




# Rewriting and Inductive Reasoning

Márton Hajdu<sup></sup>, Laura Kovács<sup></sup>, and Michael Rawson<sup></sup>

TU Wien

## Abstract

Rewriting techniques based on reduction orderings generate “just enough” consequences to retain first-order completeness. This is ideal for superposition-based first-order theorem proving, but for at least one approach to inductive reasoning we show that we are missing crucial consequences. We therefore extend the superposition calculus with rewriting-based techniques to generate sufficient consequences for automating induction in saturation. When applying our work within the unit-equational fragment, our experiments with the theorem prover Vampire show significant improvements for inductive reasoning.

## 1 Introduction

Automating *proof by induction* is a particularly hard task with a long history [1, 6, 8, 24, 31]. A recent promising line of research in this direction comes with the integration of induction into saturation-based first-order theorem proving [28, 19], by extending the logical calculus with induction inference rules. Such induction rules are of the form

$$\frac{\neg L[t] \vee C}{\text{cnf}(\neg F \vee C)}$$

where  $\neg L[t] \vee C$  is a clause,  $\neg L[t]$  a ground literal,  $t$  a term of some inductive data type, and  $F \rightarrow \forall x.L[x]$  is an instance of some valid second-order induction *schema*, for example structural induction<sup>1</sup>. Note that the schema instance  $F \rightarrow \forall x.L[x]$  is applied to resolve  $\neg L[t]$ , but the schema instance is never added to the saturation search space, making sure that the conclusion of the inference is derived via inductive reasoning. As such, the application of the induction rule is *triggered* by the presence of the literal  $\neg L[t]$  in the search space. Hence, if there is no such literal, no induction is applied. Applying induction only when triggered means that only premises of schema instances directly related to a clause selected during proof search are added to the search space. This is a strong heuristic method for automating induction, particularly in the presence of full first-order logic and theories [20, 22, 21].

**Challenge.** Unfortunately, applying induction only when triggered leads to tension between two competing factors:

- (i) It may be that a valid goal is not provable without deriving a certain consequence  $C$ , that in turn triggers an induction rule with premise  $C$  with a specific schema.
- (ii) Efficient first-order calculi, such as superposition [3], go to great lengths to derive and retain only those consequences absolutely required for completeness. Therefore, induction rules that could be triggered by a (missing) consequence  $C$  might not be applied.

We show in our motivating example that this tension causes a lost proof in practice (Section 2), as superposition may avoid generating consequences that would be needed to be used in inductive proofs.

---

<sup>1</sup> $\text{cnf}(\neg F \vee C)$  denotes the clasified formula  $\neg F \vee C$

Signature			
$0 : \text{nat}$	$\text{cons} : \text{nat} \rightarrow \text{list} \rightarrow \text{list}$	$\llbracket \cdot \rrbracket : \text{list} \rightarrow \text{nat}$	
$s : \text{nat} \rightarrow \text{nat}$	$(+) : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$	$f : \text{list} \rightarrow \text{list}$	
$\text{nil} : \text{list}$	$(@) : \text{list} \rightarrow \text{list} \rightarrow \text{list}$	$g : \text{list} \rightarrow \text{list}$	
Axioms — Ax			
$0 \neq s(x)$	(nat.1)	$\text{nil} @ x \simeq x$	(append.1)
$s(x) \neq s(y) \vee x \simeq y$	(nat.2)	$\text{cons}(x, y) @ z \simeq \text{cons}(x, y @ z)$	(append.2)
$\text{nil} \neq \text{cons}(x, y)$	(list.1)	$\llbracket \text{nil} \rrbracket \simeq 0$	(length.1)
$\text{cons}(x, y) \neq \text{cons}(z, u) \vee x \simeq z$	(list.2)	$\llbracket \text{cons}(x, y) \rrbracket \simeq s(\llbracket y \rrbracket)$	(length.2)
$\text{cons}(x, y) \neq \text{cons}(z, u) \vee y \simeq u$	(list.3)	$g(x) @ f(y) \simeq f(x) @ g(y)$	(ax.1)
$0 + x \simeq x$	(plus.1)	$\llbracket x \rrbracket + \llbracket f(y) \rrbracket \simeq \llbracket f(x) \rrbracket + \llbracket y \rrbracket$	(ax.2)
$s(x) + y \simeq s(x + y)$	(plus.2)	$\llbracket f(g(x)) \rrbracket \simeq s(\llbracket x \rrbracket)$	(ax.3)
Negated conjecture — $\neg\text{Conj}$		Auxiliary lemma — Lem	
$\llbracket g(d) @ f(c) \rrbracket \neq s(\llbracket c \rrbracket) + \llbracket d \rrbracket$	(nConj)	$\forall x, y. \llbracket x @ y \rrbracket \simeq \llbracket y \rrbracket + \llbracket x \rrbracket$	(lemma)

Figure 1: Motivating example, conjecturing that the first-order formula  $\text{Conj}$  is implied by first-order axioms  $\text{Ax}$ .

**Our contributions.** For automating (triggered) induction in saturation, we thus need to do something counter-intuitive for those accustomed to first-order superposition reasoning. We propose deriving slightly *more* consequences than usual, in order to trigger induction rules with suitable schemas. Naturally, this must still be as few extra consequences as possible in order to retain a high level of first-order efficiency. Concretely, motivated by applications of program verification [16] in this paper we focus on the unit-equational fragment of first-order logic with induction and bring the following contributions.

- (1) We introduce a modified superposition calculus (Section 4) with slightly relaxed constraints, allowing us to derive consequences that cannot be derived by standard superposition.
- (2) We impose new restrictions on our calculus (Section 4) to further improve efficiency while retaining our newfound ability to generate consequences.
- (3) We improve redundancy elimination for saturation with induction, providing sufficient conditions for skipping redundant induction steps and rewrites (Section 5).
- (4) We implement our approach in the VAMPIRE first-order prover [25] and apply it to proving inductive properties (Section 6). Results show that our work improves the state of the art in automating proofs by induction.

## 2 Motivating Example

We motivate our work using Figure 1. The data types  $\text{nat}$  and  $\text{list}$ —corresponding to the term algebras of natural numbers and lists in first-order logic—are defined inductively using constructors as given by the first-order formulas (nat.1)–(list.3). Moreover, Figure 1 defines the recursive functions for the addition of natural numbers (+), list append (@) and list length ( $\llbracket \cdot \rrbracket$ ), encoded by (plus.1)–(length.2), and declares the uninterpreted functions  $f$  and  $g$ . We use

infix notation for the symbols  $+$ ,  $@$  and  $\llbracket \cdot \rrbracket$ . Axioms (ax.1)–(ax.3) define the behaviour of  $f$  and  $g$ . All properties in Figure 1 are implicitly universally-quantified.

Suppose we are trying to prove that the axioms (denoted  $\text{Ax}$ ) in Figure 1 imply the following first-order formula:

$$\forall x, y. \llbracket g(y) @ f(x) \rrbracket \simeq s(\llbracket x \rrbracket + \llbracket y \rrbracket) \quad (\text{Conj})$$

Proving (Conj) in classical first-order logic can be reduced to establishing the unsatisfiability of the negation of (Conj) together with the axioms of Figure 1. That is, we prove unsatisfiability of (nConj) together with the axioms (nat.1)–(ax.3); here, (nConj) is the negated and Skolemized form, of (Conj), using the Skolem (list) functions  $c, d$ . While the axioms imply formula (Conj) in the theory of lists and natural numbers,  $\text{Ax} \wedge (\text{nConj})$  is not first-order unsatisfiable. Showing unsatisfiability requires an additional first-order axiom over lists, in particular the *auxiliary lemma* Lem of Figure 1. With this lemma,  $\text{Ax} \wedge (\text{nConj}) \wedge \text{Lem}$  is unsatisfiable and validity of Conj follows.

Let us make two key observations. First, Lem is a stronger property than Conj with respect to Ax; from (nConj), the negation of an instance of Lem (denoted  $\neg\text{Lem}$  by slight abuse of notation) can be derived via rewriting with equal terms. Second, Lem is not a first-order consequence of Ax, but it is valid with respect to Ax in the background theory of lists and natural numbers, a fact that can be shown by induction. We make use of these two observations to synthesize and use Lem in the proof of Conj as follows:

- (i) We derive  $\neg\text{Lem}$  from  $\text{Ax} \wedge (\text{nConj})$  by rewriting. Soundness of rewriting ensures that the unsatisfiability of  $\text{Ax} \wedge \neg\text{Lem}$  implies the unsatisfiability of  $\text{Ax} \wedge (\text{nConj})$ .
- (ii) We refute  $\text{Ax} \wedge \neg\text{Lem}$  by instantiating a valid induction schema with Lem to obtain a valid first-order induction *axiom*, which in conjunction with  $\text{Ax} \wedge \neg\text{Lem}$  is unsatisfiable, implying the unsatisfiability of  $\text{Ax} \wedge (\text{nConj})$  and hence the claim of Figure 1.

To derive  $\neg\text{Lem}$  from  $\text{Ax} \wedge (\text{nConj})$ , we apply the following rewriting steps:

- rewrite (nConj) with (plus.2) resulting in  $\llbracket g(d) @ f(c) \rrbracket \not\simeq s(\llbracket c \rrbracket + \llbracket d \rrbracket)$  (1)
- rewrite (1) with (ax.3) resulting in  $\llbracket g(d) @ f(c) \rrbracket \not\simeq \llbracket f(g(c)) \rrbracket + \llbracket d \rrbracket$  (2)
- rewrite (2) with (ax.2) resulting in  $\llbracket g(d) @ f(c) \rrbracket \not\simeq \llbracket g(c) \rrbracket + \llbracket f(d) \rrbracket$  (3)
- rewrite (3) with (ax.1) resulting in  $\llbracket f(d) @ g(c) \rrbracket \not\simeq \llbracket g(c) \rrbracket + \llbracket f(d) \rrbracket$  (4)

Notice that clause (4) is the negation of Lem, instantiated with  $f(d)$  and  $g(c)$ . To refute  $\text{Ax} \wedge \neg\text{Lem}$ , we *conjecture* Lem to be proven by induction, by taking the negation of (4) and *by generalizing over* the term  $f(d)$ . Hence, we instantiate the following *second-order structural induction formula* over lists of natural numbers with Lem:

$$\forall F. ((F(\text{nil}) \wedge \forall x \in \text{nat}, y \in \text{list}. (F(y) \rightarrow F(\text{cons}(x, y)))) \rightarrow \forall z \in \text{list}. F(z)) \quad (5)$$

Showing the first-order unsatisfiability of this induction axiom in conjunction with  $\text{Ax} \wedge \neg\text{Lem}$  requires no further rewriting<sup>2</sup>.

Note that the above reasoning actually proves Lem *in addition* to Conj, but it comes with the following two main challenges for proving Figure 1:

(C1) use rewriting with equalities to derive  $\neg\text{Lem}$  from  $\text{Ax} \wedge (\text{nConj})$  (Section 4);

---

<sup>2</sup>See Appendix A for details

(C2) combine inductive reasoning with first-order reasoning to refute  $\text{Ax} \wedge \neg\text{Lem}$  (Section 5).

For tackling challenge (C1), we use *rewriting* inferences to rewrite equal terms and generate auxiliary lemmas. However, such proof steps cannot always be performed with the ubiquitous (ordered) superposition inferences. Let us use a Knuth-Bendix simplification ordering (KBO)  $\succ$  [26] parameterized by a constant weight function and the precedence  $\gg$ :

$$d \gg c \gg g \gg f \gg ([\cdot]) \gg (@) \gg (+) \gg \text{cons} \gg \text{nil} \gg s \gg 0$$

The ordering  $\succ$  cannot orient the equalities of (plus.2) and (ax.3) right-to-left so that clause (2) could be derived by rewriting. Addressing such obstacles, we introduce an extension of the superposition calculus (Section 4) to enable generating auxiliary lemmas during saturation. Our extension solves challenge (C1) and provides an efficient reasoning backend for challenge (C2).

### 3 Theoretical Background

We assume familiarity with many-sorted first-order logic with equality. Variables are denoted with  $x, y, z$ , terms with  $s, t, u, w, l, r$ , all possibly with indices. A term is *ground* if it contains no variables. We use the standard logical connectives  $\neg, \vee, \wedge, \rightarrow$  and  $\leftrightarrow$ , and quantifiers  $\forall$  and  $\exists$ . A *literal* is an atom or its negation. The literal  $\bar{L}$  denotes the complement of literal  $L$ . A disjunction of literals is a *clause*. We denote clauses by  $C, D$  and reserve the symbol  $\square$  for the *empty clause* that is logically equivalent to  $\perp$ . We refer to the *clausal normal form* of a formula  $F$  by  $\text{cnf}(F)$ . We assume that  $\text{cnf}$  preserves satisfiability, i.e.  $F$  is satisfiable iff  $\text{cnf}(F)$  is satisfiable. We use  $\simeq$  to denote equality and write  $\bowtie$  for either  $\simeq$  or  $\not\approx$ .

An *expression*  $E$  is a term, literal, clause or formula. We write  $E[s]_p$  to state that the expression  $E$  contains some distinguished occurrence of the term  $s$  at some position  $p$ . We might simply write  $E[s]$  if the position  $p$  is not relevant. Further,  $E[s \mapsto t]$  denotes that this occurrence of  $s$  is replaced with  $t$ ; when  $s$  is clear from the context, we simply write  $E[t]$ . We say that  $t$  is a *subterm* of  $s[t]$ , denoted by  $t \leq s[t]$ ; and a *strict subterm* if additionally  $t \neq s[t]$ , denoted by  $t < s[t]$ . A *substitution* is a mapping from variables to terms. We denote substitutions by  $\theta, \sigma, \rho, \mu, \eta$ . A substitution  $\theta$  is a *unifier* of two terms  $s$  and  $t$  if  $s\theta = t\theta$ , and is a *most general unifier* (denoted  $\text{mgu}(s, t)$ ) if for every unifier  $\eta$  of  $s$  and  $t$ , there exists a substitution  $\mu$  s.t.  $\eta = \theta\mu$ .

A *position* is a finite sequence of positive integers. The *root position* is the empty sequence, denoted by  $\epsilon$ . Let  $p$  and  $q$  be positions. The *concatenation* of  $p$  and  $q$  is denoted by  $pq$ . We say that  $p$  is *above*  $q$  if there exists a position  $r$  such that  $pr = q$ , denoted by  $p \leq q$ . We say that  $p$  and  $q$  are *parallel*, denoted by  $p \parallel q$ , if  $p \not\leq q$  and  $q \not\leq p$ . We say that  $p$  is *to the left* of  $q$ , denoted by  $p <_l q$ , if there are positive integers  $i$  and  $j$ , positions  $r, p'$  and  $q'$  such that  $p = rip'$ ,  $q = rjq'$  and  $i < j$ .

Let  $\rightarrow$  be a binary relation. The *inverse* of  $\rightarrow$  is denoted by  $\leftarrow$ . The *reflexive-transitive closure* of  $\rightarrow$  is denoted by  $\rightarrow^*$ . A binary relation  $\rightarrow$  over the set of terms is a *rewrite relation* if (i)  $l \rightarrow r \Rightarrow l\theta \rightarrow r\theta$  and (ii)  $l \rightarrow r \Rightarrow s[l] \rightarrow s[r]$  for any term  $l, r, s$  and substitution  $\theta$ . A *rewrite ordering* is a strict rewrite relation. A *reduction ordering* is a well-founded rewrite ordering. In this paper we consider reduction orderings total on ground terms. Such orderings satisfy  $s \triangleright t \Rightarrow s \succ t$  and are also called *simplification orderings*.

#### 3.1 Saturation-Based Theorem Proving

We briefly introduce saturation-based proof search in first-order theorem proving. For details, we refer to [25, 19]. The majority of first-order theorem provers work with clauses, rather than

$$\begin{array}{l}
\text{(Sup)} \quad \frac{s[u] \bowtie t \vee C \quad \underline{l \simeq r} \vee D}{(s[r] \bowtie t \vee C \vee D)\theta} \quad \text{where} \quad \begin{array}{l} (1) u \text{ is not a variable,} \\ (2) \theta = \text{mgu}(l, u), \\ (3) r\theta \not\bowtie l\theta \text{ and } t\theta \not\bowtie s\theta, \end{array} \\
\text{(EqRes)} \quad \frac{s \not\bowtie t \vee C}{C\theta} \quad \text{where} \quad \theta = \text{mgu}(s, t), \\
\text{(EqFac)} \quad \frac{s \simeq t \vee u \simeq w \vee C}{(s \simeq t \vee t \not\bowtie w \vee C)\theta} \quad \text{where} \quad \begin{array}{l} (1) \theta = \text{mgu}(s, u), \\ (2) t\theta \not\bowtie s\theta \text{ and } w\theta \not\bowtie t\theta. \end{array}
\end{array}$$

Figure 2: The superposition calculus **Sup** for first-order logic with equality.

arbitrary formulas. Let  $S = \mathcal{A} \cup \{-G\}$  be a set of clauses including assumptions  $\mathcal{A}$  and the clausified negation  $\neg G$  of a goal  $G$ . Given  $S$ , first-order provers *saturate*  $S$  by computing all logical consequences of  $S$  with respect to a sound inference system  $\mathcal{I}$ . This process is called *saturation*. An inference system  $\mathcal{I}$  is a set of inference rules of the form

$$\frac{C_1 \quad \dots \quad C_n}{C},$$

where  $C_1, \dots, C_n$  are the *premises* and  $C$  is the *conclusion* of the inference. We also write  $C_1, \dots, C_n \vdash_{\mathcal{I}} C$  to denote an inference in  $\mathcal{I}$ ; as  $\mathcal{I}$  is sound, this also means that  $C$  is a logical consequence of  $C_1, \dots, C_n$ . We denote that  $\mathcal{I}$  derives clause  $D$  from clauses  $\mathcal{C}$  with  $\mathcal{C} \vdash_{\mathcal{I}}^* D$ . If the saturated set of  $S$  contains the empty clause  $\square$ , the original set  $S$  of clauses is unsatisfiable, implying validity of  $\mathcal{A} \rightarrow G$ ; in this case, we established a *refutation* of  $\neg G$  from  $\mathcal{A}$ .

Completeness and efficiency of saturation-based reasoning relies on selecting and adding clauses from/to  $S$  using the inference system  $\mathcal{I}$ . To constrain the inference system, some first-order provers use simplification orderings on terms. Simplification orderings are extended to orderings over literals and clauses using the bag extension of the ordering; for simplicity, we write  $\succ$  both for the term ordering and its clause ordering extensions. Given an ordering  $\succ$ , a clause  $C$  is *redundant* with respect to a set  $S$  of clauses if there exists a subset  $S'$  of  $S$  such that  $S'$  implies  $C$  and is smaller than  $\{C\}$ , i.e.  $S' \rightarrow C$  and  $\{C\} \succ S'$ .

The *superposition calculus*, denoted **Sup** and given in Figure 2, is the most common inference system used by saturation-based first-order theorem provers [26]. We assume a literal selection function satisfying the standard condition on  $\succ$  and underline selected literals in **Sup** inferences. The **Sup** calculus is *sound* and *refutationally complete*: for any unsatisfiable formula  $\neg G$ , the empty clause  $\square$  can be derived as a logical consequence of  $\neg G$ .

### 3.2 Inductive Reasoning in Saturation

Inductive reasoning has recently been embedded in saturation-based theorem proving [12, 28], by extending **Sup** with a new inference rule. More precisely, we introduce a family of induction inference rules parameterized by a second-order formula  $G$  with exactly one free second-order variable  $F$ : the formula over which induction should be applied. Moreover, we restrict inductions to a set of terms  $\text{Ind}(\mathcal{T}) \subseteq \mathcal{T}$  where  $\mathcal{T}$  is the set of all terms. Then, the inference rules are of the following form:

$$\text{(Ind}_G\text{)} \quad \frac{\overline{L[t]} \vee C}{\text{cnf}(\neg G[L[x]] \vee C)} \quad \text{where} \quad \begin{array}{l} (1) L[t] \text{ is ground and } t \in \text{Ind}(\mathcal{T}), \\ (2) \forall F.(G[F] \rightarrow \forall x.F[x]) \text{ is a valid} \\ \text{second-order induction schema.} \end{array}$$

By an *induction axiom* we refer to an instance of a valid induction schema. When performing an  $\text{Ind}_G$  inference, the induction schema  $\forall F.(G[F] \rightarrow \forall x.F[x])$  is said to be *applied on* the clause  $\bar{L}[t] \vee C$ , or alternatively speaking  $\bar{L}[t] \vee C$  is *inducted upon*; in addition, we also say that we *induct on term*  $t$  in clause  $\bar{L}[t] \vee C$  with induction schema  $\forall F.(G[F] \rightarrow \forall x.F[x])$ . For example, using the schema (5), we parameterize the  $\text{Ind}_G$  schema with  $G := F[\text{nil}] \wedge \forall x, y.(F[y] \rightarrow F[\text{cons}(x, y)])$  and obtain the  $\text{Ind}_G$  instance:

$$(\text{Ind}_G) \frac{\bar{L}[t] \vee C}{\bar{L}[\text{nil}] \vee L[c_y] \vee C} \quad \text{where} \quad \begin{array}{l} (1) L[t] \text{ is ground,} \\ (2) t \in \text{Ind}(\mathcal{T}) \text{ is of sort list,} \\ (3) c_x \text{ and } c_y \text{ are fresh Skolem symbols.} \end{array}$$

$$\bar{L}[\text{nil}] \vee \bar{L}[\text{cons}(c_x, c_y)] \vee C$$

Note that the above  $\text{Ind}_G$  inference instance yields two clauses.

## 4 Efficient Rewriting in Saturation

As motivated in Section 2, rewriting derives clauses useful for auxiliary lemma generation that **Sup** is not able to derive. We therefore focus on rewriting variants captured by the following inference rule:

$$(\text{Rw}) \frac{C[l\theta] \quad l \simeq r}{C[r\theta]}$$

where  $\theta$  is a substitution. We call an  $\text{Rw}$  inference a *downward rewrite* if  $l\theta \succ r\theta$ , and call an  $\text{Rw}$  inference an *upward rewrite* if  $l\theta \prec r\theta$ .

We start by defining our base inference system, called the *Rewriting Calculus (ReC)*, as the calculus extending **Sup** with  $\text{Rw}$ . In other words, we define **ReC** to consist of the inference rules of  $\text{Sup} \cup \{\text{Rw}\}$ . The refutational completeness of **ReC** follows from the completeness of its subsystem **Sup**. In addition to completeness, we consider the following property over inference systems, and in particular over **ReC**.

**Definition 1** (Equational derivability (ED)). Let  $\theta$  be a substitution. An inference system  $\mathcal{I}$  *admits equational derivability (ED)* if, for any set of equations  $\mathcal{C}$ , equation  $l \simeq r$  and clause  $D[l\theta]$ , if  $\mathcal{C} \vdash_{\mathcal{I}}^* D[l\theta]$  then  $\mathcal{C}, l \simeq r \vdash_{\mathcal{I}}^* D[r\theta]$ .

Equational derivability in Definition 1 essentially expresses that an inference system can simulate the application of the  $\text{Rw}$  rule, by some (possibly longer) derivation. This allows us to introduce and compare variants of **ReC**, by imposing additional rewriting constraints in  $\text{Rw}$ . We state the following, straightforward result.

**Theorem 1 (ReC-ED).** *The inference system ReC admits ED.*

In the sequel, we develop three improved variants of **ReC** that admit ED, and thus derive the same consequences with equations as **ReC** does.

### 4.1 Peak Elimination in ReC

Let  $\mathcal{C}$  be a satisfiable set of clauses. Suppose there is some ground clause  $D$  that triggers the generation of a necessary inductive axiom, and suppose  $D$  can be derived from  $\mathcal{C}$  via rewrites with equations in  $\mathcal{C}$ . Hence our goal is to derive  $D$ . In Figure 3, we show<sup>3</sup> possible

<sup>3</sup>similarly to [29].

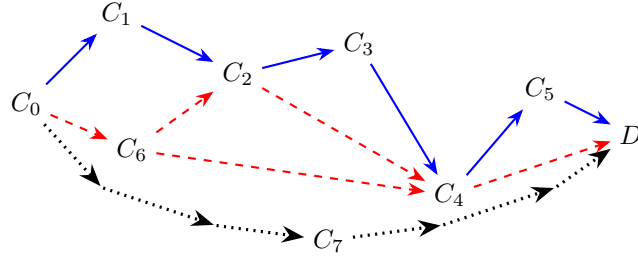


Figure 3: Possible rewrite sequences from a ground clause  $C_0$  to a ground clause  $D$ . The total order between clauses is  $C_1 \succ C_3 \succ C_2 \succ C_5 \succ C_0 \succ D \succ C_6 \succ C_4 \succ C_7$ , as also visualized by the vertical alignment of clauses.

ways to derive  $D$  from a ground clause  $C_0 \in \mathcal{C}$  using equations in  $\mathcal{C}$ . Arrows of Figure 3 point in the direction of deduction. Assume that all clauses in Figure 3 are ground; using a total simplification ordering  $\succ$  over ground clauses, we order clauses in Figure 3 as given by their vertical alignment in Figure 3. Therefore, an arrow going vertically upwards (resp. downwards) in Figure 3 corresponds to an upward (resp. downward) rewrite variant of  $\text{Rw}$ . We use three different arrows in Figure 3, corresponding to paths available at different saturation steps (iterations) while saturating  $\mathcal{C}$ :

- (1) Arrows  $\longrightarrow$  designate a path which is possible in a certain iteration  $i$  during saturation, that is with equations available at iteration  $i$ .
- (2) Arrows  $\dashrightarrow$  correspond to paths in later iterations than  $i$  but not necessarily at the end of the saturation process.
- (3) Arrows  $\cdots\rightarrow$  correspond to the “ideal path” at the end of the saturation process, that is, when the equations are transformed into a set of equations corresponding to a complete (non-overlapping and terminating) rewrite system.

As shown by the many rewriting steps of Figure 3, choosing a path between  $C_0$  and  $D$  is not trivial. For example, using arrows  $\longrightarrow$ , we may derive  $D$  from  $C_0$  in iteration  $i$  already, but in principle we have to exhaustively apply rewrites in all “directions”, resulting in many duplicate clauses. A different strategy is to wait until saturation end, in which case using arrows  $\cdots\rightarrow$  we rewrite  $C_0$  into its normal form  $C_7$ , and then from  $C_7$  we reach  $D$  only by upward rewrites. However, saturation may never terminate, for example in the presence of associativity and commutativity (AC) axioms.

Another option is to find a path of a specific form during saturation, such as the paths designated by  $\dashrightarrow$  arrows in Figure 3. We propose to avoid so-called *peaks* during saturation, where a peak comes with an upward rewrite followed by a downward rewrite. That is, upward rewrites followed by downward rewrites should be avoided. Depending on the positions in which the upward and downward rewrites happen, the following two possibilities occur:

- (i) If the positions of upward and downward rewrites are *parallel*, the two rewrites can be simply flipped. For example, the path  $C_0 \longrightarrow C_1 \longrightarrow C_2$  of Figure 3 is replaced by the path  $C_0 \dashrightarrow C_6 \dashrightarrow C_2$ .
- (ii) If the positions are *overlapping*, there is a superposition between the two rewriting equations of the peak. This superposition inference generates an equation that “cuts” the



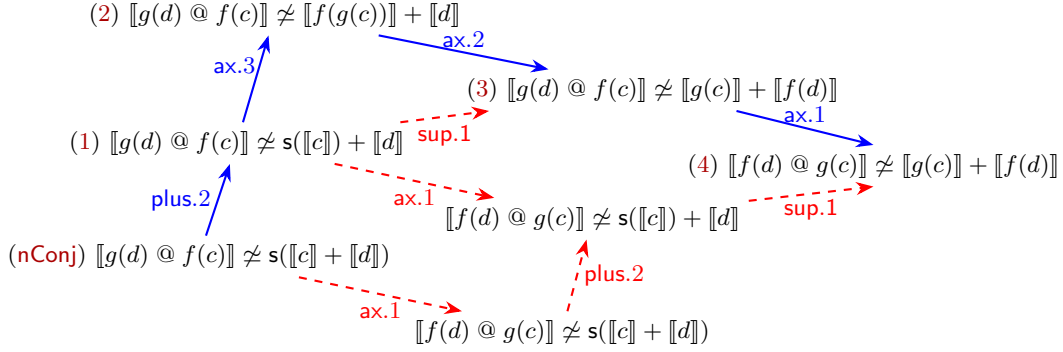


Figure 4: Possible rewrite sequences to derive clause (4) from (nConj) in the example of Section 2.

peak, giving a one-step rewrite alternative instead of two rewrites. For example, the peak  $C_4 \rightarrow C_5 \rightarrow D$  can be replaced by  $C_4 \dashrightarrow D$ . Note that sometimes multiple superpositions have to be performed before the path can be continued, e.g. the (double) peak  $C_6 \dashrightarrow C_2 \rightarrow C_3 \rightarrow C_4$  needs two superpositions to be eliminated, and performed simply as  $C_6 \dashrightarrow C_4$ .

To avoid peaks in saturation, we distinguish clauses resulting from upward rewrites (annotated as  $\blacksquare C$ ) from other clauses (annotated as  $\square C$ ). We use the notation  $\blacklozenge C$  to denote either of these. We might leave clauses without annotation if this information is not relevant in the context. We split the  $\text{Rw}$  inference into two components, resulting in the following inferences:

$$(\text{Rw}_\downarrow) \frac{\square C[l\theta] \quad \square l \simeq r}{\square C[r\theta]} \quad \text{where } l\theta \not\leq r\theta, \quad (\text{Rw}_\uparrow) \frac{\blacklozenge C[l\theta] \quad \square l \simeq r}{\blacksquare C[r\theta]} \quad \text{where } l\theta \not\leq r\theta.$$

We denote our **ReC** variant for avoiding peaks in saturation by **ReC<sub>v</sub>**, and define **ReC<sub>v</sub>** to consist of the inference rules of **Sup**  $\cup$   $\{\text{Rw}_\downarrow, \text{Rw}_\uparrow\}$ .

**Remark 1.** Note that the  $\text{Rw}_\downarrow$  and  $\text{Rw}_\uparrow$  rules both allow rewriting with incomparable equations. The reason for this is that disallowing rewrites with incomparable equations after upward rewrites violates *ED* in some cases<sup>4</sup>.

**Example 1.** Solid blue lines ( $\rightarrow$ ) in Figure 4.1 show the sequence of rewrite steps to reach clause (4) from clause (nConj) within the motivating example of Section 2, when using **ReC**.

Alternatively, we can perform the following  $\dashrightarrow$  steps with **ReC<sub>v</sub>**. A superposition into clause (ax.2) with (ax.3) results in

$$[[g(x)] + [f(y)]] \simeq s([x]) + [y]. \quad (\text{sup.1})$$

Using clause (sup.1), we eliminate the peak through clause (2), and directly derive clause (3) from clause (1) in **ReC<sub>v</sub>**. Note that we can switch the order of rewrites using clauses (ax.1) and (plus.2); and similarly the order of rewrites using clauses (ax.1) and (sup.1). We thus obtain the **ReC<sub>v</sub>** derivation of clause (4) via rewriting clause (nConj) with (ax.1), then with (plus.2), and finally with (sup.1).  $\square$

<sup>4</sup>see Appendix B for details



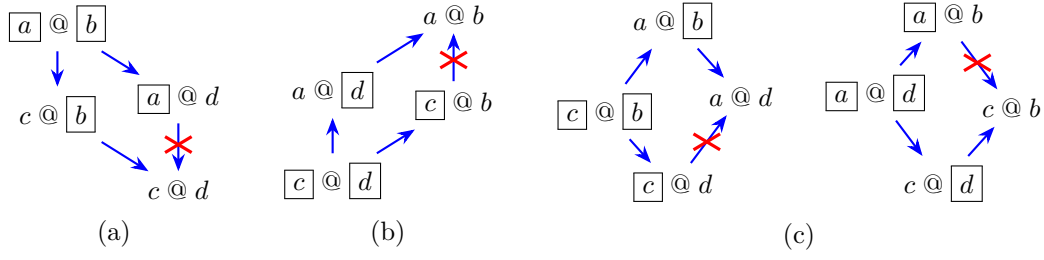


Figure 5: Possible parallel rewrites in **ReC** given equations  $a \simeq c$ ,  $b \simeq d$  with  $a \succ c$  and  $b \succ d$ . Rewrites corresponding to crossed out arrows are not performed in the left-to-right order.

We conclude **ReC<sub>v</sub>** with the following result.

**Theorem 2 (ReC<sub>v</sub>–ED).** *The inference system **ReC<sub>v</sub>** admits ED.*

## 4.2 Diamond Elimination in ReC

Note that rewrites in parallel positions can be performed in any order. If the rewriting is performed in all possible orders, this leads to a large number of duplicated clauses. In this section, we restrict **ReC** and **ReC<sub>v</sub>** to eliminate this effect, while preserving equational derivability from Definition 1.

Figure 5 illustrates rewriting possibilities for two parallel rewrites with **ReC**. Depending on the direction of the rewrites, there are three possibilities, that is, three “diamonds”: Figure 5(a) shows two downward rewrites; Figure 5(b) illustrates two upward rewrites; while Figure 5(c) shows one downward and one upward rewrite. Note that Figure 5(c) contains two subcases, one where the upward rewrite happens in the left position and one where it happens in the right position. We denote the positions to be rewritten with boxes, e.g.  $\boxed{a}$ . To generate all terms in these diamonds without duplicating any terms, we follow the tradition of reduction strategies in programming languages, for example *leftmost-outermost* (also *call-by-need*) and *leftmost-innermost* (also *call-by-value*) strategies [2]. We choose a *left-to-right* rewriting order, that is, we cannot perform a rewrite to the left of the previous rewrite. In Figure 5, the skipped rewrites are crossed out in red.

Figure 5(a) and Figure 5(b) are the same in **ReC<sub>v</sub>**. However, **ReC<sub>v</sub>** avoids duplication of Figure 5(c) in the first place (recall the parallel positions in Figure 3). Therefore, in the case of **ReC<sub>v</sub>**, we only apply the left-to-right order in the case of multiple consecutive  $\text{Rw}_\downarrow$  (resp.  $\text{Rw}_\uparrow$ ) inferences. Towards this, we associate with each clause  $C$  a position  $p$  where the previous rewrite was performed. We denote such clauses by  ${}_pC$ . Our modified  $\text{Rw}$  rule for avoiding duplicated diamonds is:

$$(\text{Rw}^\rightarrow) \frac{{}_pC[l\theta]_q \quad l \simeq r}{{}_qC[r\theta]_q} \quad \text{where } q \not\prec_l p.$$

Our **ReC** variant for *avoiding duplicated diamonds during rewritings* is denoted by **ReC<sup>→</sup>** and is defined to be  $\text{Sup} \cup \{\text{Rw}^\rightarrow\}$ . We state the following result.

**Theorem 3 (ReC<sup>→</sup>–ED).** *The inference system **ReC<sup>→</sup>** admits ED.*

Finally, we define a **ReC** variant that *combines peak-elimination with left-to-right rewriting orders*. We denote this **ReC** variant by  $\mathbf{ReC}_{\nabla}^{\rightarrow}$ . Here, we enforce the left-to-right order separately between downward and upward rewrites, as captured via the following **Rw** variants:

$$\begin{array}{l}
 (\mathbf{Rw}_{\downarrow}^{\rightarrow}) \frac{\frac{\Box_p C[l\theta]_q \quad \Box l \simeq r}{\Box_q C[r\theta]_q}}{\Box_q C[r\theta]_q} \quad \text{where} \quad \begin{array}{l} (1) \ l\theta \not\prec r\theta, \\ (2) \ q \not\prec_l p, \end{array} \\
 (\mathbf{Rw}_{\uparrow}^{\rightarrow}) \frac{\frac{\blacksquare_p C[l\theta]_q \quad \Box l \simeq r}{\blacksquare_q C[r\theta]_q}}{\blacksquare_q C[r\theta]_q} \quad \text{where} \quad \begin{array}{l} (1) \ l\theta \not\prec r\theta, \\ (2) \ \blacksquare_p C[l\theta] = \Box_p C[l\theta] \text{ or } q \not\prec_l p. \end{array}
 \end{array}$$

Our inference system  $\mathbf{ReC}_{\nabla}^{\rightarrow}$  is defined as  $\mathbf{Sup} \cup \{\mathbf{Rw}_{\downarrow}^{\rightarrow}, \mathbf{Rw}_{\uparrow}^{\rightarrow}\}$  and has the following property.

**Theorem 4** ( $\mathbf{ReC}_{\nabla}^{\rightarrow}$ -ED). *The inference system  $\mathbf{ReC}_{\nabla}^{\rightarrow}$  admits ED.*

## 5 Redundancy and Induction

As mentioned in Section 1, the **Sup** calculus tries to derive and retain as few clauses as possible without losing (refutational) completeness. Within the **ReC** calculus, as well as within its three refinements  $\mathbf{ReC}_{\nabla}$ ,  $\mathbf{ReC}^{\rightarrow}$  and  $\mathbf{ReC}_{\nabla}^{\rightarrow}$ , however, we not only derive more consequences, but we prevent their simplifications, resulting in less efficient reasoning than via **Sup**. The situation gets even worse when the prolific  $\mathbf{Ind}_G$  rule is used to enable inductive reasoning.

As a remedy, this section integrates redundancy elimination within our **ReC** calculi extended with  $\mathbf{Ind}_G$  inferences. Our main goal is to be as efficient as possible without losing inductive proofs. This includes, for example, preserving the ED property for our calculi extended with  $\mathbf{Ind}_G$ . We introduce sufficient criteria to skip induction inferences in **ReC**, and weaken the ED restriction to avoid deriving useless clauses for first-order and inductive reasoning.

We identify induction inferences that can be omitted without losing proofs and provide efficient ways to detect such (redundant) inferences.

**Remark 2.** *Note that constructor-based induction schemas give rise to a few optimisations:*

- (1) *Inducting on  $t$  in  $L[t] \vee C$  where  $t$  has zero occurrences is possible, but using constructor-based induction schemas only results in tautological clauses  $L \vee \neg L \vee C$  and clauses with duplicate literals  $L \vee L \vee C$ . We thus omit inducting on  $t$  in  $L[t] \vee C$  where  $t$  has zero occurrences.*
- (2) *Inducting on base constructors, such as  $0$  and  $\mathbf{nil}$ , only give weaker forms of the same clauses. For example, inducting upon  $\mathbf{nil}$  in  $L[\mathbf{nil}] \vee C$  yields clauses of the form  $L[\mathbf{nil}] \vee C' \vee C$ . We thus omit induction on base constructors.*

While the inductive inferences described in Remark 2 can easily be detected, this is not the case with more complex but useless inductive inferences, as shown in the next example.

**Example 2.** *Consider the  $\mathbf{Ind}_G$  inferences on term  $c$  in clauses (**nConj**) and (2), respectively. The first  $\mathbf{Ind}_G$  inference yields clauses*

$$\begin{array}{l}
 \llbracket g(d) @ f(\mathbf{nil}) \rrbracket \not\prec s(\llbracket \mathbf{nil} \rrbracket + \llbracket d \rrbracket) \vee \llbracket g(d) @ f(c_2) \rrbracket \simeq s(\llbracket c_2 \rrbracket + \llbracket d \rrbracket) \\
 \llbracket g(d) @ f(\mathbf{nil}) \rrbracket \not\prec s(\llbracket \mathbf{nil} \rrbracket + \llbracket d \rrbracket) \vee \llbracket g(d) @ f(\mathbf{cons}(c_1, c_2)) \rrbracket \not\prec s(\llbracket \mathbf{cons}(c_1, c_2) \rrbracket + \llbracket d \rrbracket)
 \end{array}$$

where  $c_1$  and  $c_2$  are fresh Skolem constants. The second  $\text{Ind}_G$  inference yields clauses

$$\begin{aligned} \llbracket g(d) @ f(\text{nil}) \rrbracket \not\approx \llbracket f(g(\text{nil})) \rrbracket + \llbracket d \rrbracket \vee \llbracket g(d) @ f(c_4) \rrbracket &\simeq \llbracket f(g(c_4)) \rrbracket + \llbracket d \rrbracket \\ \llbracket g(d) @ f(\text{nil}) \rrbracket \not\approx \llbracket f(g(\text{nil})) \rrbracket + \llbracket d \rrbracket \vee \llbracket g(d) @ f(\text{cons}(c_3, c_4)) \rrbracket &\not\approx \llbracket f(g(\text{cons}(c_3, c_4))) \rrbracket + \llbracket d \rrbracket \end{aligned}$$

where  $c_3$  and  $c_4$  are fresh Skolem constants. The two clause sets are equisatisfiable w.r.t. axioms (nat.1)–(ax.3) and yield the same consequences, hence it is sufficient to retain only one. This is unfortunately hard to detect due to the different sets of Skolem constants  $c_1, c_2$  and  $c_3, c_4$ . For example, simplifying the induction formulas and checking them for equivalence before clausification takes considerable effort.  $\square$

For detecting redundancies similar to Example 2, we characterize redundant induction inferences of interest and introduce sufficient conditions to efficiently detect them.

**Definition 2** (Redundant  $\text{Ind}_G$  inference). Let  $C$  be a clause and  $F$  a formula. The  $\text{Ind}_G$  inference  $C \vdash \text{cnf}(F)$  is *redundant* w.r.t. a set of equations  $E$  and a clause  $C'$ , if  $C \succ C'$  and there is a formula  $F'$  and an  $\text{Ind}_G$  inference  $C' \vdash \text{cnf}(F')$  s.t.  $F$  and  $F'$  are equivalent modulo rewriting with  $E$ .

It is easy to see that only non-redundant  $\text{Ind}_G$  inferences need to be performed to retain equational consequences and first-order refutations. The following two lemmas show sufficient conditions to efficiently check for redundant induction inferences.

**Lemma 1** (Redundant  $\text{Ind}_G$  – Condition I). Let  $l \simeq r$  and  $\bar{L}[t] \vee C$  be clauses,  $x$  a fresh variable, and  $\mathcal{I}$  an inference system that admits ED. If there is a substitution  $\theta$  s.t.  $l\theta \triangleleft L[x]$  and  $l\sigma \succ r\sigma$  where  $\sigma = \theta \cdot \{x \mapsto t\}$ , then the  $\text{Ind}_G$  inference

$$\bar{L}[t] \vee C \vdash \text{cnf}(-G[L[x]] \vee C)$$

is redundant in  $\mathcal{I} \cup \{\text{Ind}_G\}$  w.r.t. the clauses  $l \simeq r$  and  $(\bar{L}[t])[l\sigma \mapsto r\sigma] \vee C$ .

**Lemma 2** (Redundant  $\text{Ind}_G$  – Condition II). Let  $l \simeq r$  and  $\bar{L}[t] \vee C$  be clauses,  $x$  a fresh variable, and  $\mathcal{I}$  an inference system that admits ED. If there is a substitution  $\theta$  s.t.  $l\theta \leq t$  and  $l\theta \succ r\theta$ , then the  $\text{Ind}_G$  inference

$$\bar{L}[t] \vee C \vdash \text{cnf}(-G[L[x]] \vee C)$$

is redundant in  $\mathcal{I} \cup \{\text{Ind}_G\}$  w.r.t. the clauses  $l \simeq r$  and  $\bar{L}[t[l\theta \mapsto r\theta]] \vee C$ .

Lemmas 1–2 allow us to check for redundant  $\text{Ind}_G$  inferences similarly as performing demodulation, i.e. simplification by downward rewrites [17]. A consequence of these lemmas is that any non-redundant  $\text{Ind}_G$  inference on a clause that could be simplified by demodulation must induct on a subterm of a demodulatable term (i.e. a term that could be downward rewritten into a smaller term). The converse, however, that every subterm of a demodulatable term gives rise to a non-redundant  $\text{Ind}_G$  inference, does not hold in general.

**Example 3.** As shown in Example 2, the  $\text{Ind}_G$  inference on clause (2) using induction term  $c$  is redundant w.r.t. clause (ax.3) and (nConj) due to  $\llbracket f(g(c)) \rrbracket \succ s(\llbracket c \rrbracket)$ . The only non-redundant  $\text{Ind}_G$  inferences on clause (2) w.r.t. clause (ax.3) thus induct on the subterms  $f(g(c))$  or  $g(c)$  of  $\llbracket f(g(c)) \rrbracket$ .  $\square$

To control over which clauses induction should be triggered, we introduce the following notion for clauses that are not directly usable as premises for induction inferences.

**Definition 3** (Inductively redundant clause). A clause is *inductively redundant* if it is (first-order) redundant and all  $\text{Ind}_G$  inferences on it are redundant.

The following example shows an inductively redundant clause.

**Example 4.** Continuing Example 3, the term  $\llbracket g(d) @ f(c) \rrbracket$  in clause (2) can be demodulated, namely into  $\llbracket f(d) @ g(c) \rrbracket$  by clause (ax.1). This makes inducting only on the subterms  $g(d)$ ,  $f(c)$  or  $g(d) @ f(c)$  non-redundant. As  $\llbracket f(g(c)) \rrbracket$  and  $\llbracket g(d) @ f(c) \rrbracket$  render all  $\text{Ind}_G$  inferences on the subterms of each other redundant, clause (2) is inductively redundant.  $\square$

An inductively redundant clause is only necessary to preserve the ED property. Next, we show how some inductively redundant clauses can be avoided without losing ED. Towards this, we define so-called *ineffective* equations.

**Definition 4** (Ineffective equation). An equation  $l \simeq r$  is *ineffective* if  $l \succ r$ , each variable in  $l$  has at most one occurrence and there is no strict non-variable subterm  $s$  in  $l$  s.t.  $s\theta \in \text{Ind}(\mathcal{T})$  for some substitution  $\theta$ . An equation is called *effective* if it is not ineffective. We call an upward rewrite with an ineffective equation an *ineffective rewrite*.

The following example shows ineffective equations.

**Example 5.** As discussed in Example 2, base constructors such as  $0$  and  $\text{nil}$  are not inducted upon when using only constructor-based induction schemas; hence,  $0, \text{nil} \notin \text{Ind}(\mathcal{T})$ . The equations (plus.1), (append.1) and (length.1) are therefore ineffective, since they are oriented left-to-right, their left-hand sides are linear, and none of the strict non-variable subterms in their left-hand sides are inducted upon.  $\square$

The following lemma proves that the result of an ineffective rewrite is inductively redundant.

**Lemma 3** (Redundancy of ineffective rewrites). Let  $l \simeq r$  be an ineffective equation and  $C[l\theta]$  a ground clause. If  $C[l\theta] \succ (l \simeq r)\theta$ , the clause  $C[l\theta]$  is inductively redundant in any inference system that admits ED.

Consider a derivation of Rw inferences from an inductively non-redundant clause  $C$  into an inductively non-redundant conclusion  $D$ , where every intermediate clause in the derivation is inductively redundant. We may notice in such derivations that an ineffective rewrite is eventually followed by a rewrite that is not ineffective in an overlapping position. These rewrites can be performed together to avoid the intermediate inductively redundant clauses. To control and trigger such rewriting chains, we introduce the following *chaining* inferences.

$$\begin{array}{l}
\text{(CRw)} \quad \frac{C[l\theta] \quad l \simeq r}{C[r\theta]} \quad \text{where} \quad l\theta \not\prec r\theta \text{ or } l \simeq r \text{ is effective,} \\
\text{(Chain}_1\text{)} \quad \frac{s[l'] \simeq t \quad l \simeq r}{(s[r] \simeq t)\theta} \quad \text{where} \quad \begin{array}{l} (1) \theta = \text{mgu}(l, l'), \\ (2) s[l'] \simeq t \text{ is ineffective,} \\ (3) l\theta \not\prec r\theta \text{ and } l \simeq r \text{ is effective,} \end{array} \\
\text{(Chain}_2\text{)} \quad \frac{s \simeq t[l'] \quad l \simeq r}{(s \simeq t[r])\theta} \quad \text{where} \quad \begin{array}{l} (1) \theta = \text{mgu}(l, l'), \\ (2) l \simeq r \text{ is ineffective,} \\ (3) t[l']\theta \not\prec s\theta \text{ and } s \simeq t[l'] \text{ is effective.} \end{array}
\end{array}$$

The chaining inferences  $\text{Chain}_1$  and  $\text{Chain}_2$  combine ineffective and effective equations together in new, effective equations. Further,  $\text{CRw}$  disallows ineffective rewrites for consequence generation. By using chaining inferences for efficient rewrites in saturation with induction, we define the calculus  $\text{CReC}$  as  $\text{Sup} \cup \{\text{CRw}, \text{Chain}_1, \text{Chain}_2\}$ . While  $\text{CReC}$  does not admit ED, we note that if an inductively non-redundant clause  $D$  is derivable via  $\text{Rw}$  in  $\text{ReC}$ , then  $D$  is also derivable using chaining inferences in  $\text{CReC}$ . That is, inductive consequences are not lost in  $\text{CReC}$ . The following theorem adjusts such a variant of ED to  $\text{CReC}$ .

**Theorem 5 (CReC derivability).** *Let  $D$  be an inductively non-redundant clause. If  $\mathcal{C} \vdash_{\{\text{Rw}\}}^* D$ , then  $\mathcal{C} \vdash_{\{\text{CRw}, \text{Chain}_1, \text{Chain}_2\}}^* D$ .*

The calculi  $\text{CReC}_\vee$ ,  $\text{CReC}^\rightarrow$  and  $\text{CReC}_\vee^\rightarrow$  are respectively the variants of  $\text{ReC}_\vee$ ,  $\text{ReC}^\rightarrow$  and  $\text{ReC}_\vee^\rightarrow$ , when using  $\text{Chain}_1$  and  $\text{Chain}_2$ , and restricting the  $\text{Rw}$  variants in these calculi to be used with effective equations similarly as in  $\text{CRw}$ .<sup>5</sup> Derivability results for these calculi similar to Theorem 5 are straightforward based on Theorems 2–5.

## 6 Evaluation

**Implementation.** We implement our calculi in the VAMPIRE<sup>6</sup> prover. Our framework for equational consequence generation is controlled via the new option `-grw` which has the following values: `off` disables equational consequence generation and uses only the  $\text{Sup}$  calculus; `all` uses the calculus  $\text{ReC}$ ; `up` uses  $\text{ReC}_\vee$ ; `ltr` uses  $\text{ReC}^\rightarrow$ ; and `up_ltr` uses  $\text{ReC}_\vee^\rightarrow$ . With the further option `-mgrwd`, we limit the maximum depth of rewrites for  $\text{Rw}$  inference variants. The option `-grwc` toggles the chaining inferences, that is, the use of  $\text{CReC}$  variants. Finally, the option `-indrc` controls the redundancy check for induction, by using Lemmas 1–2 to avoid performing redundant  $\text{Ind}_G$  inferences.

Additionally, we use the following heuristics to control consequence generation in saturation with induction. We apply rewriting in a goal-oriented manner, only allowing rewriting into conjectures and their subgoals. Moreover, to avoid useless clauses from rewriting between unrelated subgoals, we disallow rewriting inferences which would introduce new Skolem constants into a clause. Finally, we avoid rewriting inferences which introduce variables into our conjectures, as these have to be instantiated before induction.

**Experimental setup.** We run our experiments with the following option setup: `-sa discount` to use the DISCOUNT saturation algorithm [13]; `-drc encompass` to enable encompassment demodulation [15]; and `-thsq on` to control pure theory derivations [18]. Experiments were run on computers with AMD Epyc 7502 2.5GHz processors and 1TB RAM, with each individual benchmark run given a single core. For inductive reasoning experiments, we used the UFDTLIA benchmark set from SMT-LIB [4], the TIP benchmark set [9] and the VAMPIRE inductive benchmark set [20]. We also used benchmarks from the UEQ division of TPTP [30] to test first-order reasoning.

**Evaluation of inductive reasoning.** The first part of our experiments consisted of running VAMPIRE on 1266 inductive benchmarks from the UFDTLIA, TIP and VAMPIRE benchmark sets. We used a 60-second timeout and the options `-ind struct -indoct on` to enable induction and generalisations over complex terms. We used two different simplification orderings:

<sup>5</sup>see Appendix C

<sup>6</sup><https://github.com/vprover/vampire/commit/16a38442515f8385>

<code>-indrc off</code>	<b>Sup</b>	<b>ReC</b>	<b>ReC<sub>v</sub></b>	<b>ReC<sup>→</sup></b>	<b>ReC<sub>v</sub><sup>→</sup></b>	<b>CReC</b>	<b>CReC<sub>v</sub></b>	<b>CReC<sup>→</sup></b>	<b>CReC<sub>v</sub><sup>→</sup></b>
KBO	270	290	290	290	<b>291</b>	302	<b>305</b>	302	302
LPO	290	318	320	<b>322</b>	320	331	<b>332</b>	<b>332</b>	<b>332</b>

<code>-indrc on</code>	<b>Sup</b>	<b>ReC</b>	<b>ReC<sub>v</sub></b>	<b>ReC<sup>→</sup></b>	<b>ReC<sub>v</sub><sup>→</sup></b>	<b>CReC</b>	<b>CReC<sub>v</sub></b>	<b>CReC<sup>→</sup></b>	<b>CReC<sub>v</sub><sup>→</sup></b>
KBO	270	299	299	<b>301</b>	300	305	304	<b>307</b>	<b>307</b>
LPO	290	321	319	<b>322</b>	321	<b>333</b>	<b>333</b>	330	329

Figure 6: Comparison of the **Sup** calculus with variants of the **ReC** and **CReC** calculi, using 1266 inductive benchmarks. Redundant  $\text{Ind}_G$  inference detection is disabled (resp. enabled) in the top (resp. bottom) table with option `-indrc off` (resp. `-indrc on`). Maximum rewriting depth (`-mgrwd`) is set to 3.

`-to kbo` for KBO ordering with constant weight and precedence determined by the arity of symbols; `-to lpo -sp occurrence` for the LPO ordering with a symbol precedence given by the declaration order. This LPO order is usually better at orienting recursive function axioms [21].

Our results are summarised in Figure 6, showcasing that each **ReC** and **CReC** calculi variant performs significantly better than **Sup**. Using the LPO ordering turned out to be advantageous over the KBO ordering. Performance is further improved via detection of redundant  $\text{Ind}_G$  inferences and using chaining inferences via **CReC** variants. Among each group, however, the differences in the number of solved benchmarks are minimal, and there is no calculus that is a clear winner in all configurations. Statistics reveal that redundant  $\text{Ind}_G$  inference detection used together with **ReC** was able to eliminate 79.3% of the overall 390,657,294  $\text{Ind}_G$  inferences, while redundant  $\text{Ind}_G$  inference detection in **CReC** variant runs detected 42.6% of the overall 169,457,851  $\text{Ind}_G$  inferences redundant. This suggests that both redundant  $\text{Ind}_G$  detection and chaining inferences in **CReC** variants are effective in keeping the search space small. In total, the configurations for the **ReC** and **CReC** variants solved 45 problems that no **Sup** variant could solve. Based on these results, we conclude that rewriting in inductive reasoning significantly improves upon standard superposition.

Figure 7 shows cactus plots [7] within the LPO configuration of **Sup**, **ReC** and **CReC** variants, and **ReC** and **CReC** variants with redundant  $\text{Ind}_G$  inference detection. Each plot line lists the logarithm of the time needed (vertical axis) to solve a certain number of the benchmarks (horizontal axis) individually, for a particular configuration. The left diagram shows the entire plot, and the right diagram a magnified (and rescaled) plot above 180 problems. The baseline configuration **Sup** is a bit faster than the **ReC** variants up to around 260 problems, and after that it only solves a few problems in the several seconds region. The **ReC** variants without redundant  $\text{Ind}_G$  detection are almost indistinguishable in the entire plot. The **ReC** calculus with redundant  $\text{Ind}_G$  detection is however better, while there is a greater gap between the **Sup** and **ReC** calculi and the two **CReC** variants. The **CReC** calculus with redundant  $\text{Ind}_G$  detection has the slowest growing curve, corresponding to the fastest solving times.

**Evaluation of first-order reasoning.** We also measured how our calculi behave with pure first-order problems. In particular, we have run experiments on the UEQ division of TPTP using the CASC2019 portfolio mode of VAMPIRE with a 300 seconds timeout (`--mode portfolio -sched casc_2019 -t 300`). While our calculi performed slightly worse than VAMPIRE portfolio, we managed to solve a few unique and hard UEQ problems: VAMPIRE without consequence generation could not solve: GRP664-12 (rating 0.96), COL066-1 (rating 0.79), LAT166-1 (rating 0.71), LAT156-1 (rating 0.71) and REL026-1 (rating 0.67). As such, our work is useful not only

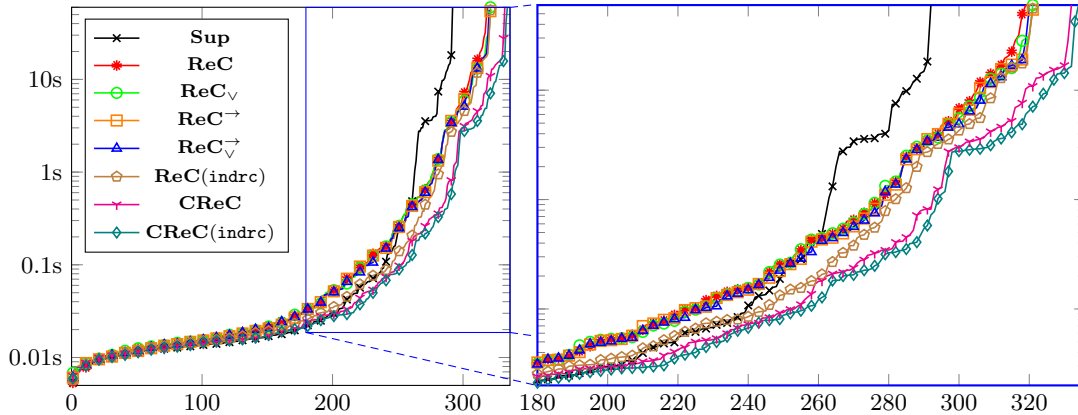


Figure 7: Plots of (logarithmic) time against number of problems that would be solved individually given that time limit. Selected configurations, all using LPO.

for inductive, but also for first-order reasoning.

## 7 Related Work and Conclusion

We improve the generation of equational consequences within saturation-based theorem proving extended with inductive reasoning. The generated consequences serve as auxiliary lemmas to be used for proving (inductive) goals.

While auxiliary lemmas might be provided by users in interactive theorem proving [27, 14], saturation-based automated theorem provers, by design, do not support user guidance during proof search. Automation of induction in saturation therefore implements inductive generalizations [28, 22, 1, 12], using failed proof attempts [8] and specialized sound inferences with common patterns [32]. Our work extends these techniques by guiding proof search with auxiliary lemmas generated during proof search. Lemma generation is also exploited in theory exploration [10, 23], without however imposing the relevance of generated lemmas with respect to a given conjecture, which is the focus of our method.

The use of equational theories and term rewriting in inductive reasoning has been addressed in [6], by ensuring termination of function definitions, and in [32, 21], by using completion procedures [5, 8] to interleave heuristic rewriting with theorem proving. Our work complements these approaches, by using rewriting-based equational reasoning for auxiliary lemma generation, and extending redundancy elimination in inductive reasoning. Our experiments have shown a significant improvement in inductive reasoning and equational first-order reasoning, solving 45 new inductive problems and 5 hard TPTP problems when compared to standard superposition.

Integrating our method with proof assistants, e.g. Sledgehammer [14], is an interesting line of future work, with the aim of splitting goals into subgoals described by auxiliary lemmas. Applying our approach to non-equational fragments, such as Horn formulas in equational logic [11], is another future challenge.

**Acknowledgements.** We thank Martin Suda for discussing ideas related to this work. This work was partially funded by the ERC Consolidator Grant ARTIST 101002685, the TU Wien Doctoral College SecInt, and the FWF SFB project SpyCoDe F8504, and the WWTF ICT22-007 grant ForSmart.



## References

- [1] Raymond Aubin. *Mechanizing Structural Induction*. PhD thesis, University of Edinburgh, UK, 1976.
- [2] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, USA, 1998.
- [3] Leo Bachmair and Harald Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In *Automated Deduction: A Basis for Applications*. Kluwer, 1998.
- [4] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). [www.SMT-LIB.org](http://www.SMT-LIB.org), 2016.
- [5] Adel Bouhoula and Michael Rusinowitch. Implicit Induction in Conditional Theories. *J. Autom. Reasoning*, 1995.
- [6] Robert S. Boyer and J. Strother Moore. *A Computational Logic Handbook*. Academic Press, 1988.
- [7] Martin Brain, James H. Davenport, and Alberto Griggio. Benchmarking solvers, sat-style. In *Proceedings of the 2nd SCSC workshop 2017*, CEUR Workshop Proceedings. CEUR-WS.org, 2017.
- [8] A. Bundy, A. Stevens, F. V. Harmelen, A. Ireland, and A. Smail. Rippling: A Heuristic for Guiding Inductive Proofs. *Artif. Intell.*, 1993.
- [9] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. TIP: Tons of Inductive Problems. In *Intelligent Computer Mathematics*. Springer International Publishing, 2015.
- [10] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating Inductive Proofs Using Theory Exploration. In *CADE*, 2013.
- [11] Koen Claessen and Nicholas Smallbone. Efficient encodings of first-order Horn formulas in equational logic. In *Automated Reasoning*. Springer International Publishing, 2018.
- [12] Simon Cruanes. Superposition with Structural Induction. In *FroCoS*, 2017.
- [13] Jörg Denzinger, Martin Kronenburg, and Stephan Schulz. DISCOUNT - A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 1997.
- [14] Martin Desharnais, Petar Vukmirović, Jasmin Blanchette, and Makarius Wenzel. Seventeen Provers Under the Hammer. In *ITP*, 2022.
- [15] André Duarte and Konstantin Korovin. Ground Joinability and Connectedness in the Superposition Calculus. In *IJCAR*, 2022.
- [16] Pamina Georgiou, Bernhard Gleiss, Ahmed Bhayat, Michael Rawson, Laura Kovács, and Giles Reger. The Rapid Software Verification Framework. In *FMCAD*. IEEE, 2022.
- [17] Bernhard Gleiss, Laura Kovács, and Jakob Rath. Subsumption demodulation in first-order theorem proving. In *Automated Reasoning*. Springer International Publishing, 2020.
- [18] Bernhard Gleiss and Martin Suda. Layered Clause Selection for Saturation-Based Theorem Proving. In *PAAR and SC-Square @ IJCAR 2020, Proceedings*. CEUR-WS.org, 2020.
- [19] Márton Hajdu, Petra Hozzová, Laura Kovács, Giles Reger, and Andrei Voronkov. Getting Saturated with Induction. In *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, 2022.
- [20] Márton Hajdu, Petra Hozzová, Laura Kovács, Johannes Schoisswohl, and Andrei Voronkov. Induction with Generalization in Superposition Reasoning. In *CICM*, 2020.
- [21] Márton Hajdu, Petra Hozzová, Laura Kovács, and Andrei Voronkov. Induction with Recursive Definitions in Superposition. In *FMCAD*, 2021.
- [22] Petra Hozzová, Laura Kovács, and Andrei Voronkov. Integer Induction in Saturation. In *CADE*, pages 361–377, 2021.
- [23] Moa Johansson, Lucas Dixon, and Alan Bundy. Conjecture Synthesis for Inductive Theories. *J. of Automated Reasoning*, 2011.
- [24] Deepak Kapur and Hantao Zhang. RRL: A rewrite rule laboratory. In *9th International Conference on Automated Deduction*. Springer Berlin Heidelberg, 1988.

- [25] Laura Kovács and Andrei Voronkov. First-Order Theorem Proving and Vampire. In *CAV*, 2013.
- [26] R. Nieuwenhuis and A. Rubio. Paramodulation-Based Theorem Proving. In *Handbook of Automated Reasoning*. MIT Press, 2001.
- [27] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. LNCS. Springer Berlin Heidelberg, 2002.
- [28] Giles Reger and Andrei Voronkov. Induction in Saturation-Based Proof Search. In *CADE*, 2019.
- [29] Rolf Socher-Ambrosius. A goal oriented strategy based on completion. In *Algebraic and Logic Programming*, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
- [30] G. Sutcliffe. The Logic Languages of the TPTP World. *Logic Journal of the IGPL*, 2022.
- [31] Christoph Walther. Computing induction axioms. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning*, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
- [32] Daniel Wand. *Superposition: Types and Induction*. PhD thesis, Saarland University, Saarbrücken, Germany, 2017.

## Appendix A

### Proof of our motivating example

1.  $0 \neq s(x)$  [nat.1]
2.  $s(x) \neq s(y) \vee x \simeq y$  [nat.2]
3.  $\text{nil} \neq \text{cons}(x, y)$  [list.1]
4.  $\text{cons}(x, y) \neq \text{cons}(z, u) \vee x \simeq z$  [list.2]
5.  $\text{cons}(x, y) \neq \text{cons}(z, u) \vee y \simeq u$  [list.3]
6.  $0 + x \simeq x$  [plus.1]
7.  $s(x) + y \simeq s(x + y)$  [plus.2]
8.  $\text{nil} @ x \simeq x$  [append.1]
9.  $\text{cons}(x, y) @ z \simeq \text{cons}(x, y @ z)$  [append.2]
10.  $[\text{nil}] \simeq 0$  [length.1]
11.  $[\text{cons}(x, y)] \simeq s([y])$  [length.2]
12.  $g(x) @ f(y) \simeq f(x) @ g(y)$  [ax.1]
13.  $[x] + [f(y)] \simeq [f(x)] + [y]$  [ax.2]
14.  $[f(g(x))] \simeq s([x])$  [ax.3]
15.  $[g(d) @ f(c)] \neq s([c] + [d])$  [nConj]
16.  $[g(d) @ f(c)] \neq s([c]) + [d]$  [Rw 15,7]
17.  $[g(d) @ f(c)] \neq [f(g(c))] + [d]$  [Rw 16,14]
18.  $[g(d) @ f(c)] \neq [g(c)] + [f(d)]$  [Rw 17,13]
19.  $[f(d) @ g(c)] \neq [g(c)] + [f(d)]$  [Rw 18,12]
20.  $[\text{nil} @ g(c)] \neq [g(c)] + [\text{nil}] \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Ind 19]
21.  $[\text{nil} @ g(c)] \neq [g(c)] + [\text{nil}] \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Ind 19]
22.  $[g(c)] \neq [g(c)] + [\text{nil}] \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Sup 20,8]
23.  $[g(c)] \neq [g(c)] + 0 \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Sup 22,10]
24.  $0 \neq 0 + 0 \vee c_3 \simeq c_3 + 0 \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Ind 23]
25.  $0 \neq 0 + 0 \vee s(c_3) \neq s(c_3) + 0 \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Ind 23]
26.  $0 \neq 0 \vee c_3 \simeq c_3 + 0 \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Sup 24,6]
27.  $c_3 \simeq c_3 + 0 \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [EqRes 26]
28.  $0 \neq 0 \vee s(c_3) \neq s(c_3) + 0 \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Sup 25,6]
29.  $s(c_3) \neq s(c_3) + 0 \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [EqRes 28]
30.  $s(c_3) \neq s(c_3 + 0) \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Sup 29,7]
31.  $s(c_3) \neq s(c_3) \vee [c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [Sup 30,27]
32.  $[c_2 @ g(c)] \simeq [g(c)] + [c_2]$  [EqRes 31]
33.  $[g(c)] \neq [g(c)] + [\text{nil}] \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Sup 21,8]
34.  $[g(c)] \neq [g(c)] + 0 \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Sup 33,10]
35.  $0 \neq 0 + 0 \vee c_4 \simeq c_4 + 0 \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Ind 34]
36.  $0 \neq 0 + 0 \vee s(c_4) \neq s(c_4) + 0 \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Ind 34]
37.  $0 \neq 0 \vee c_4 \simeq c_4 + 0 \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Sup 35,6]
38.  $c_4 \simeq c_4 + 0 \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [EqRes 37]
39.  $0 \neq 0 \vee s(c_4) \neq s(c_4) + 0 \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Sup 36,6]
40.  $s(c_4) \neq s(c_4) + 0 \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [EqRes 39]
41.  $s(c_4) \neq s(c_4 + 0) \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Sup 40,7]
42.  $s(c_4) \neq s(c_4) \vee [\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Sup 41,27]
43.  $[\text{cons}(c_1, c_2) @ g(c)] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [EqRes 42]
44.  $[\text{cons}(c_1, c_2 @ g(c))] \neq [g(c)] + [\text{cons}(c_1, c_2)]$  [Sup 43,9]

45.  $s(\llbracket c_2 @ g(c) \rrbracket) \not\approx \llbracket g(c) \rrbracket + \llbracket \text{cons}(c_1, c_2) \rrbracket$  [Sup 44,11]  
46.  $s(\llbracket g(c) \rrbracket + \llbracket c_2 \rrbracket) \not\approx \llbracket g(c) \rrbracket + \llbracket \text{cons}(c_1, c_2) \rrbracket$  [Sup 45,32]  
47.  $s(\llbracket g(c) \rrbracket + \llbracket c_2 \rrbracket) \not\approx \llbracket g(c) \rrbracket + s(\llbracket c_2 \rrbracket)$  [Sup 46,11]  
48.  $s(0 + \llbracket c_2 \rrbracket) \not\approx 0 + s(\llbracket c_2 \rrbracket) \vee s(c_5 + \llbracket c_2 \rrbracket) \simeq c_5 + s(\llbracket c_2 \rrbracket)$  [Ind 47]  
49.  $s(0 + \llbracket c_2 \rrbracket) \not\approx 0 + s(\llbracket c_2 \rrbracket) \vee s(s(c_5) + \llbracket c_2 \rrbracket) \not\approx s(c_5) + s(\llbracket c_2 \rrbracket)$  [Ind 47]  
50.  $s(\llbracket c_2 \rrbracket) \not\approx 0 + s(\llbracket c_2 \rrbracket) \vee s(c_5 + \llbracket c_2 \rrbracket) \simeq c_5 + s(\llbracket c_2 \rrbracket)$  [Sup 48,6]  
51.  $s(\llbracket c_2 \rrbracket) \not\approx s(\llbracket c_2 \rrbracket) \vee s(c_5 + \llbracket c_2 \rrbracket) \simeq c_5 + s(\llbracket c_2 \rrbracket)$  [Sup 50,6]  
52.  $s(c_5 + \llbracket c_2 \rrbracket) \simeq c_5 + s(\llbracket c_2 \rrbracket)$  [EqRes 51]  
53.  $s(\llbracket c_2 \rrbracket) \not\approx 0 + s(\llbracket c_2 \rrbracket) \vee s(s(c_5) + \llbracket c_2 \rrbracket) \not\approx s(c_5) + s(\llbracket c_2 \rrbracket)$  [Sup 49,6]  
54.  $s(\llbracket c_2 \rrbracket) \not\approx s(\llbracket c_2 \rrbracket) \vee s(s(c_5) + \llbracket c_2 \rrbracket) \not\approx s(c_5) + s(\llbracket c_2 \rrbracket)$  [Sup 53,6]  
55.  $s(s(c_5) + \llbracket c_2 \rrbracket) \not\approx s(c_5) + s(\llbracket c_2 \rrbracket)$  [EqRes 54]  
56.  $s(s(c_5 + \llbracket c_2 \rrbracket)) \not\approx s(c_5) + s(\llbracket c_2 \rrbracket)$  [Sup 55,7]  
57.  $s(s(c_5 + \llbracket c_2 \rrbracket)) \not\approx s(c_5 + s(\llbracket c_2 \rrbracket))$  [Sup 56,7]  
58.  $s(c_5 + s(\llbracket c_2 \rrbracket)) \not\approx s(c_5 + s(\llbracket c_2 \rrbracket))$  [Sup 57,52]  
59.  $\square$  [EqRes 58]

## Appendix B

### Rewriting calculi – variants of ReC

**Definition 1** (Equational derivability (ED)). Let  $\theta$  be a substitution. An inference system  $\mathcal{I}$  admits equational derivability (ED) if, for any set of equations  $\mathcal{C}$ , equation  $l \simeq r$  and clause  $D[l\theta]$ , if  $\mathcal{C} \vdash_{\mathcal{I}}^* D[l\theta]$  then  $\mathcal{C}, l \simeq r \vdash_{\mathcal{I}}^* D[r\theta]$ .

#### The ReC calculus

$$\begin{array}{l}
 \text{(Sup)} \frac{s[u] \bowtie t \vee C \quad l \simeq r \vee D}{(s[r] \bowtie t \vee C \vee D)\theta} \quad \text{where} \quad \begin{array}{l} (1) u \text{ is not a variable,} \\ (2) \theta = \text{mgu}(l, u), \\ (3) r\theta \not\approx l\theta \text{ and } t\theta \not\approx s\theta, \end{array} \\
 \\
 \text{(EqRes)} \frac{s \not\approx t \vee C}{C\theta} \quad \text{where} \quad \theta = \text{mgu}(s, t), \\
 \\
 \text{(EqFac)} \frac{s \simeq t \vee u \simeq w \vee C}{(s \simeq t \vee t \not\approx w \vee C)\theta} \quad \text{where} \quad \begin{array}{l} (1) \theta = \text{mgu}(s, u), \\ (2) t\theta \not\approx s\theta \text{ and } w\theta \not\approx t\theta, \end{array} \\
 \\
 \text{(Rw)} \frac{C[l\theta] \quad l \simeq r}{C[r\theta]}
 \end{array}$$

**Theorem 1 (ReC–ED).** *The inference system ReC admits ED.*

*Proof.* Let  $\mathcal{C}$  be a set of equations and  $l \simeq r$  an equation. By assumption we are given a derivation  $\mathcal{C} \vdash_{\text{ReC}}^* D[l\theta]$ . We trivially extend it to the following ReC-derivation:

$$\text{(Rw)} \frac{\begin{array}{c} \vdots \\ D[l\theta] \quad l \simeq r \end{array}}{D[r\theta]}$$

□

#### The ReC<sub>∇</sub> calculus

$$\begin{array}{l}
 \text{(Sup)} \frac{\square s[u] \bowtie t \vee C \quad \square l \simeq r \vee D}{\square (s[r] \bowtie t \vee C \vee D)\theta} \quad \text{where} \quad \begin{array}{l} (1) u \text{ is not a variable,} \\ (2) \theta = \text{mgu}(l, u), \\ (3) r\theta \not\approx l\theta \text{ and } t\theta \not\approx s\theta, \end{array} \\
 \\
 \text{(EqRes)} \frac{\square s \not\approx t \vee C}{\square C\theta} \quad \text{where} \quad \theta = \text{mgu}(s, t), \\
 \\
 \text{(EqFac)} \frac{\square s \simeq t \vee u \simeq w \vee C}{\square (s \simeq t \vee t \not\approx w \vee C)\theta} \quad \text{where} \quad \begin{array}{l} (1) \theta = \text{mgu}(s, u), \\ (2) t\theta \not\approx s\theta \text{ and } w\theta \not\approx t\theta, \end{array} \\
 \\
 \text{(Rw}_{\downarrow}) \frac{\square C[l\theta] \quad l \simeq r}{\square C[r\theta]} \quad \text{where } l\theta \not\approx r\theta, \quad \text{(Rw}_{\uparrow}) \frac{\blacksquare C[l\theta] \quad l \simeq r}{\blacksquare C[r\theta]} \quad \text{where } l\theta \not\approx r\theta.
 \end{array}$$

**Remark 3.** *As noted earlier, we have to allow rewriting with incomparable equations in both  $\text{Rw}_\downarrow$  and  $\text{Rw}_\uparrow$  inferences. We show an example where a rewrite with an incomparable equation after an upward rewrite is needed.*

*Let  $g(x, x) \simeq x$ ,  $f(x, y) \simeq f(y, x)$  and  $p(f(x, y))$  be clauses. Deriving  $p(g(f(x, y), f(y, x)))$  can be done in two different ways:*

$$(i) \quad \frac{\frac{\frac{\square p(f(x, y)) \quad \square g(x, x) \simeq x}{\blacksquare p(g(f(x, y), f(x, y)))} \quad \square f(x, y) \simeq f(y, x)}{\blacksquare p(g(f(x, y), f(y, x)))}}$$

$$(ii) \quad \frac{\frac{\frac{\square p(f(x, y)) \quad \square f(x, y) \simeq f(y, x)}{\square p(f(y, x))} \quad \square g(x, x) \simeq x}{\blacksquare p(g(f(y, x), f(y, x)))} \quad \square f(x, y) \simeq f(y, x)}{\blacksquare p(g(f(x, y), f(y, x)))}$$

*Notice that in both cases, we have to perform a rewrite with an incomparable equation after the upward rewrite.*

*We call an equation linear if both of its sides contain at most one occurrence of each variable. The equation  $f(x, y) \simeq f(y, x)$  is linear, while  $g(x, x) \simeq x$  is not. We can also avoid this issue if we restrict rewriting equations to be linear.*

**Theorem 2 (ReC $_{\vee}$ -ED).** *The inference system ReC $_{\vee}$  admits ED.*

*Proof.* Let  $\mathcal{C}$  be a set of equations and  $l \simeq r$  an equation. We apply well-founded induction on  $\succ$  for clauses. In particular, we assume contrary to the claim that we have a minimal (w.r.t.  $\succ$ ) clause  $\blacksquare D[l\theta]$  such that  $\mathcal{C} \vdash_{\text{ReC}_{\vee}}^* \blacksquare D[l\theta]$  but  $\mathcal{C}, l \simeq r \not\vdash_{\text{ReC}_{\vee}}^* \blacksquare D[r\theta]$ , violating equational derivability. Since this clause is a minimal clause (w.r.t.  $\succ$ ) where we cannot perform such a derivation, we have the induction hypothesis that for all  $\blacksquare D'[u\sigma] \prec \blacksquare D[l\theta]$  and all sets of equations  $\mathcal{D}$  and equation  $u \simeq w$ , if  $\mathcal{D} \vdash_{\text{ReC}_{\vee}}^* \blacksquare D'[u\sigma]$  then  $\mathcal{D}, u \simeq w \vdash_{\text{ReC}_{\vee}}^* \blacksquare D'[w\sigma]$ . As we will see, the only problematic case in the proof is when we need to perform a downward rewrite after an upward rewrite. In this case, the clause before the upward rewrite is always smaller than  $\blacksquare D[l\theta]$ , hence we can apply the induction hypothesis. We consider the following cases.

(i) If  $l\theta \not\prec r\theta$ , then we have the following derivation:

$$\vdots$$

$$(\text{Rw}_\uparrow) \frac{\blacksquare D[l\theta] \quad l \simeq r}{\blacksquare D[r\theta]}$$

Hence we could derive  $\blacksquare D[r\theta]$ , contradiction.

(ii) If  $l\theta = r\theta$ , then  $\blacksquare D[l\theta] = \blacksquare D[r\theta]$ , so we get the derivation trivially, contradiction.

(iii) If  $l\theta \succ r\theta$  and  $\blacksquare D[l\theta] = \square D[l\theta]$ , then we have the following derivation:

$$\vdots$$

$$(\text{Rw}_\downarrow) \frac{\square D[l\theta] \quad l \simeq r}{\square D[r\theta]}$$

Hence we could derive  $\blacksquare D[r\theta]$ , contradiction.

(iv) If  $l\theta \succ r\theta$  and  $\blacksquare D[l\theta] = \blacksquare D[l\theta]$ , then we have the following derivation by assumption:

$$\begin{array}{c} \vdots \\ (\text{Rw}_{\uparrow}) \frac{\blacksquare D[u\sigma] \quad u \simeq w}{\blacksquare (D[w\sigma])[l\theta]} \end{array}$$

where  $u\sigma \not\prec w\sigma$ . If  $u\sigma \not\prec w\sigma$ , then  $u\sigma$  and  $w\sigma$  are incomparable. But then, there is also a derivation:

$$\begin{array}{c} \vdots \\ (\text{Rw}_{\downarrow}) \frac{\blacksquare D[u\sigma] \quad u \simeq w}{\square (D[w\sigma])[l\theta]} \end{array}$$

This case is similar to Case (iii). Otherwise, we have  $u\sigma \prec w\sigma$  and we consider the following subcases:

(a) If  $l\theta$  and  $w\sigma$  are in parallel positions in  $\blacksquare (D[w\sigma])[l\theta] = \blacksquare D[w\sigma][l\theta]$ . Then, we have  $\mathcal{C} \vdash_{\mathbf{ReC}_{\vee}}^* \blacksquare D[u\sigma][l\theta]$  and since  $u\sigma \prec w\sigma$ ,  $\blacksquare D[u\sigma][l\theta] \prec \blacksquare D[w\sigma][l\theta]$ , the claim holds for  $\blacksquare D[u\sigma][l\theta]$ , in particular, from  $\mathcal{C} \vdash_{\mathbf{ReC}_{\vee}}^* \blacksquare D[u\sigma][l\theta]$  follows that and  $\mathcal{C} \vdash_{\mathbf{ReC}_{\vee}}^* \blacksquare D[u\sigma][r\theta]$ . Then, we get the following derivation:

$$\begin{array}{c} \vdots \\ (\text{Rw}_{\uparrow}) \frac{\blacksquare D[u\sigma][r\theta] \quad u \simeq w}{\blacksquare D[w\sigma][r\theta]} \end{array}$$

Hence we could derive  $\blacksquare D[r\theta]$ , contradiction.

(b) If  $l\theta$  is a subterm of  $w\sigma$  and is at or below a variable position in  $w$ , then there is some variable  $x$  in  $w$  s.t.  $x\sigma = t[l\theta]$  for some term  $t$ . We have  $\blacksquare D[u\sigma] \prec \blacksquare D[w\sigma]$  by assumption, hence from  $\mathcal{C} \vdash_{\mathbf{ReC}_{\vee}}^* \blacksquare D[u\sigma]$ , (by applying the induction hypothesis several times, since  $l\theta \succ r\theta$ ), we have a derivation  $\mathcal{C}, l \simeq r \vdash_{\mathbf{ReC}_{\vee}}^* \blacksquare D[u\sigma']$  where for each  $y$  in  $u$

$$\sigma'(y) = \begin{cases} \sigma(y) & \text{if } x \neq y \\ t[r\theta] & \text{otherwise} \end{cases}$$

Note that this also works if  $u$  does not contain  $x$ , because then  $u\sigma = u\sigma'$  and we can use  $\sigma'$  in the rewriting. Since  $\blacksquare D[u\sigma'] \preceq \blacksquare D[u\sigma]$ , we can apply the induction hypothesis one more time to get from  $\mathcal{C}, l \simeq r \vdash_{\mathbf{ReC}_{\vee}}^* \blacksquare D[u\sigma']$  the derivation  $\mathcal{C}, l \simeq r \vdash_{\mathbf{ReC}_{\vee}}^* \blacksquare D[w\sigma']$  (note that  $u \simeq w \in \mathcal{C}$  by assumption). Now we have  $\blacksquare D[w\sigma'] = \blacksquare D[r\theta][r\theta] \dots [r\theta]$  where some  $r\theta$ s should be  $l\theta$ s. By  $l\theta \succ r\theta$ , we restore these terms in the following derivation:

$$\begin{array}{c} \vdots \\ (\text{Rw}_{\uparrow}) \frac{\blacksquare D[r\theta][r\theta] \dots [r\theta] \quad l \simeq r}{\blacksquare D[l\theta][r\theta] \dots [r\theta]} \quad l \simeq r \\ (\text{Rw}_{\uparrow}) \frac{\blacksquare D[l\theta][r\theta] \dots [r\theta]}{\blacksquare D[l\theta][l\theta] \dots [l\theta]} \quad l \simeq r \end{array}$$



Hence we could derive  $\blacksquare D[r\theta]$ , contradiction.

- (c) If  $w\sigma$  is a subterm of  $l\theta$  and is at or below a variable position in  $l$ , then there is some variable  $x$  in  $l$  s.t.  $x\theta = t[w\sigma]$  for some term  $t$ . Consider the substitution  $\theta'$  where for each variable  $y$  in  $l$ , we have

$$\theta'(y) = \begin{cases} \theta(y) & \text{if } x \neq y \\ t[u\sigma] & \text{otherwise} \end{cases}$$

Since  $\blacksquare D[u\sigma] \prec \blacksquare (D[w\sigma])[l\theta]$ , and  $u\sigma \prec w\sigma$ , we can apply the induction hypothesis several times to get by  $\mathcal{C} \vdash_{\mathbf{ReC}_\vee}^* \blacksquare D[u\sigma]$  the derivation  $\mathcal{C} \vdash_{\mathbf{ReC}_\vee}^* \blacksquare (D[u\sigma])[l\theta']$ . Then, by  $\blacksquare (D[u\sigma])[l\theta'] \prec \blacksquare (D[w\sigma])[l\theta]$  and by the induction hypothesis and  $\mathcal{C} \vdash_{\mathbf{ReC}_\vee}^* \blacksquare (D[u\sigma])[l\theta']$ , we get a derivation  $\mathcal{C}, l \simeq r \vdash_{\mathbf{ReC}_\vee}^* \blacksquare (D[u\sigma])[r\theta']$ . Now we have the desired  $\blacksquare (D[w\sigma])[r\theta]$  except there are some  $u\sigma$ s which should be  $w\sigma$ s. By  $u\sigma \prec w\sigma$ , we restore these with the following derivation:

$$\begin{array}{c} \vdots \\ (\mathbf{RW}_\uparrow) \frac{\blacksquare (D[u\sigma])[r\theta'] \quad u \simeq w}{\blacksquare (D[w\sigma])[r\theta]} \end{array}$$

Hence we could derive  $\blacksquare D[r\theta]$ , contradiction.

- (d)  $l\theta$  is a subterm of  $w\sigma$  in a position which is above a variable position in  $w$ . Let  $l'$  be the term in  $w$  in this position. Then, there is a superposition

$$\frac{w[l'] \simeq u \quad l \simeq r}{(w[r] \simeq u)\rho}$$

where  $\rho = \mathbf{mgu}(l, l')$ ,  $\theta = \rho\mu$  for some substitution  $\mu$ , and  $\sigma = \rho\eta$  for some substitution  $\eta$  (otherwise  $l$  or  $w$  would not match the clause  $D$ ). Then,  $\blacksquare D[u\sigma] \prec \blacksquare (D[w\sigma])[l\theta]$  so the induction hypothesis applies and we get from  $\mathcal{C} \vdash_{\mathbf{ReC}_\vee}^* \blacksquare D[u\sigma]$  the derivation  $\mathcal{C}, (w[r] \simeq u)\rho \vdash_{\mathbf{ReC}_\vee}^* \blacksquare (D[w\sigma])[r\theta]$  and hence  $\mathcal{C}, l \simeq r \vdash_{\mathbf{ReC}_\vee}^* \blacksquare (D[w\sigma])[r\theta]$ . Hence we could derive  $\blacksquare D[r\theta]$ , contradiction.

- (e)  $w\sigma$  is a subterm of  $l\theta$  in a position which is above a variable position in  $l$ . Let  $w'$  be the term in  $l$  in this position. Then, there is a superposition

$$\frac{l[w'] \simeq r \quad w \simeq u}{(l[u] \simeq r)\rho}$$

where  $\rho = \mathbf{mgu}(w, w')$ ,  $\theta = \rho\mu$  for some substitution  $\mu$ , and  $\sigma = \rho\eta$  for some substitution  $\eta$  (otherwise  $l$  or  $w$  would not match the clause  $D$ ). Then,  $\blacksquare D[u\sigma] \prec \blacksquare (D[w\sigma])[l\theta]$  so the induction hypothesis applies and we get from  $\mathcal{C} \vdash_{\mathbf{ReC}_\vee}^* \blacksquare D[u\sigma]$  the derivation  $\mathcal{C}, (l[w] \simeq r)\rho \vdash_{\mathbf{ReC}_\vee}^* \blacksquare (D[w\sigma])[r\theta]$  and hence  $\mathcal{C}, l \simeq r \vdash_{\mathbf{ReC}_\vee}^* \blacksquare (D[w\sigma])[r\theta]$ . Hence we could derive  $\blacksquare D[r\theta]$ , contradiction.

We have covered all subcases of case (iv).

We have covered all cases, which proves the claim.  $\square$

**The  $\mathbf{ReC}^{\rightarrow}$  calculus**

$$\begin{array}{l}
 \text{(Sup)} \frac{ps[u] \bowtie t \vee C \quad ql \simeq r \vee D}{\epsilon(s[r] \bowtie t \vee C \vee D)\theta} \quad \text{where} \quad \begin{array}{l} (1) u \text{ is not a variable,} \\ (2) \theta = \text{mgu}(l, u), \\ (3) r\theta \not\prec_l \theta \text{ and } t\theta \not\prec_l s\theta, \end{array} \\
 \\
 \text{(EqRes)} \frac{ps \not\prec_l t \vee C}{\epsilon C\theta} \quad \text{where} \quad \theta = \text{mgu}(s, t), \\
 \\
 \text{(EqFac)} \frac{ps \simeq t \vee u \simeq w \vee C}{\epsilon(s \simeq t \vee t \not\prec_l w \vee C)\theta} \quad \text{where} \quad \begin{array}{l} (1) \theta = \text{mgu}(s, u), \\ (2) t\theta \not\prec_l s\theta \text{ and } w\theta \not\prec_l t\theta, \end{array} \\
 \\
 \text{(Rw}^{\rightarrow}) \frac{{}_p C[l\theta]_q \quad {}_{p'} l \simeq r}{{}_q C[r\theta]_q} \quad \text{where} \quad q \not\prec_l p.
 \end{array}$$

**Theorem 3** ( $\mathbf{ReC}^{\rightarrow}$ -ED). *The inference system  $\mathbf{ReC}^{\rightarrow}$  admits ED.*

*Proof.* We annotate each  $\mathbf{ReC}$  derivation  $\Pi$  with a finite sequence of positions  $S(\Pi)$  inductively as follows. If the derivation is empty, i.e. we have  $\vdash D$ , then  $S(\vdash D) = \langle \rangle$  (the empty sequence). Otherwise, we have a derivation of the following form:

$$\begin{array}{c}
 \vdots \\
 \text{(Rw)} \frac{D[l\theta]_p \quad l \simeq r}{D[r\theta]_p}
 \end{array}$$

and we have  $S(C \vdash^* D[l\theta]_p) = \langle p_1, \dots, p_n \rangle$ . Then, the sequence is  $S(C, l \simeq r \vdash^* D[r\theta]) = \langle p_1, \dots, p_n, p \rangle$ . We define a partial ordering  $\prec$  on finite sequences of positions as follows:  $\langle p_1, \dots, p_n, p, q, q_1, \dots, q_m \rangle \prec \langle p_1, \dots, p_n, q, p, q_1, \dots, q_m \rangle$  if  $p <_l q$ . It is easy to see that  $\prec$  is well-founded.

We show the claim by well-founded induction on  $\prec$ . Let  $C$  be a set of equations and  $D$  a clause. Take a  $\mathbf{ReC}$  derivation  $\Pi = C \vdash^* D$  s.t.  $S(\Pi)$  is minimal w.r.t.  $\prec$  but is not a  $\mathbf{ReC}^{\rightarrow}$  derivation. This means that there are two consecutive inferences of the following form in the derivation:

$$\begin{array}{c}
 \vdots \\
 \frac{{}_{p'} C[l\theta]_p [u\sigma]_q \quad u \simeq w}{{}_q C[l\theta]_p [w\sigma]_q} \quad l \simeq r \\
 \frac{\quad}{{}_p C[r\theta]_p [w\sigma]_q} \\
 \vdots
 \end{array}$$

where  $p <_l q$  and hence we have  $S(\Pi) = \langle p_1, \dots, p_n, p', q, p, q_1, \dots, q_m \rangle$ . Consider the derivation  $\Pi'$ :

$$\begin{array}{c}
 \vdots \\
 \frac{p' C[l\theta]_p[u\sigma]_q \quad l \simeq r}{p C[r\theta]_p[u\sigma]_q} \quad u \simeq w \\
 \frac{\quad}{q C[r\theta]_p[w\sigma]_q} \\
 \vdots
 \end{array}$$

We have that  $S(\Pi') = \langle p_1, \dots, p_n, p', p, q, q_1, \dots, q_m \rangle$ . We have  $S(\Pi') \prec S(\Pi)$ , so  $S(\Pi)$  is not the minimal, contradiction. Hence, by well-foundedness of  $\prec$  we conclude that there is a  $\mathbf{ReC}^{\rightarrow}$  derivation  $\mathcal{C} \vdash^* D$ , so  $\mathbf{ReC}^{\rightarrow}$  admits ED.  $\square$

### The $\mathbf{ReC}_{\checkmark}^{\rightarrow}$ calculus

$$(\text{Sup}) \frac{\frac{\square_p s[u] \bowtie t \vee C \quad \square_q l \simeq r \vee D}{\square_\epsilon (s[r] \bowtie t \vee C \vee D)} \theta}{\square_\epsilon (s[r] \bowtie t \vee C \vee D)} \theta \quad \text{where} \quad \begin{array}{l} (1) u \text{ is not a variable,} \\ (2) \theta = \text{mgu}(l, u), \\ (3) r\theta \not\prec_l \theta \text{ and } t\theta \not\prec_s \theta, \end{array}$$

$$(\text{EqRes}) \frac{\square_p s \not\prec t \vee C}{\square_\epsilon C\theta} \quad \text{where} \quad \theta = \text{mgu}(s, t),$$

$$(\text{EqFac}) \frac{\square_p s \simeq t \vee u \simeq w \vee C}{\square_\epsilon (s \simeq t \vee t \not\prec w \vee C)} \theta \quad \text{where} \quad \begin{array}{l} (1) \theta = \text{mgu}(s, u), \\ (2) t\theta \not\prec_s \theta \text{ and } w\theta \not\prec_t \theta, \end{array}$$

$$(\text{Rw}_{\downarrow}^{\rightarrow}) \frac{\square_p C[l\theta]_q \quad p'l \simeq r}{\square_q C[r\theta]_q} \quad \text{where} \quad \begin{array}{l} (1) l\theta \not\prec_r \theta, \\ (2) q \not\prec_l p, \end{array}$$

$$(\text{Rw}_{\uparrow}^{\rightarrow}) \frac{\blacksquare_p C[l\theta]_q \quad p'l \simeq r}{\blacksquare_q C[r\theta]_q} \quad \text{where} \quad \begin{array}{l} (1) l\theta \not\prec_r \theta, \\ (2) \blacksquare_p C[l\theta] = \square_p C[l\theta] \text{ or } q \not\prec_l p. \end{array}$$

**Theorem 4** ( $\mathbf{ReC}_{\checkmark}^{\rightarrow}$ -ED). *The inference system  $\mathbf{ReC}_{\checkmark}^{\rightarrow}$  admits ED.*

*Proof.* The proof has similar ideas as the proof of Theorem 3. Since each  $\mathbf{ReC}_{\checkmark}$  derivation is a sequence of  $\text{Rw}_{\downarrow}$  inferences followed by a sequence of  $\text{Rw}_{\uparrow}$  inferences, we can argue about the two sequences separately. First, we show that for each  $\mathbf{ReC}_{\checkmark}$  derivation, there is a derivation where each  $\text{Rw}_{\downarrow}$  inference is an  $\text{Rw}_{\downarrow}^{\rightarrow}$  inference but not each  $\text{Rw}_{\uparrow}$  inference is necessarily an  $\text{Rw}_{\uparrow}^{\rightarrow}$  inference. First, we associate a sequence of positions  $S(\Pi)$  to each derivation  $\Pi$  only based on  $\text{Rw}_{\downarrow}$  inferences. We use the same  $\prec$  ordering as in the proof of Theorem 3. Let  $\mathcal{C}$  be a set of equations and  $D$  a clause. Take a  $\mathbf{ReC}_{\checkmark}$  derivation  $\Pi = \mathcal{C} \vdash^* D$  s.t.  $S(\Pi)$  is minimal w.r.t.  $\prec$  but there is some  $\text{Rw}_{\downarrow}$  inference which is not an  $\text{Rw}_{\downarrow}^{\rightarrow}$  inference. This means that there are two consecutive  $\text{Rw}_{\downarrow}$  inferences of the following form:

$$\frac{\frac{\vdots}{\frac{\frac{\square_{p'} C[l\theta]_p[u\sigma]_q \quad u \simeq w}{\square_q C[l\theta]_p[w\sigma]_q} \quad l \simeq r}{\square_p C[r\theta]_p[w\sigma]_q}}{\vdots}}$$

where  $p <_l q$  and we have  $S(\Pi) = \langle p_1, \dots, p_n, p', q, p, q_1, \dots, q_m \rangle$ . We switch the order of inferences as follows:

$$\frac{\frac{\vdots}{\frac{\frac{\square_{p'} C[l\theta]_p[u\sigma]_q \quad l \simeq r}{\square_p C[r\theta]_p[u\sigma]_q} \quad u \simeq w}{\square_q C[r\theta]_p[w\sigma]_q}}{\vdots}}$$

We get  $\langle p_1, \dots, p_n, p', p, q, q_1, \dots, q_m \rangle < \langle p_1, \dots, p_n, p', q, p, q_1, \dots, q_m \rangle$ , hence  $S(\Pi)$  was not minimal w.r.t.  $<$ , contradiction. Hence, by well-foundedness of  $<$ , we conclude that there exists a  $\mathbf{ReC}_\downarrow$  derivation  $\mathcal{C} \vdash^* D$  where each  $\text{Rw}_\downarrow$  inference is an  $\text{Rw}_\downarrow^\rightarrow$  inference.

Next, we show that given a  $\mathbf{ReC}_\downarrow$  derivation  $\Pi = \mathcal{C} \vdash^* D$  where each  $\text{Rw}_\downarrow$  inference is an  $\text{Rw}_\downarrow^\rightarrow$  inference, there is a  $\mathbf{ReC}_\downarrow^\rightarrow$  derivation. Now, we define  $S(\Pi)$  only based on  $\text{Rw}_\uparrow$  inferences. Again, we assume there is a  $\mathbf{ReC}_\downarrow$  derivation  $\Pi = \mathcal{C} \vdash^* D$  s.t.  $S(\Pi)$  is minimal w.r.t.  $<$  but there is some  $\text{Rw}_\uparrow$  inference which is not an  $\text{Rw}_\uparrow^\rightarrow$  inference and there are two consecutive  $\text{Rw}_\uparrow$  inferences of the following form:

$$\frac{\frac{\vdots}{\frac{\frac{\blacksquare_{p'} C[l\theta]_p[u\sigma]_q \quad u \simeq w}{\blacksquare_q C[l\theta]_p[w\sigma]_q} \quad l \simeq r}{\blacksquare_p C[r\theta]_p[w\sigma]_q}}{\vdots}}$$

where  $p <_l q$  and we have  $S(\Pi) = \langle p_1, \dots, p_n, p', q, p, q_1, \dots, q_m \rangle$ . We switch the order of inferences:

$$\frac{\frac{\vdots}{\frac{\frac{\blacksquare_{p'} C[l\theta]_p[u\sigma]_q \quad l \simeq r}{\blacksquare_p C[r\theta]_p[u\sigma]_q} \quad u \simeq w}{\blacksquare_q C[r\theta]_p[w\sigma]_q}}{\vdots}}$$

We get  $\langle p_1, \dots, p_n, p', p, q, q_1, \dots, q_m \rangle < \langle p_1, \dots, p_n, p', q, p, q_1, \dots, q_m \rangle$ , hence  $S(\Pi)$  was not minimal w.r.t.  $<$ , contradiction. Hence, by well-foundedness of  $<$ , we conclude that there exists a  $\mathbf{ReC}_\downarrow^\rightarrow$  derivation  $\mathcal{C} \vdash^* D$ . Hence,  $\mathbf{ReC}_\downarrow^\rightarrow$  admits ED.  $\square$

## Appendix C

### Redundancy elimination

**Lemma 1** (Redundant  $\text{Ind}_G$  – Condition I). *Let  $l \simeq r$  and  $\bar{L}[t] \vee C$  be clauses,  $x$  a fresh variable, and  $\mathcal{I}$  an inference system that admits ED. If there is a substitution  $\theta$  s.t.  $l\theta \triangleleft L[x]$  and  $l\sigma \succ r\sigma$  where  $\sigma = \theta \cdot \{x \mapsto t\}$ , then the  $\text{Ind}_G$  inference*

$$\bar{L}[t] \vee C \vdash \text{cnf}(\neg G[L[x]] \vee C)$$

is redundant in  $\mathcal{I} \cup \{\text{Ind}_G\}$  w.r.t. the clauses  $l \simeq r$  and  $(\bar{L}[t])[l\sigma \mapsto r\sigma] \vee C$ .

*Proof.* Since  $\mathcal{I}$  admits ED, and by assumption there is a derivation  $\mathcal{C} \vdash_{\mathcal{I}}^* (L[t])[l\sigma] \vee C$  for some set of clauses  $\mathcal{C}$ . Then, there is also a derivation of the form  $\mathcal{C}, l \simeq r \vdash_{\mathcal{I}}^* (L[t])[l\sigma \mapsto r\sigma] \vee C$ . From  $l\theta \succ r\theta$ , we have  $(L[t])[l\sigma] \vee C \succ (L[t])[l\sigma \mapsto r\sigma] \vee C$ . There is also an  $\text{Ind}_G$  inference

$$(L[t])[l\sigma \mapsto r\sigma] \vee C \vdash \text{cnf}(\neg G[(L[y])[l\sigma \mapsto r\sigma]] \vee C)$$

where  $y$  is fresh. The two formulas  $\neg G[L[x]] \vee C$  and  $\neg G[(L[y])[l\sigma \mapsto r\sigma]] \vee C$  are equivalent w.r.t. the equation  $l \simeq r$ . Hence, the inference is redundant.  $\square$

**Lemma 2** (Redundant  $\text{Ind}_G$  – Condition II). *Let  $l \simeq r$  and  $\bar{L}[t] \vee C$  be clauses,  $x$  a fresh variable, and  $\mathcal{I}$  an inference system that admits ED. If there is a substitution  $\theta$  s.t.  $l\theta \trianglelefteq t$  and  $l\theta \succ r\theta$ , then the  $\text{Ind}_G$  inference*

$$\bar{L}[t] \vee C \vdash \text{cnf}(\neg G[L[x]] \vee C)$$

is redundant in  $\mathcal{I} \cup \{\text{Ind}_G\}$  w.r.t. the clauses  $l \simeq r$  and  $\bar{L}[t[l\theta \mapsto r\theta]] \vee C$ .

*Proof.* Since  $\mathcal{I}$  admits ED and there is a derivation  $\mathcal{C} \vdash_{\mathcal{I}}^* \bar{L}[t] \vee C$ , then there is a derivation  $\mathcal{C}, l \simeq r \vdash_{\mathcal{I}}^* \bar{L}[t[l\theta \mapsto r\theta]] \vee C$ . By  $l\theta \succ r\theta$ , we have  $\bar{L}[t] \vee C \succ \bar{L}[t[l\theta \mapsto r\theta]] \vee C$ . Since  $\bar{L}[t \mapsto x]$  is equivalent to  $\bar{L}[t[l\theta \mapsto r\theta] \mapsto y]$  (where  $y$  is fresh) up to variable renaming, we have following the  $\text{Ind}_G$  inference

$$\bar{L}[t[l\theta \mapsto r\theta]] \vee C \vdash \text{cnf}(\neg G[L[y]] \vee C).$$

The formula  $\neg G[L[x]] \vee C$  is equivalent to  $\neg G[L[y]] \vee C$ . Hence, we conclude that the  $\text{Ind}_G$  inference is redundant w.r.t.  $\bar{L}[t[l\theta \mapsto r\theta]] \vee C$ .  $\square$

**Lemma 3** (Redundancy of ineffective rewrites). *Let  $l \simeq r$  be an ineffective equation and  $C[l\theta]$  a ground clause. If  $C[l\theta] \succ (l \simeq r)\theta$ , the clause  $C[l\theta]$  is inductively redundant in any inference system that admits ED.*

*Proof.* By the condition  $C[l\theta] \succ (l \simeq r)\theta$  it is straightforward to show that the clause  $C[l\theta]$  is redundant w.r.t.  $C[r\theta]$  and  $l \simeq r$ , and that by the ED property  $C[r\theta]$  can be derived. Take an arbitrary induction on  $C[l\theta]$ , inducting on term  $t$ . We consider the following cases:

- (i) If  $t \triangleleft l\theta$ ,  $l \simeq r$  being ineffective, all occurrences of  $t$  inside  $l\theta$  must be under a single variable position in  $l\theta$  (since  $l$  has only one occurrence of each variable). Let  $\sigma = \{y \mapsto r[t \mapsto x] \mid y \mapsto r \in \theta\}$  for some fresh variable  $x$ . Since  $l \succ r$ , we also have  $l\sigma \succ r\sigma$  and Lemma 1 applies with  $l\sigma$ , hence the  $\text{Ind}_G$  inference is redundant.
- (ii) If  $l\theta \trianglelefteq t$ , by Lemma 2 the inference is redundant.
- (iii) Otherwise,  $l\theta$  is in a parallel position to  $t$  and Lemma 1 applies with  $l\theta$ .

$\square$

**The CReC calculus**

$$\begin{array}{l}
 \text{(Sup)} \frac{s[u] \bowtie t \vee C \quad l \simeq r \vee D}{(s[r] \bowtie t \vee C \vee D)\theta} \quad \text{where} \quad \begin{array}{l} (1) \ u \text{ is not a variable,} \\ (2) \ \theta = \text{mgu}(l, u), \\ (3) \ r\theta \not\prec l\theta \text{ and } t\theta \not\prec s\theta, \end{array} \\
 \\
 \text{(EqRes)} \frac{s \not\prec t \vee C}{C\theta} \quad \text{where} \quad \theta = \text{mgu}(s, t), \\
 \\
 \text{(EqFac)} \frac{s \simeq t \vee u \simeq w \vee C}{(s \simeq t \vee t \not\prec w \vee C)\theta} \quad \text{where} \quad \begin{array}{l} (1) \ \theta = \text{mgu}(s, u), \\ (2) \ t\theta \not\prec s\theta \text{ and } w\theta \not\prec t\theta, \end{array} \\
 \\
 \text{(CRw)} \frac{C[l\theta] \quad l \simeq r}{C[r\theta]} \quad \text{where} \quad l\theta \not\prec r\theta \text{ or } l \simeq r \text{ is effective,} \\
 \\
 \text{(Chain}_1\text{)} \frac{s[l'] \simeq t \quad l \simeq r}{(s[r] \simeq t)\theta} \quad \text{where} \quad \begin{array}{l} (1) \ \theta = \text{mgu}(l, l'), \\ (2) \ s[l'] \simeq t \text{ is ineffective,} \\ (3) \ l\theta \not\prec r\theta \text{ and } l \simeq r \text{ is effective,} \end{array} \\
 \\
 \text{(Chain}_2\text{)} \frac{s \simeq t[l'] \quad l \simeq r}{(s \simeq t[r])\theta} \quad \text{where} \quad \begin{array}{l} (1) \ \theta = \text{mgu}(l, l'), \\ (2) \ l \simeq r \text{ is ineffective,} \\ (3) \ t[l']\theta \not\prec s\theta \text{ and } s \simeq t[l'] \text{ is effective.} \end{array}
 \end{array}$$

**Lemma 4.** *Let  $\mathcal{C}$  be a set of clauses and  $D$  a clause. If there is a derivation  $\mathcal{C} \vdash_{\{\text{Rw}\}}^* D$ , then there is a derivation of the following form:*

$$\begin{array}{c}
 \text{(Rw)} \frac{C_1 \quad l_1 \simeq r_1}{\text{(Rw)} \frac{C_2 \quad l_2 \simeq r_2}{C_3}} \\
 \vdots \\
 \text{(Rw)} \frac{C_n \quad l_n \simeq r_n}{D}
 \end{array}$$

where  $l_1 \simeq r_1, \dots, l_n \simeq r_n \in \mathcal{C}$ .

*Proof.* If a derivation is not of the desired form, we have one of the following cases in the derivation:

$$\begin{array}{ll}
 \text{(i)} \quad \text{(Rw)} \frac{C[l[w\sigma]\theta] \quad \text{(Rw)} \frac{l[u\sigma] \simeq r \quad u \simeq w}{l[w\sigma] \simeq r}}{C[r\theta]} & \text{(ii)} \quad \text{(Rw)} \frac{C[l[w\sigma]\theta] \quad \text{(Rw)} \frac{l \simeq r[u\sigma] \quad u \simeq w}{l \simeq r[w\sigma]}}{C[r[w\sigma]\theta]}
 \end{array}$$

We transform the two cases into the following derivations:

$$\begin{array}{ll}
 \text{(i')} \quad \text{(Rw)} \frac{C[l[w\sigma]\theta] \quad u \simeq w}{\text{(Rw)} \frac{C[l[u\sigma]\theta] \quad l[u\sigma] \simeq r}{C[r\theta]}} & \text{(ii')} \quad \text{(Rw)} \frac{C[l\theta] \quad l \simeq r[u\sigma]}{\text{(Rw)} \frac{C[r[u\sigma]\theta] \quad u \simeq w}{C[r[w\sigma]\theta]}}
 \end{array}$$

It is easy to see that repeating this transformation results in a derivation of the desired form.  $\square$

**Theorem 5 (CReC derivability).** *Let  $D$  be an inductively non-redundant clause. If  $\mathcal{C} \vdash_{\{\text{Rw}\}}^* D$ , then  $\mathcal{C} \vdash_{\{\text{CRw}, \text{Chain}_1, \text{Chain}_2\}}^* D$ .*

*Proof.* In this proof, we call *violating inferences* any Rw inferences that are not CRw inferences. The proof is by induction on the number of violating inferences. Take an arbitrary derivation  $\mathcal{C} \vdash_{\{\text{Rw}\}}^* D$  s.t.  $D$  is an inductively non-redundant clause. Let us denote this derivation with  $\Pi$ . If  $\Pi$  does not contain any violating inferences, we are done as  $\Pi$  is also a  $\{\text{CRw}, \text{Chain}_1, \text{Chain}_2\}$  derivation.

Otherwise, by Lemma 4 we may assume that  $\Pi$  is in a form where Rw inferences are only applied on a single clause repeatedly. Take the last (in the order of Rw inferences) violating inference in  $\Pi$  rewriting term  $r\theta$  into  $l\theta$  with ineffective  $l \simeq r$  and  $l\theta \succ r\theta$ . We build a new derivation  $\Gamma$  by induction on the derivation length of  $\Pi$  s.t. it does not contain the violating inference. We claim that for each inference in  $\Pi$  resulting in some clause  $C[l\theta_1] \dots [l\theta_n]$  for some substitutions  $\theta_1, \dots, \theta_n$ , there is a derivation resulting in  $C[r\theta_1] \dots [r\theta_n]$ . Let us assume that we have a derivation  $\Gamma$  corresponding to  $n - 1$  inferences in  $\Pi$ . We take the  $n$ th inference in  $\Pi$  and consider the following cases.

- (i) If the clause  $C$  we have in  $\Pi$  is the same as the clause in  $\Gamma$ , we either have the violating inference, in which case we skip the inference and we get  $C[r\theta]$  instead of  $C[l\theta]$  or we have some other inference and we perform it, resulting in the same clause as in the conclusion of the  $n$ th inference of  $\Pi$ .
- (ii) Otherwise, in  $\Pi$  the premise of the  $n$ th step is of the form  $C[l\theta_1] \dots [l\theta_n]$  but the premise in  $\Gamma$  is  $C[r\theta_1] \dots [r\theta_n]$ . If the inference is an Rw inference parallel to all  $l\theta_i$ s in  $\Pi$  resulting in some  $C'[l\theta_1] \dots [l\theta_n]$ , then we simply perform the inference in  $\Gamma$  as well, resulting in the desired  $C'[r\theta_1] \dots [r\theta_n]$ .
- (iii) If the inference is an Rw inference on a subterm of some  $l\theta_i$  in  $\Pi$  and at or below a variable position in  $l$ , then let  $x$  be this variable. Since  $l$  has only one occurrence of  $x$ , the result is  $l\theta'_i$  for some  $\theta'_i$ . In the new derivation, we perform the same rewrite inside each occurrence of  $x$  in  $r\theta_i$ , resulting in  $r\theta'_i$ . Hence, in  $\Pi$  we get  $C[l\theta_1] \dots [l\theta'_i] \dots [l\theta_n]$ , and in  $\Gamma$  we get the desired  $C[r\theta_1] \dots [r\theta'_i] \dots [r\theta_n]$ .
- (iv) If the inference is an Rw inference with some equation  $u \simeq w$  s.t. it rewrites some subterm  $u\sigma$  above a variable position of an  $l\theta_i$  term in  $\Pi$  for some substitution  $\sigma$ , then there is an inference

$$\kappa \frac{l[u'] \simeq r \quad u \simeq w}{(l[w] \simeq r)\rho}$$

where  $\rho = \text{mgu}(u', u)$ . We have  $l[u']\rho \succ r\rho$  by assumption. If  $u\rho = w\rho$ , then the rewrite with  $u \simeq w$  in  $l\theta_i$  results in the same clause  $C[l\theta_1] \dots [l\theta_n]$  in  $\Pi$ , so by not performing the inference in  $\Gamma$  we get the desired clause  $C[r\theta_1] \dots [r\theta_n]$  trivially. Otherwise, if  $w\rho \not\simeq u\rho$ , then the inference  $\kappa$  is a **Sup** inference. If  $u\rho \not\simeq w\rho$ , then the inference  $\kappa$  is a **Chain<sub>1</sub>** inference (since  $u \simeq w$  cannot be an ineffective equation, by assumption). Hence, we can use  $(l[w] \simeq r)\rho$  to rewrite  $r\theta_i$  instead, resulting in  $C[r\theta_1] \dots [r\theta_{i-1}][r\theta_{i+1}] \dots [l\theta_n]$  as desired.

- (v) Otherwise, the inference is an Rw inference s.t. the rewritten term is the superterm of some  $l\theta_i$ s. Let  $u \simeq w$  be the rewriting equation and  $u\sigma$  be the rewritten term in  $\Pi$ . Let  $p$  be the position of  $u\sigma$  in  $C[l\theta_1] \dots [l\theta_n]$ . Let  $u'$  be the term in the same position  $p$  in  $C[r\theta_1] \dots [r\theta_n]$ .



We induct on the number of positions inside  $u'$  which prevent it from being rewritten by  $u \simeq w$  (i.e. the positions preventing  $u'$  being matched by  $u$ ). Note that these terms are all  $l\theta_i$ s. If we have an  $l\theta_i$  in a position in  $u\sigma$  which is above a variable position in  $u$ , then there is an inference

$$\kappa \frac{u[l'] \simeq w \quad l \simeq r}{(u[r] \simeq w)\rho}$$

where  $\rho = \text{mgu}(l, l')$ . Again, we have  $l\rho \succ r\rho$  by assumption. Similarly as in the previous case, if  $u[l']\rho = w\rho$ , then the rewrite with  $u \simeq w$  on  $u\sigma$  results in the same clause  $C[l\theta_1] \dots [l\theta_n]$  in  $\Pi$ , so by not performing the inference in  $\Gamma$  we get the desired clause  $C[r\theta_1] \dots [r\theta_n]$  trivially. If  $w\rho \not\equiv u[l']\rho$ , then  $\kappa$  is a **Sup** inference. If  $u[l']\rho \not\equiv w\rho$ , then  $\kappa$  is a **Chain<sub>2</sub>** inference. By using  $(u[r] \simeq w)\rho$  instead of  $u[l'] \simeq w$  for the rewrite, we get one less position where the rewrite (or match) is prevented, hence the induction hypothesis applies.

If there is no such  $l\theta_i$ , then either we have  $u\sigma'$  for some  $\sigma'$  and we can apply the rewrite resulting in  $w\sigma'$ , and we get a clause of the desired form, or  $u$  does not match  $u'$  because there are two occurrences of a variable  $x$  in  $u$  s.t. there are two distinct terms  $t_1$  and  $t_2$  in these positions in  $u'$ . This must be because  $t_1$  contains  $r\theta_i$  in some position where  $t_2$  contains  $l\theta_i$ . We rewrite all such  $l\theta_i$ s in  $u'$  into  $r\theta_i$  using downward rewrites with  $l \simeq r$ , resulting in a term that can be rewritten by  $u \simeq w$ .

We have covered all cases, proving the claim that all  $n$  inferences in  $\Pi$  can be performed or replaced by suitable inferences in  $\Gamma$  resulting in some  $D[r\theta_1] \dots [r\theta_n]$  instead of  $D = D[l\theta_1] \dots [l\theta_n]$ . But  $D$  is inductively non-redundant, so it cannot contain any such  $l\theta_i$ s. Hence, we get  $D$  as the result of  $\Gamma$  as well, with  $\Gamma$  containing one less violating inference. We apply the induction hypothesis, and get a **CReC** derivation.  $\square$

**The  $\text{CReC}_\vee$  calculus**

$\text{(Sup)} \frac{\square s[u] \bowtie t \vee C \quad \square l \simeq r \vee D}{\square (s[r] \bowtie t \vee C \vee D)\theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>u</math> is not a variable,</li> <li>(2) <math>\theta = \text{mgu}(l, u)</math>,</li> <li>(3) <math>r\theta \not\prec l\theta</math> and <math>t\theta \not\prec s\theta</math>,</li> </ol>
$\text{(EqRes)} \frac{\square s \not\prec t \vee C}{\square C\theta}$	where	$\theta = \text{mgu}(s, t)$ ,
$\text{(EqFac)} \frac{\square s \simeq t \vee u \simeq w \vee C}{\square (s \simeq t \vee t \not\prec w \vee C)\theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(s, u)</math>,</li> <li>(2) <math>t\theta \not\prec s\theta</math> and <math>w\theta \not\prec t\theta</math>,</li> </ol>
$\text{(CRw}_\downarrow) \frac{\square C[l\theta] \quad \square l \simeq r}{\square C[r\theta]}$	where	$l\theta \not\prec r\theta$ ,
$\text{(CRw}_\uparrow) \frac{\blacksquare C[l\theta] \quad \square l \simeq r}{\blacksquare C[r\theta]}$	where	<ol style="list-style-type: none"> <li>(1) <math>l\theta \not\prec r\theta</math>,</li> <li>(2) <math>l\theta \not\prec r\theta</math> or <math>l \simeq r</math> is effective.</li> </ol>
$\text{(Chain}_1) \frac{\square s[l'] \simeq t \quad \square l \simeq r}{\square (s[r] \simeq t)\theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(l, l')</math>,</li> <li>(2) <math>s[l'] \simeq t</math> is ineffective,</li> <li>(3) <math>l\theta \not\prec r\theta</math> and <math>l \simeq r</math> is effective,</li> </ol>
$\text{(Chain}_2) \frac{\square s \simeq t[l'] \quad \square l \simeq r}{\square (s \simeq t[r])\theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(l, l')</math>,</li> <li>(2) <math>l \simeq r</math> is ineffective,</li> <li>(3) <math>t[l']\theta \not\prec s\theta</math> and <math>s \simeq t[l']</math> is effective.</li> </ol>

**Theorem 6** ( $\text{CReC}_\vee$  derivability). *Given an inductively non-redundant clause  $D$ , if  $\mathcal{C} \vdash_{\{\text{Rw}_\downarrow, \text{Rw}_\uparrow\}}^* D$ , then  $\mathcal{C} \vdash_{\{\text{CRw}_\downarrow, \text{CRw}_\uparrow, \text{Chain}_1, \text{Chain}_2\}}^* D$ .*

*Proof.* Since each  $\text{Rw}_\downarrow$  and  $\text{Rw}_\uparrow$  inference is also an  $\text{Rw}$  inference, the derivation is also a  $\{\text{Rw}\}$  derivation and Theorem 5 applies. Hence we get a derivation  $\mathcal{C} \vdash_{\{\text{CRw}, \text{Chain}_1, \text{Chain}_2\}}^* D$ . From this derivation, similarly to the proof of Theorem 2, we obtain a  $\{\text{CRw}_\downarrow, \text{CRw}_\uparrow, \text{Chain}_1, \text{Chain}_2\}$  derivation.  $\square$

**The CReC<sup>→</sup> calculus**

$\text{(Sup)} \frac{ps[u] \bowtie t \vee C \quad ql \simeq r \vee D}{\epsilon(s[r] \bowtie t \vee C \vee D)\theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>u</math> is not a variable,</li> <li>(2) <math>\theta = \text{mgu}(l, u)</math>,</li> <li>(3) <math>r\theta \not\prec l\theta</math> and <math>t\theta \not\prec s\theta</math>,</li> </ol>
$\text{(EqRes)} \frac{ps \not\prec t \vee C}{\epsilon C\theta}$	where	$\theta = \text{mgu}(s, t)$ ,
$\text{(EqFac)} \frac{ps \simeq t \vee u \simeq w \vee C}{\epsilon(s \simeq t \vee t \not\prec w \vee C)\theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(s, u)</math>,</li> <li>(2) <math>t\theta \not\prec s\theta</math> and <math>w\theta \not\prec t\theta</math>,</li> </ol>
$\text{(CRw}^{\rightarrow}) \frac{pC[l\theta]_q \quad p'l \simeq r}{qC[r\theta]_q}$	where	<ol style="list-style-type: none"> <li>(1) <math>q \not\prec_l p</math>,</li> <li>(2) <math>l\theta \not\prec r\theta</math> or <math>l \simeq r</math> is effective.</li> </ol>
$\text{(Chain}_1) \frac{ps[l'] \simeq t \quad ql \simeq r}{\epsilon(s[r] \simeq t)\theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(l, l')</math>,</li> <li>(2) <math>s[l'] \simeq t</math> is ineffective,</li> <li>(3) <math>l\theta \not\prec r\theta</math> and <math>l \simeq r</math> is effective,</li> </ol>
$\text{(Chain}_2) \frac{ps \simeq t[l'] \quad ql \simeq r}{\epsilon(s \simeq t[r])\theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(l, l')</math>,</li> <li>(2) <math>l \simeq r</math> is ineffective,</li> <li>(3) <math>t[l']\theta \not\prec s\theta</math> and <math>s \simeq t[l']</math> is effective.</li> </ol>

**Theorem 7 (CReC<sup>→</sup> derivability).** *Given an inductively non-redundant clause  $D$ , if  $\mathcal{C} \vdash_{\{\text{Rw}^{\rightarrow}\}}^* D$ , then  $\mathcal{C} \vdash_{\{\text{CRw}^{\rightarrow}, \text{Chain}_1, \text{Chain}_2\}}^* D$ .*

*Proof.* Similarly to the proof of Theorem 6, for any  $\{\text{Rw}\}$  derivation, we get a  $\{\text{CRw}, \text{Chain}_1, \text{Chain}_2\}$  derivation by Theorem 5. We then use the same approach as in Theorem 3 to get the desired derivation.  $\square$

**The  $\text{CReC}_{\checkmark}^{\rightarrow}$  calculus**

$\text{(Sup)} \frac{\frac{\Box_p s[u] \bowtie t \vee C \quad \Box_q l \simeq r \vee D}{\Box_\epsilon (s[r] \bowtie t \vee C \vee D)} \theta}{\Box_\epsilon (s[r] \bowtie t \vee C \vee D) \theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>u</math> is not a variable,</li> <li>(2) <math>\theta = \text{mgu}(l, u)</math>,</li> <li>(3) <math>r\theta \not\prec l\theta</math> and <math>t\theta \not\prec s\theta</math>,</li> </ol>
$\text{(EqRes)} \frac{\frac{\Box_p s \not\prec t \vee C}{\Box_\epsilon C\theta}}{\Box_\epsilon C\theta}$	where	$\theta = \text{mgu}(s, t)$ ,
$\text{(EqFac)} \frac{\frac{\Box_p s \simeq t \vee u \simeq w \vee C}{\Box_\epsilon (s \simeq t \vee t \not\prec w \vee C)} \theta}{\Box_\epsilon (s \simeq t \vee t \not\prec w \vee C) \theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(s, u)</math>,</li> <li>(2) <math>t\theta \not\prec s\theta</math> and <math>w\theta \not\prec t\theta</math>,</li> </ol>
$\text{(CRw}_{\downarrow}) \frac{\frac{\Box_p C[l\theta]_q \quad \Box_p l \simeq r}{\Box_q C[r\theta]_q}}{\Box_q C[r\theta]_q}$	where	<ol style="list-style-type: none"> <li>(1) <math>l\theta \not\prec r\theta</math>,</li> <li>(2) <math>q \not\prec_l p</math>,</li> </ol>
$\text{(CRw}_{\uparrow}) \frac{\frac{\Box_p C[l\theta]_q \quad \Box_p l \simeq r}{\Box_q C[r\theta]_q}}{\Box_q C[r\theta]_q}$	where	<ol style="list-style-type: none"> <li>(1) <math>l\theta \not\prec r\theta</math>,</li> <li>(2) <math>\Box_p C[l\theta] = \Box_p C[l\theta]</math> or <math>q \not\prec_l p</math>,</li> <li>(3) <math>l\theta \not\prec r\theta</math> or <math>l \simeq r</math> is effective,</li> </ol>
$\text{(Chain}_1) \frac{\frac{\Box_p s[l'] \simeq t \quad \Box_q l \simeq r}{\Box_\epsilon (s[r] \simeq t) \theta}}{\Box_\epsilon (s[r] \simeq t) \theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(l, l')</math>,</li> <li>(2) <math>s[l'] \simeq t</math> is ineffective,</li> <li>(3) <math>l\theta \not\prec r\theta</math> and <math>l \simeq r</math> is effective,</li> </ol>
$\text{(Chain}_2) \frac{\frac{\Box_p s \simeq t[l'] \quad \Box_q l \simeq r}{\Box_\epsilon (s \simeq t[r]) \theta}}{\Box_\epsilon (s \simeq t[r]) \theta}$	where	<ol style="list-style-type: none"> <li>(1) <math>\theta = \text{mgu}(l, l')</math>,</li> <li>(2) <math>l \simeq r</math> is ineffective,</li> <li>(3) <math>t[l']\theta \not\prec s\theta</math> and <math>s \simeq t[l']</math> is effective.</li> </ol>

**Theorem 8** ( $\text{CReC}_{\checkmark}^{\rightarrow}$  derivability). *Given an inductively non-redundant clause  $D$ , if  $\mathcal{C} \vdash_{\{\text{Rw}_{\downarrow}^{\rightarrow}, \text{Rw}_{\uparrow}^{\rightarrow}\}}^* D$ , then  $\mathcal{C} \vdash_{\{\text{CRw}_{\downarrow}^{\rightarrow}, \text{CRw}_{\uparrow}^{\rightarrow}, \text{Chain}_1, \text{Chain}_2\}}^* D$ .*

*Proof.* Any  $\{\text{Rw}_{\downarrow}^{\rightarrow}, \text{Rw}_{\uparrow}^{\rightarrow}\}$  derivation is a  $\{\text{Rw}_{\downarrow}, \text{Rw}_{\uparrow}\}$  derivation, hence we get a  $\{\text{CRw}_{\downarrow}, \text{CRw}_{\uparrow}, \text{Chain}_1, \text{Chain}_2\}$  derivation by Theorem 6. We then use the same approach as in Theorem 4 to get the desired derivation.  $\square$