the extended formula $\alpha' = \beta' \vee \mathfrak{I}^\delta(\mathcal{D}, s \simeq t[x]_p, p)$ is activated. But we have:

$$\beta' \vee \mathfrak{I}^\delta(\mathcal{D}, s \simeq t[x]_p, p) \equiv \neg \bigwedge_{C \in E'} C \vee \neg \bigwedge_{C \in E''} C \equiv \neg \bigwedge_{C \in E' \cup E''} C$$

and $E' \subsetneq E' \cup E''$, which contradicts the maximality of E'.

7 Conclusion

A new method has been described to integrate inductive reasoning into saturation-based proof procedures such as Resolution or Superposition. This approach strongly increases the scope of these procedures by making them capable of refuting formulas containing function symbols interpreted over inductively defined domains. Some (necessarily restricted, since the logic is not semi-decidable) completeness results have been established.

Several lines of future work deserve to be considered. The first one consists in extending the completeness results given in Section 6, for instance by devising syntactic fragments for which completeness or decidability can be ensured (as it is done in [24] for clauses with indices interpreted as words), or by taking deletion and simplification rules into account. Another idea is to extend the procedure to handle more general inductive domains. The main restriction of the present work is that only free constructors are considered, and the first priority is to get rid of this condition. This affects both the definition of the semantics and the form of the `Domain Decomposition` rule – in particular, it is unclear how covering sets could be identified if the constructors are not free. It would also be fruitful to combine our approach with constrained superposition calculi (see, e.g., [4,6,21]) that allow one to combine first-order proofs with more specific theory reasoning. Another interesting follow-up would be to consider more general induction orders other than the subterm relation. Our technique should be applicable to any order, provided it can be axiomatized. Considering orders defined over tuples of variables could also extend the scope of our approach.

The implementation of the calculus will also be considered. For the sake of efficiency, we plan to implement the induction rules on the top of existing inference engines such as *E* [32] or Prover9 [26]. From a practical point of view, the systematic application of the induction rules generates a huge search space. It is thus essential to devise heuristics to identify the terms on which the induction must be applied and the candidate invariants that may be activated (see Remark 63).

References

1. V. Aravantinos, M. Echenim, and N. Peltier. A resolution calculus for first-order schemata. *Fundamenta Informaticae*, 125(2):101–133, 2013.
2. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 3(4):217–247, 1994.
3. L. Bachmair and H. Ganzinger. Resolution theorem proving. In Robinson and Voronkov [31], pages 19–99.
4. L. Bachmair, H. Ganzinger, and U. Waldmann. Refutational theorem proving for hierarchic first-order theories. *Applicable Algebra in Engineering, Communication and Computing*, 5(3):193–212, 1994.
5. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Proceedings of the 23^{rd} International Conference on Computer Aided Verification (CAV '11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, July 2011. Snowbird, Utah.
6. P. Baumgartner, J. Bax, and U. Waldmann. Finite quantification in hierarchic theorem proving. In S. Demri, D. Kapur, and C. Weidenbach, editors, *IJCAR*, volume 8562 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2014.
7. A. Bouhoula, E. Kounalis, and M. Rusinowitch. SPIKE, an automatic theorem prover. In *Proceedings of LPAR'92*, volume 624, pages 460–462. Springer-Verlag, 1992.
8. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14:14–189, 1995.
9. R. S. Boyer and J. S. Moore. A theorem prover for a computational logic. In M. E. Stickel, editor, *CADE*, volume 449 of *LNCS*, pages 1–15. Springer, 1990.
10. J. Brotherston. Cyclic Proofs for First-Order Logic with Inductive Definitions. In B. Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: Proceedings of TABLEAUX 2005*, volume 3702 of *LNAI*, pages 78–92. Springer-Verlag, 2005.
11. J. Brotherston, N. Gorogiannis, and R. L. Petersen. A generic cyclic theorem prover. In R. Jhala and A. Igarashi, editors, *Programming Languages and Systems - 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings*, volume 7705 of *Lecture Notes in Computer Science*, pages 350–367. Springer, 2012.
12. A. Bundy. The automation of proof by mathematical induction. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 845–911. Elsevier and MIT Press, 2001.
13. K. Claessen, M. Johansson, D. Rosén, and N. Smallbone. Automating inductive proofs using theory exploration. In M. P. Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 392–406. Springer, 2013.
14. H. Comon. Inductionless induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 14, pages 913–962. North-Holland, 2001.
15. H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7:371–475, 1989.
16. S. Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond*. Theses, École polytechnique, Sept. 2015.
17. S. Cruanes. Superposition with structural induction. In C. Dixon and M. Finger, editors, *Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings*, volume 10483 of *Lecture Notes in Computer Science*, pages 172–188. Springer, 2017.
18. S. Eberhard and S. Hetzl. Inductive theorem proving based on tree grammars. *Ann. Pure Appl. Logic*, 166(6):665–700, 2015.
19. S. Falke and D. Kapur. Rewriting induction + linear arithmetic = decision procedure. In B. Gramlich, D. Miller, and U. Sattler, editors, *Automated Reasoning*, volume 7364 of *LNCS*, pages 241–255. Springer Berlin Heidelberg, 2012.
20. M. Horbach and C. Weidenbach. Deciding the inductive validity of for all there exists * queries. In E. Grädel and R. Kahle, editors, *CSL*, volume 5771 of *LNCS*, pages 332–347. Springer, 2009.
21. M. Horbach and C. Weidenbach. Superposition for fixed domains. *ACM Trans. Comput. Logic*, 11(4):1–35, 2010.

22. M. Johansson, L. Dixon, and A. Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47(3):251–289, 2011.
23. A. Kersani and N. Peltier. Combining Superposition and Induction: A Practical Realization. In *Proceedings of FROCOS'13 (Frontiers of Combining Systems*, volume 8152 of *LNCS*, pages 7–22. Springer, 2013.
24. A. Kersani and N. Peltier. Completeness and Decidability Results for First-order Clauses with Indices. In *Proceedings of CADE'13 (24th International Conference on Automated Deduction)*, volume 7898 of *LNCS*, pages 58–75. Springer, 2013.
25. A. Leitsch. *The resolution calculus*. Springer. Texts in Theoretical Computer Science, 1997.
26. W. McCune. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.
27. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In Robinson and Voronkov [31], pages 371–443.
28. A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Form. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science, 2001.
29. D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.
30. A. Reynolds and V. Kuncak. *Verification, Model Checking, and Abstract Interpretation: 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, chapter Induction for SMT Solvers, pages 80–98. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
31. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
32. S. Schulz. System Description: E 1.8. In K. McMillan, A. Middeldorp, and A. Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.
33. S. Stratulat. A general framework to build contextual cover set induction provers. *J. Symb. Comput.*, 32(4):403–445, 2001.
34. A. Voronkov. The anatomy of vampire: Implementing bottom-up procedures with code trees. *Journal of Atomated Reasoning*, 15 (2):237–265, Jan. 1995.
35. A. Voronkov. *Computer Aided Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, chapter AVATAR: The Architecture for First-Order Theorem Provers, pages 696–710. Springer International Publishing, Cham, 2014.
36. C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Topic. System description: SPASS version 1.0.0. In *Proceedings of the 16th Conference on Automated Deduction (CADE-16)*, pages 378–382. Springer LNCS 1632, 2001.
37. C. Wirth. Descente infinie + deduction. *Logic Journal of the IGPL*, 12(1):1–96, 2004.