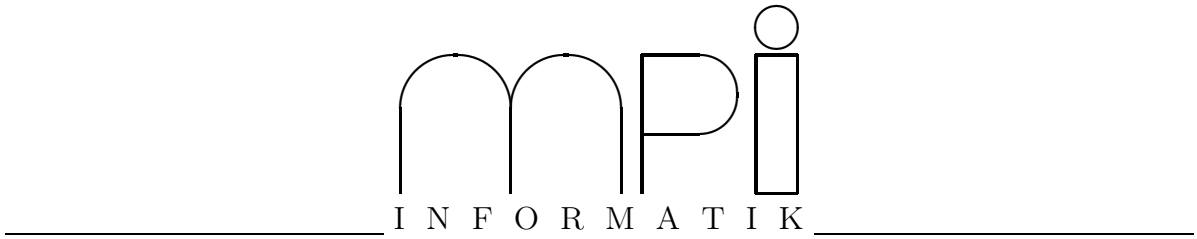


A Framework of Refutational Theorem Proving for Saturation-Based Decision Procedures

Yevgeny Kazakov

MPI–I–2005–2-004 August 2005

FORSCHUNGSBERICHT RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

Stuhlsatzenhausweg 85 66123 Saarbrücken Germany

Authors' Addresses

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany
Phone: +49 681 9325-215
Fax: +49 681 9325-299
Email: ykazakov@mpi-inf.mpg.de

Acknowledgements

The author has been supported by the International Max Planck Research School for computer science (IMPRS). I would like to thank Hans de Nivelle for reading several sections of this report and providing a valuable feedback.

Abstract

Automated state-of-the-art theorem provers are typically optimised for particular strategies, and there are only limited number of options that can be set by the user. Probably because of this, the general conditions on applicability of saturation-based calculi have not been thoroughly investigated. However for some applications, e.g., for saturation-based decision procedures, one would like to have more options in order to design flexible saturation strategies.

In this report we revisit several well-known saturation-based calculi used in automated deduction: Ordered Resolution, Ordered Paramodulation, Superposition and Chaining calculi. We give a uniform account on completeness proofs for these calculi using the standard model construction procedures of Bachmair and Ganzinger. By careful inspection of these proofs, we formulate some variations of inference rules and general conditions on orderings under which the calculi remain refutationally complete. In particular, we considerably generalise the known class of admissible orderings for the Chaining calculi.

We also consider in details the standard notion of redundancy, estimate the complexity for the steps of the clause normal form transformation, and give a computational model of the saturation process.

Keywords

Automated deduction, Theorem proving

Contents

1 Introduction

In this report we revisit the general framework of *saturation-based theorem proving* in its modern form, which is known after [Bachmair & Ganzinger, 1990, 1994]. Unfortunately, since this material is relatively new, there is almost no literature on this topic that covers all known calculi in a systematic way and gives a detailed account on *model construction* procedures and *redundancy elimination* techniques. We recommend to look into the overview papers [Bachmair & Ganzinger, 1998*a*, 2001; Nieuwenhuis & Rubio, 2001]. In this report we try to present all standard calculi and their completeness proofs in a uniform and detailed way.

Saturation-based calculi are mainly used in automated theorem provers like VAMPIRE [Riazanov & Voronkov, 2002] and SPASS [Weidenbach, Brahm, Hillenbrand, Keen, Theobalt & Topić, 2002]. These theorem provers are designed for proving or disproving first-order formulas in a fully-automated manner without a user interaction. To obtain the best performance, such provers employ particular saturation strategies which are tightly connected with the indexing data structures used in the provers. For example, many theorem provers support only few types of orderings and use data structures which allow one to quickly retrieve maximal elements w.r.t. these orderings.

Optimisations can considerably boost up the performance of a prover which makes it useful for solving general first-order problems. However such optimisations limit the flexibility of the prover: it is often not possible to alter the default strategy significantly using, say, custom orderings and selection functions. But why should one change these parameters if a prover performs well with the default ones? It is often the case that one needs to apply a theorem prover for a rather restricted class of problems, say for reasoning problems in description logics. Then it is possible, at least theoretically, to tweak the strategy for this particular class of problems. In many cases one can come up with a procedure that can *decide* the given class of formulas, or even guarantee some *complexity bounds* for the memory consumption and the running time of the procedure. Such procedures that use theorem-proving calculi for deciding classes of formulas are called *saturation-based decision procedures*.

Saturation-based decision procedures have been studied starting from works of Joyner Jr. [1976] who described several saturation strategies that decide some well-known fragments of first-order logics. Later, his technique has been extended to many clause classes [Tammet, 1990; Fermüller, Leitsch, Tammet & Zamov, 1993; Hustadt & Schmidt, 1999; de Nivelle, 2000], modal and description logics [Schmidt, 1997; Hustadt, 1999; Ganzinger, Hustadt, Meyer & Schmidt, 2001; Hustadt, Motik & Sattler, 2004] and fragments of first-order logics [Bachmair, Ganzinger & Waldmann, 1993; Ganzinger & de Nivelle, 1999; de Nivelle & Pratt-Hartmann, 2001; de Nivelle & de Rijke, 2003].

Saturation-based decision procedures for expressive fragments of first-order logics typically make use of several refinements of saturation-based calculi, such as *redundancy elimination techniques* and *basic strategies*. Designing of powerful saturation-based procedures is highly dependant on *flexibility* of the underlying calculi. Hence the primary goal of this report is to study *general conditions* on applicability of standard calculi, which we do by careful analysis of completeness proofs of [Bachmair & Ganzinger, 1990, 1994]. Because most of the results in this report have been obtained by using the standard techniques of Bachmair & Ganzinger [1990, 1994], we do not claim novelty of the work presented here. Instead of this, our work should be regarded as a reference to saturation-based calculi, in particular, for saturation-based decision procedures. However, there are also some new results in this report. These include some generalisations for the class of admissible orderings for the *chaining calculi* introduced by [Bachmair & Ganzinger, 1995, 1998*b*]. These generalisations make it possible to obtain saturation-based decision procedures for extensions of the guarded fragment with compositional axioms [see Kazakov, 2005].

Another emphasis in this report is put on the notion of *redundancy*. In theorem proving, redundancy is used to justify certain simplification techniques which have a considerable impact on the performance of theorem provers. As has been mentioned above, redundancy elimination techniques also play a significant rôle in many saturation-based decision procedures, where it allows one to handle potentially dangerous situations.

This report is organised as follows. We start with standard logical preliminaries in section 2, where the relevant material about first-order logic and term rewriting is given. In section 3 we introduce the resolution calculus, on which we demonstrate the main theoretical aspects of refutational theorem proving: model construction, refinements, redundancy and lifting. Then we show how this approach can be extended to equational theories in section 4, and theories of transitive and compositional relations in section 5. After that we focus on complexity issues: In section 6 we consider the clause normal form transformations in details, and, in section 7 we formulate some useful simplification rules and give a computational model of the saturation process.

2 Logical Preliminaries

Before proceeding to preliminary material, we make several conventions about abbreviations and notations used throughout this report.

We assume the usual notation for sets: $\{a, b, \dots\}$ denotes a set consisting of elements a, b,..., $\{\}$ is the *empty set*, 2^S denotes a *powerset* of S (the set of subsets of S), $\{S|P\}$ denotes a set of elements from S which have the property P, $\sharp S$ is the *cardinality* of S. The usual set-theoretic operations include: $\cup$ *(union)*, $\cap$ *(intersection)*, $\setminus$ *(difference)*, we also denote by $S_1 \sqcup S_2$ the *disjoint union* which is the same as $S_1 \cup S_2$, but additionally expresses that $S_1 \cap S_2 = \{\}$. Here and everywhere else $=$ is the *syntactic equality*.

We also adopt some notation for recursively defined sets and functions from functional programming (which could be better understood from examples than explained here). Finally the following abbreviations should be expanded as:

w.r.t.	$\Rightarrow$	*"with respect to"*
w.l.o.g.	$\Rightarrow$	*"without loss of generality"*
iff	$\Rightarrow$	*"if and only if"*

2.1 First-Order Logic

In this section we give basic definitions and state important facts about the first-order predicate logic. A more detailed introduction to some material in this section can be found in standard logical textbooks, e.g., in [Fitting, 1996].

2.1.1 Syntax of first-order logic

A *first-order signature* is a triple $\Sigma = (\mathrm{Pre}, \mathrm{Fun}, \mathrm{Var})$, where Pre is a set of *first-order predicate symbols*, Fun is a set of *first-order functional symbols* and Var is a set of *first-order variables*. Every predicate symbol $p \in \mathrm{Pre}$ and every functional symbol $f \in \mathrm{Fun}$ is assigned with a unique integer $ar(p) \geq 0$, $ar(f) \geq 0$ called the *arity* of a predicate/functional symbol. Functional symbols f with $ar(f) = 0$ are usually called *constants*. A signature $\Sigma = (\mathrm{Pre}, \mathrm{Fun}, \mathrm{Var})$ is *relational*, if $\mathrm{Fun} = \{\}$. Let us fix some signature Σ.

The set of *first-order terms* over a signature Σ is recursively defined using the grammar:

$$\mathrm{Tm}_\Sigma ::= x \mid f(t_1,..,t_n) \ . \tag{1}$$

where $x \in \mathrm{Var}$, $f \in \mathrm{Fun}$, $n = ar(f)$ and $t_i \in \mathrm{Tm}_\Sigma$ for $1 \leq i \leq n$ are already constructed terms. In other words, the set of first-order terms over Σ is the smallest set containing all variables from Var and is closed under application of the constructor $f(\cdot,..,\cdot)$ with $ar(f)$ positions for terms. Similarly, the set of *first-order*

atoms over a signature Σ is defined by the grammar:

$$\mathrm{At}_\Sigma ::= p(t_1,..,t_m)\ . \tag{2}$$

where $p \in \mathrm{Pre}$ is a predicate symbol with $m = ar(p)$ and $t_j \in \mathrm{Tm}_\Sigma$ for $1 \leq j \leq m$.

The set of *first-order formulas* over Σ is defined recursively by the following grammar:

$$\mathrm{Fm}_\Sigma ::= A \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \neg F_1 \mid \forall y.F_1 \mid \exists y.F_1\ . \tag{3}$$

where $A \in \mathrm{At}_\Sigma$, $F_i \in \mathrm{Fm}_\Sigma$, $i = 1, 2$, and $y \in \mathrm{Var}$. Symbols $\vee, \wedge, \neg$ called respectively the *disjunction*, the *conjunction* and the *negation*, are the *boolean connectives*. The symbols $\forall$ and $\exists$ are called the *universal quantifier* and the *existential quantifier*. We will make use of additional abbreviations: $F_1 \rightarrow F_2$ stands for $\neg F_1 \vee F_2$; $\between$ stands for either conjunction *or* disjunction and Q stands for either the universal *or* the existential quantifier. The vector $\overline{x}$ represents some sequence of variables $x_1,..,x_k$ for $k \geq 0$, where $x_i \in \mathrm{Var}$, $1 \leq i \leq k$. In such a case, the formula $\exists\overline{x}.F$ ($\forall\overline{x}.F$) is syntactical sugar for $\exists x_1.\exists x_2...\exists x_k.F$ ($\forall x_1.\forall x_2...\forall x_k.F$).

We adopt the following notation for the introduced objects. We will usually use the letters (perhaps with indices) x, y, z – for variables; f, g, h – for functional symbols; c – for constants; a, b, p, q – for predicate symbols; r, s, t – for terms; A, B, P, Q, R – for atoms; F, G, H – for formulas. In the form these symbols are written here, they refer to a type of an element (meaning *some* functional symbol, predicate symbol, atom etc.), rather than to a concrete element. So, in some sense these letters are *meta-variables* of the respective types. We use this notation in recursive definitions (like those given above) or for describing transformation procedures. For instance, by writing $p(x) \vee q(y)$ we *do not* mean that p and q are distinct predicate symbols and x and y are disjoint variables. When we use typewriting font for these letters: `x`, `f`, `a`, `A`, `F`, etc., we refer to *particular* elements. This will be usually used in examples, where, say `x` stands for a fixed variable name, `a` – for a fixed predicate symbol and `A` – for some fixed atom.

We use the pairs of parenthesis (..) and [..] to indicate the order in which the connectives are used in the construction of a first-order formula: $\forall\mathtt{y}.(\mathtt{a}(\mathtt{x}) \rightarrow [\mathtt{b}(\mathtt{x}) \wedge \mathtt{c}(\mathtt{y})])$. However, for conciseness, we may omit some parenthesis. In such situations we assume the following precedences on the first-order constructors from highest to lowest: $\forall, \exists, \neg, \wedge, \vee, \rightarrow$. In case of ambiguity, parenthesis are restored starting with symbols with the higher precedence. For example the formula $\forall\mathtt{x}.\neg\mathtt{A} \rightarrow \mathtt{B} \wedge \neg\mathtt{C} \vee \mathtt{D}$ shorthands $([\forall\mathtt{x}.(\neg\mathtt{A})] \rightarrow [(\mathtt{B} \wedge [\neg\mathtt{C}]) \vee \mathtt{D}])$. However, this example demonstrates a rather pathological case, which would likely not appear in real formulas.

A signature Σ may contain a distinguished binary predicate symbol $\simeq$ which is called the *equality predicate*. In this case we deal with the *first-order logic with equality*. We use the *infix notation* for *equational atoms*: $s \simeq t$, which is not distinguished from $t \simeq s$. The *negation* of an equational atom is denoted by $s \not\simeq t$.

Throughout this theses we will usually omit the prefix "first-order" when speaking about terms, atoms and formulas of first-order logic. We will often give recursive definitions for functions and prove properties by induction over (1), (2) and (3). Below we define some useful recursive functions and relations.

The *size* $|t|$, $|A|$, $|F|$, of a *first-order term* t, *atom* A and *formula* F are defined recursively over definitions (1), (2) and (3) as follows:

$$\begin{array}{l|l|l} |t| := & |x| = 1 & \\ & |f(t_1,..,t_n)| = 1+|t_1|+\cdots+|t_n| \ . & \\ \hline |A| := & & \\ & |p(t_1,..,t_m)| = 1+|t_1|+\cdots+|t_m| \ . & \end{array} \qquad \begin{array}{l|l} |F| := |A| = |A| & \\ |\neg F_1| = |F_1|+1 & \\ |F_1 \between F_2| = |F_1|+|F_2|+1| & \\ |Qy.F_1| = |F_1|+1 \ . & \end{array} \tag{4}$$

The *subterm* relation $s \trianglelefteq t$, $s \trianglelefteq F$ and *subformula* relation $G \trianglelefteq F$ for terms s, t and formulas G and F are defined as follows:

$$\begin{array}{rll|rll} s \trianglelefteq t := s \trianglelefteq s, & \textit{always} & | & G \trianglelefteq F := G \trianglelefteq G, & \textit{always} & | \\ s \trianglelefteq f(t_1,..,t_n), & \textit{if } s \trianglelefteq t_1 \textit{ or .. or } s \trianglelefteq t_n & | & G \trianglelefteq F_1 \between F_2, & \textit{if } G \trianglelefteq F_1, & \\ s \trianglelefteq p(t_1,..,t_m), & \textit{if } s \trianglelefteq t_1 \textit{ or .. or } s \trianglelefteq t_m & | & & \textit{or } G \trianglelefteq F_2 & | \\ s \trianglelefteq F_1 \between F_2, & \textit{if } s \trianglelefteq F_1 \textit{ or } s \trianglelefteq F_2 & | & G \trianglelefteq \neg F_1, & \textit{if } G \trianglelefteq F_1 & | \\ s \trianglelefteq \neg F_1, & \textit{if } s \trianglelefteq F_1 & | & G \trianglelefteq Qy.F_1, & \textit{if } G \trianglelefteq F_1 \ . & \\ s \trianglelefteq Qy.F_1, & \textit{if } s \trianglelefteq F_1 \ . & & & & \end{array} \tag{5}$$

The relation $\triangleleft$ is the *strict* variant of $\trianglelefteq$ (i.e., $\triangleleft$ is $\trianglelefteq \cap \neq$); the relations $\trianglerighteq$ and $\triangleright$ are the symmetric variants of respectively $\trianglelefteq$ and $\triangleleft$.

An *occurrence* of a variable x is *free* in a formula F, if there is no subformula of the form $Qx.F'$ of F containing this occurrence. Otherwise the occurrence of x is *bounded*. A variable x is *free* in F if x has a free occurrence in F. The set of *free variables of a formula* of F is denoted by $\mathit{free}[F]$, whereas the set of *all variables* of F is denoted by $\mathit{vars}[F]$ (note that $\mathit{free}[F] \subseteq \mathit{vars}[F]$ but not the other way around). The *width* $\mathit{width}(F)$ of a first-order formula F is the maximal number of free variables in a subformula of F: $\mathit{width}(F) = \max \sharp\{\mathit{free}[G] \mid G \trianglelefteq F\}$.

We write $F[G]$ ($F[s]$, $h[s]$) to denote a respective formula or a term with *indicated occurrences* of its subformula G (or its subterm s). $F[G/H]$ ($F[s/t]$, $h[s/t]$) denotes the result of replacing all these occurrences by a formula H (term t). When replaced occurrences are clear from the context, we shorten this to $F[H]$ ($F[t]$, $h[t]$). The *polarity* $\mathit{pol}(F[H])$ of an occurrence of a formula H in F is a

value from $\{1, 0\}$ (1 – for *positive*, 0 – for *negative*, that is determined as follows:

$$\begin{array}{rcll} pol(F[G]) := \quad pol(G[G]) & = & 1 & | \\ pol(F_1[G] \between F_2) & = & pol(F_1[G]) & | \\ pol(F_1 \between F_2[G]) & = & pol(F_2[G]) & | \\ pol(\neg F_1[G]) & = & 1 - pol(F_1[G]) & | \\ pol(\, Qy.F_1[G]) & = & pol(F_1[G]) \ . & \end{array} \tag{6}$$

That is, (***i***) a formula has a positive occurrence in itself; (***ii***) polarity of occurrences are not changed when applying conjunction, disjunction or quantification and (***iii***) polarity is flipped if negation is applied.

2.1.2 Semantics of first-order logic

The semantics for first-order formulas is defined by means of first-order interpretations. Given a signature Σ, a *first-order interpretation* (sometimes called a Σ-*structure*) is a pair $\mathcal{I} = (\boldsymbol{D}, \cdot^{\mathcal{I}})$, where $\boldsymbol{D}$ is a non-empty set called the *domain* of the interpretation, and $\cdot^{\mathcal{I}}$ is a mapping that associates (***i***) to every functional symbol $f \in \mathrm{Fun}$ with $n = ar(f)$ a function $f^{\mathcal{I}} : \boldsymbol{D}^n \to \boldsymbol{D}$; (***ii***) to every non-equality predicate symbol $p \in \mathrm{Pre} \setminus \{\simeq\}$ with $m = ar(p)$ a relation $p^{\mathcal{I}} \subseteq \boldsymbol{D}^m$.

Let $\Sigma = (\mathrm{Pre}, \{\}, \mathrm{Var})$ be a relational signature and $\mathcal{I} = (\boldsymbol{D}, \cdot^{\mathcal{I}})$ be a Σ-interpretation. A *restriction* of the interpretation $\mathcal{I}$ to a non-empty subdomain $\boldsymbol{D}' \subseteq \boldsymbol{D}$, $\boldsymbol{D}' \neq \{\}$ is a Σ-interpretation $\mathcal{I}' = \mathcal{I}_{|\boldsymbol{D}'} = (\boldsymbol{D}', \cdot^{\mathcal{I}'})$ such that $p^{\mathcal{I}'} = p^{\mathcal{I}} \cap \boldsymbol{D}'^m$ for every $p \in \mathrm{Pre}$, where $m = ar(p)$. In this case we say that $\mathcal{I}$ is an *extension* of $\mathcal{I}'$.

A *(variable) valuation* is a mapping $\eta : \mathrm{Var} \to \boldsymbol{D}$. For any $x \in \mathrm{Var}$ and $d \in \boldsymbol{D}$, let $\{x \mapsto d\}\cdot\eta$ denote the valuation for which $\eta'(x) = d$ and $\eta'(y) = \eta(y)$ for $x \neq y$. The *value* $[t]^{\mathcal{I}}_{\eta} \in \boldsymbol{D}$ of a *term* $t \in \mathrm{Tm}_{\Sigma}$ under an interpretation $\mathcal{I}$ with a valuation η is defined recursively over the term structure (1):

$$\begin{array}{rcll} [t]^{\mathcal{I}}_{\eta} := \quad [x]^{\mathcal{I}}_{\eta} & = & \eta(x) & | \\ [f(t_1,..,t_n)]^{\mathcal{I}}_{\eta} & = & f^{\mathcal{I}}([t_1]^{\mathcal{I}}_{\eta},..,[t_n]^{\mathcal{I}}_{\eta}) \ . & \end{array} \tag{7}$$

The *truth value* $[F]^{\mathcal{I}}_{\eta} \in \{\mathtt{true}, \mathtt{false}\}$ for a formula $F \in \mathrm{Fm}_{\Sigma}$ under $\mathcal{I}$ and η is

defined recursively over the definitions for atoms (2) and for formulas (3):

$$
\begin{array}{rrcll}
[A]^{\mathcal{I}}_{\eta} := [p(t_1,..,t_m)]^{\mathcal{I}}_{\eta} & = & \mathtt{true} & \textit{iff} & ([t_1]^{\mathcal{I}}_{\eta},..,[t_m]^{\mathcal{I}}_{\eta}) \in p^{\mathcal{I}} \quad | \\
[t_1 \simeq t_2]^{\mathcal{I}}_{\eta} & = & \mathtt{true} & \textit{iff} & [t_1]^{\mathcal{I}}_{\eta} = [t_2]^{\mathcal{I}}_{\eta} \ . \\
[F]^{\mathcal{I}}_{\eta} := \quad [A]^{\mathcal{I}}_{\eta} & = & \textit{given} & & \quad | \\
[F_1 \vee F_2]^{\mathcal{I}}_{\eta} & = & \mathtt{true} & \textit{iff} & [F_1]^{\mathcal{I}}_{\eta} = \mathtt{true} \quad \textit{or} \quad [F_2]^{\mathcal{I}}_{\eta} = \mathtt{true} \quad | \\
[F_1 \wedge F_2]^{\mathcal{I}}_{\eta} & = & \mathtt{true} & \textit{iff} & [F_1]^{\mathcal{I}}_{\eta} = \mathtt{true} \quad \textit{and} \quad [F_2]^{\mathcal{I}}_{\eta} = \mathtt{true} \quad | \\
[\neg F_1]^{\mathcal{I}}_{\eta} & = & \mathtt{true} & \textit{iff} & [F_1]^{\mathcal{I}}_{\eta} = \mathtt{false} \quad | \\
[\forall y.F_1]^{\mathcal{I}}_{\eta} & = & \mathtt{true} & \textit{iff} & [F_1]^{\mathcal{I}}_{\{y \mapsto d\}\cdot\eta} = \mathtt{true} \quad \textit{for all} \quad d \in \boldsymbol{D} \quad | \\
[\exists y.F_1]^{\mathcal{I}}_{\eta} & = & \mathtt{true} & \textit{iff} & [F_1]^{\mathcal{I}}_{\{y \mapsto d\}\cdot\eta} = \mathtt{true} \quad \textit{for some} \quad d \in \boldsymbol{D} \ .
\end{array}
\tag{8}
$$

A first-order formula F is *satisfiable in an interpretation* $\mathcal{I}$, if there exists a valuation η such that $[F]^{\mathcal{I}}_{\eta} = \mathtt{true}$. In this case $\mathcal{I}$ is a *model* for F. We usually denote models of formulas by the letter $\mathcal{M}$, possibly with indices. A formula F is *satisfiable* if it is satisfiable in some interpretation. The dual notion to satisfiability is validity. A formula F is *valid in an interpretation* $\mathcal{I}$ (notation: $\mathcal{I} \vDash F$, if $[F]^{\mathcal{I}}_{\eta} = \mathtt{true}$ for *every* valuation η. A formula F is *valid* (in symbols: $\vDash F$) if F is valid in every interpretation $\mathcal{I}$. The following proposition expressing the duality between the notions of satisfiability and validity can be easily proven by induction over definition (8):

Proposition 2.1. *For every first-order formula F, F is valid iff $\neg F$ is not satisfiable.*

A formula G is a *logical consequence* of a formula F (notation: $F \vDash G$), if for every interpretation $\mathcal{I}$ and valuation η, $[F]^{\mathcal{I}}_{\eta} = \mathtt{true}$ implies that $[G]^{\mathcal{I}}_{\eta} = \mathtt{true}$. A formula G is *(logically) equivalent* to F (notation: $G \equiv F$) if both formulas are logical consequences of each other. Formulas F and G are *equisatisfiable* when F is satisfiable *iff* G is satisfiable.

We will often extend signatures by adding new predicate or functional symbols. This requires modification of interpretations in such a way that satisfiability of formulas over the old signature is preserved.

Definition 2.2 (Conservative). A signature $\Sigma' = (\mathrm{Pre}', \mathrm{Fun}', \mathrm{Var}')$ is called an *extension* of a signature $\Sigma = (\mathrm{Pre}, \mathrm{Fun}, \mathrm{Var})$, if $\mathrm{Pre} \subseteq \mathrm{Pre}'$, $\mathrm{Fun} \subseteq \mathrm{Fun}'$ and $\mathrm{Var} \subseteq \mathrm{Var}'$. In such a situation, we say that a Σ'-interpretation $\mathcal{I}' = (\boldsymbol{D}', \cdot^{\mathcal{I}'})$ is an *expansion* of a Σ-interpretation $\mathcal{I} = (\boldsymbol{D}, \cdot^{\mathcal{I}})$, if (***i***) $\boldsymbol{D} = \boldsymbol{D}'$ and (***ii***) $f^{\mathcal{I}'} = f^{\mathcal{I}}$, $p^{\mathcal{I}'} = p^{\mathcal{I}}$ for every functional symbol $f \in \mathrm{Fun} \subseteq \mathrm{Fun}'$ and every predicate symbol $p \in \mathrm{Pre} \subseteq \mathrm{Pre}'$.

We say that a formula F' is *conservative over* a formula F if (***i***) F is a logical consequence of F' and (***ii***) every model of F can be expanded to a model of F'. ⋘

The following proposition is an easy consequence of the definitions above:

Proposition 2.3. *Let F' be a formula that is conservative over a formula F. Then (**i**) F' and F are equisatisfiable and (**ii**) for any formula F'', if F'' is conservative over F', then F'' is conservative over F.*

In other words, Proposition 2.3 says that the result of any conservative transformation is always equisatisfiable with its input and that a composition of several conservative transformations is again a conservative transformation. These properties shall be often used in transformation procedures for first-order formulas that we consider later. A particularly useful transformation is a replacement of occurrences of a subformula by some other formula: $F[G] \Rightarrow F[G/H]$. Under certain conditions one can show that the result of the transformation is a logical consequence of its input:

Lemma 2.4 (Replacement Lemma). *Let $F[G]$ be a first-order formula with indicated positive occurrences of a subformula G. Let $\mathcal{M}$ be a model for $F[G]$ and H be a formula such that $\mathcal{M} \models G \rightarrow H$. Then $\mathcal{M}$ is also a model for the formula $F[G/H]$.*

Proof. The lemma can be straightforwardly shown by induction over the structure of the formula $F[G]$ using definition (6) for polarity of a subformula. □

2.2 First-Order Clause Logic

Most automated theorem provers (ATPs) for first-order logic do not operate directly with formulas, but with their simpler clause normal forms. A *(first-order) literal* L is an atom A or a negation of an atom $\neg A$. Two literals A and $\neg A$ are said to be *complementary*. Lt_Σ denotes the set of all literals constructed over a signature Σ. A *clause* is a disjunction of literals $C = L_1 \vee \cdots \vee L_k$. The set of all clauses is denoted by Cl_Σ. A clause C is interpreted as the first-order formula $\forall \overline{x}.C$, where $\overline{x}$ are all variables of C: $\overline{x} = vars[C]$. In other words, all variables of a clause C are *implicitly universally quantified*. Hence, C is `true` in an interpretation $\mathcal{I}$, if $\mathcal{I} \models \forall \overline{x}.C$. A *clause set* $N \subseteq \mathrm{Cl}_\Sigma$ is `true` in an interpretation $\mathcal{I}$ if every clause C from N is `true` in $\mathcal{I}$.

It is possible to give an *effective* conservative transformation converting any first-order formula to a clause set (this will be discussed in detail in section 6). This conversion, together with Proposition 2.3 allows one to reduce the validity problem $\models^? F$ for a first-order formula F to the satisfiability problem for a clause set: *"Given a clause set N check if it is satisfiable in some interpretation"*. With this problem we are concerned in the rest of this section.

As long as we stay within clause logic, models can be restricted to those of a very special form that are called Herbrand models.

A *term/atom/literal or* a *clause* is called *ground* if it contains no variables. We assume that a first-order signature $\Sigma = (\mathrm{Pre}, \mathrm{Fun}, \mathrm{Var})$ contains at least one constant (otherwise we add some fixed constant $\mathtt{c}_0$), so the set Tm^0_Σ of *ground terms* over Σ is not empty: $\{\} \neq \mathrm{Tm}^0_\Sigma \subseteq \mathrm{Tm}_\Sigma$. The sets of ground atoms and ground literals are denoted respectively by At^0_Σ and Lt^0_Σ.

First we restrict ourselves to the clause logic *without equality*. In such a case, a *Herbrand* Σ*-interpretation* is an interpretation $\mathcal{H} = (\mathrm{Tm}^0_\Sigma, \cdot^{\mathcal{H}})$, where $f^{\mathcal{H}}(t^0_1,..,t^0_n) = f(t^0_1,..,t^0_n) \in \mathrm{Tm}^0_\Sigma$ for every $f \in \mathrm{Fun}$ and $t^0_i \in \mathrm{Tm}^0_\Sigma$, $i = 1,..,n$. That is, the domain of a Herbrand interpretation (also known as a *Herbrand base* and a *Herbrand universe*) is the set of all ground terms over Σ, and functional symbols are interpreted in a *canonical way*: the result of application of a function to ground terms is a ground term constructed from these elements. In particular, this implies that for any ground term $t^0 \in \mathrm{Tm}^0_\Sigma$ and any variable valuation $* : \mathrm{Var} \to \mathrm{Tm}^0_\Sigma$, the value of $[t^0]^{\mathcal{H}}_* = t^0$.

Remark 2.5. A Herbrand interpretation for a signature Σ without equality can be uniquely represented by a subset I of ground atoms At^0_Σ over Σ. Indeed, the only parameter of a Herbrand interpretation $\mathcal{H}$ that is not fixed, is the interpretation of predicate symbols, which can be represented using I as follows: $(t^0_1,..,t^0_m) \in p^{\mathcal{H}}$ *iff* $p(t^0_1,..,t^0_m) \in I$. Such representations of Herbrand models will be used in section 3, where we prove completeness for the ordered resolution calculus. ⋘

The following is a variant of the fundamental *Herbrand theorem* formulated for clause logic[1]:

Theorem 2.6 (Herbrand Theorem). *Every satisfiable clause set N has a Herbrand model.*

Proof. We prove Theorem 2.6 for the clause logic without equality and later we will show how to extend this proof for equality Herbrand interpretations (which will be defined).

Let $\mathcal{M} = (\boldsymbol{D}, \cdot^{\mathcal{M}})$ be a model of N. We construct a Herbrand model $\mathcal{H}$ of N from $\mathcal{M}$. Let us fix some valuation $* : \mathrm{Var} \to \boldsymbol{D}$ of variables. For every predicate symbol $p \in \mathrm{Pre}$ and ground term $t^0_i \in \mathrm{Tm}^0_\Sigma$, $1 \leq i \leq m = ar(p)$, we define

$$(t^0_1,..,t^0_m) \in p^{\mathcal{H}} \quad \textit{iff} \quad ([t^0_1]^{\mathcal{M}}_*,..,[t^0_m]^{\mathcal{M}}_*) \in p^{\mathcal{M}} \tag{9}$$

Note that this definition does not really depends on the choice of $*$, since all terms are ground. To put this definition differently, $\mathcal{H}$ corresponds to the set of ground atoms from At_Σ that are `true` in $\mathcal{M}$ (see Remark 2.5).

[1]The original Herbrand theorem was formulated for provability of first-order formulas. There are many different variations of this theorem in literature [see e.g., Fitting, 1996]

We claim that $\mathcal{H}$ is a model for N. More precisely, for every every valuation $\eta : \mathrm{Var} \to \mathrm{Tm}^0_\Sigma$ we construct a valuation $\eta' : \mathrm{Var} \to \boldsymbol{D}$ such that for every clause $C \in \mathrm{Cl}_\Sigma$, we have $[C]^{\mathcal{H}}_\eta = [C]^{\mathcal{M}}_{\eta'}$. Since for every clause $C \in N$, we have $[C]^{\mathcal{M}}_{\eta'} = \mathtt{true}$, this will imply that $[C]^{\mathcal{H}}_\eta = \mathtt{true}$, which proves that $\mathcal{H}$ is a model for N, since η is an arbitrary valuation.

Let the valuation η' be defined from η as follows: $\eta'(x) := [\eta(x)]^{\mathcal{M}}_*$. By induction over definition (7) it is easy to show that for every term $t \in \mathrm{Tm}_\Sigma$, we have $[t]^{\mathcal{M}}_{\eta'} = [[t]^{\mathcal{H}}_\eta]^{\mathcal{M}}_*$. Hence for every atom $A = p(t_1,..,t_m) \in \mathrm{At}_\Sigma$, according to definition (8) we have:

$$[A]^{\mathcal{M}}_{\eta'} = \mathtt{true} \quad \textit{iff} \quad ([t_1]^{\mathcal{M}}_{\eta'},..,[t_m]^{\mathcal{M}}_{\eta'}) \in p^{\mathcal{M}} \quad \textit{iff} \quad ([[t_1]^{\mathcal{H}}_\eta]^{\mathcal{M}}_*,..,[[t_m]^{\mathcal{H}}_\eta]^{\mathcal{M}}_*) \in p^{\mathcal{M}}$$
$$(\textit{by } (9)) \quad \textit{iff} \quad ([t_1]^{\mathcal{H}}_\eta,..,[t_m]^{\mathcal{H}}_\eta) \in p^{\mathcal{H}} \quad \textit{iff} \quad [A]^{\mathcal{H}}_\eta = \mathtt{true}. \quad (10)$$

Property (10) is extended using (8) for every clause $C \in \mathrm{Cl}_\Sigma$: $[C]^{\mathcal{M}}_{\eta'} = [C]^{\mathcal{H}}_\eta$, which was required to show. ⧉

There is a difficulty in extending Theorem 2.6 to the case with equality. The construction given in the proof above does not go through, since, in particular the property (10) fails for equational atoms. Sure, there might be a situation when $[s^0]^{\mathcal{M}}_\eta = [t^0]^{\mathcal{M}}_\eta$ for some *different* ground terms $s^0 \neq t^0$, so $[s^0 \simeq t^0]^{\mathcal{M}}_\eta = \mathtt{true}$, but $[s^0 \simeq t^0]^{\mathcal{H}}_{\eta'} = \mathtt{false}$, since $[s^0]^{\mathcal{H}}_{\eta'} = s^0 \neq t^0 = [t^0]^{\mathcal{H}}_{\eta'}$. The solution to this problem is to modify the notion of Herbrand interpretation by unifying several ground terms into a single domain element.

A *equivalence relation* $\sim \in \boldsymbol{D} \times \boldsymbol{D}$ is any reflexive, symmetric and transitive binary relation, i.e., satisfying the properties: (***i***) $d_1 \sim d_1$ (*reflexivity*); (***ii***) $d_1 \sim d_2$ implies $d_2 \sim d_1$ (*symmetry*) and (***iii***) ($d_1 \sim d_2$ and $d_2 \sim d_3$) implies $d_1 \sim d_3$ (*transitivity*) for every $d_i \in \boldsymbol{D}$, $i = 1, 2, 3$. An *$\sim$-equivalence class for an element* $d \in \boldsymbol{D}$ is the maximal subset $\tilde{d} \subseteq \boldsymbol{D}$ containing d such that $d_1 \sim d$ for every $d_1 \in \tilde{d}$. An equivalence class for an element is unique and can be *represented* by any of its elements: $\tilde{d}_1 = \tilde{d}$ for any $d_1 \in \tilde{d}$. We assume that there is a function assigning to every equivalence class $\tilde{d}$, one of its elements $\tilde{d}{\downarrow} \in \tilde{d}$ called the *representative* of $\tilde{d}$.

An equivalence relation on ground terms $\approx \in \mathrm{Tm}^0_\Sigma \times \mathrm{Tm}^0_\Sigma$ is called a *congruence relation* if it is *compatible w.r.t. application of functional symbols*, or, in other words, it admits the following *monotonicity axioms*: for every functional symbol $f \in \mathrm{Fun}$, and ground terms $s^0_i, t^0_i \in \mathrm{Tm}^0_\Sigma$, $1 \leq i \leq n = ar(f)$, we have $s^0_1 \approx t^0_1, \ldots, s^0_n \approx t^0_n$ implies that $f(s^0_1,..,s^0_n) \approx f(t^0_1,..,t^0_n)$. An equivalence class w.r.t. to a congruence relation is called a *congruence class*. Now we are ready to define a notion of Herbrand interpretation for the clause logic with equality.

Definition 2.7. An *(equality) Herbrand interpretation* is an interpretation $\mathcal{H} = (\widetilde{\mathrm{Tm}^0_\Sigma}, \cdot^{\mathcal{H}})$, where (***i***) $\widetilde{\mathrm{Tm}^0_\Sigma}$ is the set of all congruence classes of ground terms w.r.t.

some congruence relation $\approx \subseteq \mathrm{Tm}^0_\Sigma \times \mathrm{Tm}^0_\Sigma$: $\widetilde{\mathrm{Tm}}^0_\Sigma = \{\tilde{t}^0 \mid t^0 \in \mathrm{Tm}^0_\Sigma\}$ and (***ii***) $f^{\mathcal{H}}(\tilde{t}^0_1,..,\tilde{t}^0_n) = \tilde{t}^0 \in \widetilde{\mathrm{Tm}}^0_\Sigma$ where $t^0 = f(t^0_1,..,t^0_n) \in \mathrm{Tm}^0_\Sigma$. ⋘

We need to show that this definition is correct, i.e., the interpretation $\mathcal{H}$ does not depend on the choice of representatives for equivalence classes in case (***ii***):

Proposition 2.8 (Correctness of Definition 2.7). *Let* $s^0 = f(s^0_1,..,s^0_n)$, $t^0 = f(t^0_1,..,t^0_n)$ *and* $\tilde{s^0_i} = \tilde{t^0_i}$ *for all* i *with* $1 \le i \le n$. *Then* $\tilde{s^0} = \tilde{t^0}$.

Proof. For every i with $1 \le i \le n$, we have that $\tilde{s^0_i} = \tilde{t^0_i}$ implies $s^0_i \approx t^0_i$. So, by monotonicity axiom (since $\approx$ is a congruence relation) we obtain $s^0 \approx t^0$ which implies $\tilde{s^0} = \tilde{t^0}$. ⧉

Note that the notion of Herbrand interpretation for signatures without equality is subsumed by Definition 2.7, when $\approx$ is the *identity* congruence relation "=" (that is, the *syntactic equality*). Now the Herbrand Theorem can be also proven for the case with equality:

Proof of Theorem 2.6 for clause logic with equality. We modify the construction of Herbrand interpretation as follows. Given a model $\mathcal{M} = (\boldsymbol{D}, \cdot^{\mathcal{M}})$ for N, we define a congruence relation $\approx \subseteq \mathrm{Tm}^0_\Sigma \times \mathrm{Tm}^0_\Sigma$ by setting $s^0 \approx t^0$ *iff* $[s^0]^{\mathcal{M}}_* = [t^0]^{\mathcal{M}}_*$ (again, the choice of a variable valuation $*$ is irrelevant since the terms s^0 and t^0 are ground). Obviously $\approx$ is an equivalence relation, and by (8) it fulfills monotonicity axioms, so $\approx$ is a congruence relation. The rest of Herbrand interpretation, i.e., the interpretation of non-equality predicate symbols can be defined analogously to (9). One should simply replace ground terms with their equivalence classes:

$$(\tilde{t^0_1},..,\tilde{t^0_m}) \in p^{\mathcal{H}} \quad \textit{iff} \quad ([\tilde{t^0_1}\downarrow]^{\mathcal{M}}_*,..,[\tilde{t^0_m}\downarrow]^{\mathcal{M}}_*) \in p^{\mathcal{M}} \tag{11}$$

This definition is correct, i.e., it does not matter which representatives $\tilde{t^0_i}\downarrow$ for equivalence classes $\tilde{t^0_i}$ we choose for $1 \le i \le m$, since $\tilde{s^0_i} = \tilde{t^0_i}$ implies $s_i \approx t_i$, and so $[s^0_i]^{\mathcal{M}}_* = [t^0_i]^{\mathcal{M}}_*$ according to the definition of $\approx$. The rest of the proof goes without considerable modifications:

Given a valuation $\eta : \mathrm{Var} \to \widetilde{\mathrm{Tm}}^0_\Sigma$, we define a new valuation $\eta' : \mathrm{Var} \to \boldsymbol{D}$ by $\eta'(x) := [\eta(x)\downarrow]^{\mathcal{M}}_*$ (which is again a correct definition). By induction over the definition (7) one easily extends this property to arbitrary terms: $[t]^{\mathcal{M}}_{\eta'} = [[t]^{\mathcal{H}}_{\eta}\downarrow]^{\mathcal{M}}_*$, $t \in \mathrm{Tm}_\Sigma$. For non-equational atoms $A = p(t_1,..,t_m)$ this property together with (11) implies the following analog of (10):

$$\begin{aligned}[A]^{\mathcal{M}}_{\eta'} = \mathtt{true} \quad \textit{iff} \quad ([t_1]^{\mathcal{M}}_{\eta'},..,[t_m]^{\mathcal{M}}_{\eta'}) \in p^{\mathcal{M}} \quad \textit{iff} \quad ([[t_1]^{\mathcal{H}}_{\eta}\downarrow]^{\mathcal{M}}_*,..,[[t_m]^{\mathcal{H}}_{\eta}\downarrow]^{\mathcal{M}}_*) \in p^{\mathcal{M}} \\ (\textit{by (11)}) \quad \textit{iff} \quad ([t_1]^{\mathcal{H}}_{\eta},..,[t_m]^{\mathcal{H}}_{\eta}) \in p^{\mathcal{H}} \quad \textit{iff} \quad [A]^{\mathcal{H}}_{\eta} = \mathtt{true}. \end{aligned} \tag{12}$$

In addition, for equational atoms $A = t \simeq s$ we have:

$$\begin{array}{l}[A]^{\mathcal{M}}_{\eta'} = \mathtt{true} \quad \textit{iff} \quad [s]^{\mathcal{M}}_{\eta'} = [t]^{\mathcal{M}}_{\eta'} \quad \textit{iff} \quad [[s]^{\mathcal{H}}_{\eta}\downarrow]^{\mathcal{M}}_{*} = [[t]^{\mathcal{H}}_{\eta}\downarrow]^{\mathcal{M}}_{*} \\ (\textit{by definition of } \approx) \quad \textit{iff} \quad [s]^{\mathcal{H}}_{\eta}\downarrow \approx [t]^{\mathcal{H}}_{\eta}\downarrow \quad \textit{iff} \quad [s]^{\mathcal{H}}_{\eta} = [t]^{\mathcal{H}}_{\eta} \quad \textit{iff} \quad [A]^{\mathcal{H}}_{\eta} = \mathtt{true}.\end{array} \tag{13}$$

The property $[A]^{\mathcal{M}}_{\eta'} = [A]^{\mathcal{H}}_{\eta}$ proven for all atoms, can be extended using (8) to all clauses $C \in \mathrm{Cl}_\Sigma$: $[C]^{\mathcal{M}}_{\eta'} = [C]^{\mathcal{H}}_{\eta}$. Hence, for every $C \in N$ we have $[C]^{\mathcal{H}}_{\eta} = [C]^{\mathcal{M}}_{\eta'} = \mathtt{true}$. ⧉

We are now concerned with the question of how to represent equality Herbrand interpretations. It is easy to modify the representation described in Remark 2.5 to take into account congruence relations. We extend any congruence relation $\approx$ on ground terms to ground non-equational atoms as follows: $p(s^0_1,..,s^0_m) \approx p'(t^0_1,..,t^0_{m'})$ *iff* (***i***) $p = p'$ (and consequently $m = m'$), and (***ii***) $s^0_i \approx t^0_i$ for all i with $1 \le i \le m$. Let $\widetilde{\mathrm{At}}^0_\Sigma$ be the set of equivalence classes of At^0_Σ modulo $\approx$.

Remark 2.9. An equality Herbrand interpretation can be uniquely represented by a pair $(\approx, \tilde{I})$, where $\approx$ is a congruence relation on Tm^0_Σ and $\tilde{I} \subseteq \widetilde{\mathrm{At}}^0_\Sigma$. Indeed, the only parameter of an equality Herbrand interpretation $\mathcal{H}$ that is not fixed, is the interpretation of non-equational predicate symbols, which can be represented using $\tilde{I}$ as follows: $(\tilde{t^0_1},..,\tilde{t^0_m}) \in p^{\mathcal{H}}$ *iff* $\tilde{A} \in \tilde{I}$ for $A := p(t^0_1,..,t^0_m)$. This definition can be easily shown to be correct (i.e., does not depend on the choice of representatives for equivalence classes). ⟐

Now the question is, how to represent a congruence relation and equivalence classes induced by it? It seems to be not very efficient to enumerate all equivalent pairs of ground terms to store a congruence relation, since many of these pairs can be "derived" using the *congruence axioms*: reflexivity, symmetry, transitivity and monotonicity. In subsection 2.3 we demonstrate how congruence relations and congruence classes can be efficiently represented using so-called rewrite rules.

In the remaining part of this section we introduce additional terminology that helps to characterize different types of clauses in saturation-based decision procedures. By an *expression* E we mean a term or a literal. An *expression symbol* e is either a functional symbol f or a predicate symbol p or a *negated* predicate symbol $\neg p$. In the last two cases we deal with a *literal symbol* l. The *arity* $ar(e)$ *(*$ar(l)$*) of an expression symbol* e *(or a literal symbol* l*)* is the arity of the predicate or the functional symbol it is produced from. Sometimes we will form expressions by attaching a *sequence of arguments* $(t_1,..,t_n)$ to an expression symbol e: $E = e(t_1,..,t_n)$, where $t_i \in \mathrm{Tm}_\Sigma$, $1 \le i \le n = ar(e)$. In this case we say also that $(t_1,..,t_n)$ are the *arguments of* E.

The *size* $|E|$, $|C|$ *of an expression* E *or a clause* C is determined by treating them as appropriate terms or formulas. The *depth* $depth(E)$ and the *variable depth* $vardepth(E)$ of an expression E are determined as follows:

$$\begin{array}{rcl} depth(E) := \quad depth(x) & = & 1 \\ depth(e(t_1,..,t_n)) & = & 1 + \max\{0, depth(t_1),.., depth(t_n)\} \ . \end{array}$$

$$\begin{array}{rcl} vardepth(E) := vardepth(x) & = & 1 \\ vardepth(E^0) & = & 0 \quad \textit{if } E^0 \textit{ is ground} \\ vardepth(e(t_1,..,t_n)) & = & 1 + \max\{0, vardepth(t_1),.., vardepth(t_n)\} \\ & & \textit{if some } t_i \textit{ with } 1 \leq i \leq n \textit{ is not ground} \ . \end{array} \tag{14}$$

An expression E is *shallow*, if $depth(E) \leq 2$, i.e., all arguments of the expression are variables or constants. A literal L is *simple* if $depth(L) \leq 3$, i.e., all its arguments are shallow. An expression E or a clause C is *functional* if it contains at least one functional symbol.

2.3 Term Rewrite Systems

Term rewriting is typically used to model changes in dynamical systems (for example, for describing a model of computation for a programming language). In the context of saturation-based theorem proving, rewrite systems often represent a static information (models). The purpose of this section is to demonstrate the usage of rewriting techniques for representing congruence relations and congruence classes. A more detailed account of the material in this and the subsequent sections can be found in [Dershowitz, 1987; Baader & Nipkow, 1998; Dershowitz & Plaisted, 2001].

A *rewrite relation* $\Rightarrow$ is a monotone relation on ground terms: $\Rightarrow \subseteq \mathrm{Tm}^0_\Sigma \times \mathrm{Tm}^0_\Sigma$, i.e., satisfying the property: $h^0[s^0] \Rightarrow h^0[t^0]$ if $s^0 \Rightarrow t^0$ (*monotonicity*). By $\stackrel{*}{\Rightarrow}$ we denote transitive reflexive closure of $\Rightarrow$, and by $\stackrel{*}{\Leftrightarrow}$ we denote the *equivalence closure* of $\Rightarrow$ (i.e., $\stackrel{*}{\Leftrightarrow} := (\Rightarrow \cup \Leftarrow)^*$ is the minimal equivalence relation containing $\Rightarrow$). Note that $\stackrel{*}{\Leftrightarrow}$ is a congruence relation because the relation $\Rightarrow$, and consequently $\stackrel{*}{\Leftrightarrow}$ are monotone.

A *(ground) rewrite system* R is a set of *(ground) rewrite rules* of the form $s^0 \Rightarrow t^0$, where $s^0, t^0 \in \mathrm{Tm}^0_\Sigma$. The term s^0 is called a *redex* of the rewrite rule $s^0 \Rightarrow t^0$. A rewrite relation $\Rightarrow_R$ *induced by* a ground rewrite system R is the smallest rewrite relation containing all rules from R. A ground term $s^0 \in \mathrm{Tm}^0_\Sigma$ is *irreducible (w.r.t. R)* if $s^0 \Rightarrow_R t^0$ for no term $t^0 \in \mathrm{Tm}^0_\Sigma$. A term t^0 is a *normal form* of s^0 (w.r.t. R) if $s^0 \stackrel{*}{\Rightarrow}_R t^0$ and t^0 is irreducible. We denote by $t^0 \Downarrow_R$ *some* normal form of a term

t^0 (in case it exists). A rewrite system R is *terminating* or *well-founded* if there is no infinite sequence of terms $s_1^0 \Rightarrow_R s_2^0 \Rightarrow_R \cdots \Rightarrow_R s_i^0 \Rightarrow_R \cdots$.

Termination of a rewrite system guarantees the existence of a normal form for every ground term, but not its uniqueness. Normal forms are unique if a rewrite system is *confluent* (also called *Church-Rosser*): whenever $s^0 \stackrel{*}{\Rightarrow}_R t_1^0$ and $s^0 \stackrel{*}{\Rightarrow}_R t_2^0$, then the terms t_1^0 and t_2^0 are *R-joinable*, i.e., there exists $h^0 \in \mathrm{Tm}_\Sigma^0$ such that $t_1^0 \stackrel{*}{\Rightarrow}_R h^0$ and $t_2^0 \stackrel{*}{\Rightarrow}_R h^0$. In this case we will also write $t_1^0 \Downarrow_R t_2^0$ and say that the equation $t_1^0 \simeq t_2^0$ *converges*, or *has a rewrite proof* in R. Similarly, confluence alone does not suffice for existence of normal forms, but for their uniqueness: Indeed, two different joinable terms cannot be both irreducible, so every ground term has at most one R-normal form.

A terminating, confluent rewrite system is called *convergent*. It can be shown that for convergent rewrite systems, $s^0 \stackrel{*}{\Leftrightarrow}_R t^0$ implies $s^0\Downarrow_R = t^0\Downarrow_R$. Thus, one could effectively decide the equivalence problem $s^0 \stackrel{*}{\Leftrightarrow}_R^? t^0$ for any ground terms s^0 and t^0: indeed, this amounts in checking whether s^0 and t^0 are R-joinable: $s^0 \Downarrow_R^? t^0$, which can be done by computing and comparing their normal forms $s^0\Downarrow_R$ and $t^0\Downarrow_R$ by applying finitely many R-rewrite steps. These results can be summarized in the following Proposition:

Proposition 2.10. *Let R be a convergent rewrite system for ground terms Tm_Σ^0. Then for every $s^0, t^0 \in \mathrm{Tm}_\Sigma^0$ (**i**) there exist unique normal forms $s^0\Downarrow_R$ and $t^0\Downarrow_R$ w.r.t. R and (**ii**) $s^0 \stackrel{*}{\Leftrightarrow}_R t^0$ iff $s^0\Downarrow_R = s^0\Downarrow_R$.*

The algorithm for checking equivalence of ground terms given above, can be used to represent the *congruence closure* $\approx_E$ induced by a finite set E of ground equations of form $s^0 \simeq t^0$ (i.e., $\approx_E$ is the least congruence relation containing all equations from E). For this purpose, one should find a convergent rewrite system R, such that $\stackrel{*}{\Leftrightarrow}_R$ coincides with $\approx_E$. The last property can be achieved by taking a rewrite system which consists of the *oriented equations* from E: $s \approx_E t$ orients to $s^0 \Rightarrow t^0$. However, the resulted rewrite system may not be convergent.

Knuth & Bendix [1970] formulated a necessary and sufficient condition for a terminating rewrite system to be convergent and presented a so-called completion procedure which computes a convergent rewrite system for a set of ground equations.

We say that redexes s_1^0, s_2^0 of two rewrite rules $s_1^0 \Rightarrow t_1^0$ and $s_2^0 \Rightarrow t_2^0$ are *overlapping* if $s_1^0 \trianglelefteq s_2^0$ or $s_2^0 \trianglelefteq s_1^0$, or in words, one redex is a subterm of the other redex. We break the symmetry in this definition by assuming that $s_1^0 \trianglelefteq s_2^0$ (w.l.o.g.), i.e., $s_2^0 = s_2^0[s_1^0]$. In this situation, the equation $s_2^0[t_1^0] \simeq t_2^0$ is called the *critical pair* between these two rewrite rules and the term $s_2^0[s_1^0]$ is called the *overlapped term*. The critical pair between rules of a confluent rewrite system R has a rewrite proof in R, since the overlapped term is reducible to both terms in the critical pair:

$s_2^0[s_1^0] \Rightarrow s_2^0[t_1^0]$ and $s_2^0[s_1^0] \Rightarrow t_2^0$. The converse does not necessary hold for all rewrite systems, but for *terminating* rewrite systems this can be shown:

Lemma 2.11 (Critical Pair Lemma, Knuth & Bendix [1970]). *A terminating rewrite system R is confluent iff every critical pair between the rules in R converges.*

Lemma 2.11 provides the basis for the following congruence closure procedure that is called the *(ground) Knuth-Bendix completion*. Starting with a finite set of ground equations E, we want to compute a convergent rewrite system R such that $\approx_E$ is equal to $\overset{*}{\Leftrightarrow}_R$. This can be done by applying the inference rules from System 1.

Orient

$$\mathtt{O}: \frac{s^0 \simeq t^0}{s^0\Downarrow_R \Rightarrow t^0\Downarrow_R} \quad \left[\ \textit{if} \quad s^0\Downarrow_R \neq t^0\Downarrow_R.\ \right]$$

Superpose

$$\mathtt{S}: \frac{s^0 \Rightarrow t^0 \quad w^0[s^0] \Rightarrow v^0}{w^0[t^0] \simeq v^0}$$

System 1: Ground Knuth-Bendix completion $\mathcal{KB}^0$

System 1 describing the Knuth-Bendix completion procedure is a simple example of *inference systems* we will deal with in this report. The procedure is given in form of a *calculus* $\mathcal{KB}^0$ that consists of two inference rules: Orient and Superpose. The *short versions* for the names of the inference rules `O` and `S` will be used in applications of these rules. The *premises* of the rules are drawn above the separation line and the *conclusions* of the rules are drawn below. The first rule has one premise and the second rule is applied to two premises. The prerequisites for applications of rules called the *conditions* of inference rules are written below each rule (if there are any).

The rules can be applied in *don't care* nondeterministic fashion, i.e., any plausible inference rule can be executed at every moment of time. However, we generally assume that application of the rules is (***i***) *sequential*, i.e., several rules cannot be applied simultaneously, and (***ii***) *fair*, i.e., no inference can be postponed infinitely long. We shall see another type of nondeterminism, called *don't know* nondeterminism, when we consider nondeterministic inference rules.

Starting with a finite set of equations E we apply inference rules from System 1 which produce rewrite rules of R and new equations. The condition of the first inference rule Orient is verified w.r.t. the *current* rewrite system R. Note, that this rule can be applied to an equation $s^0 \simeq t^0$ in two different ways, since equations are treated symmetrically. One application introduces the rewrite rule $s^0 \Rightarrow t^0$, and the other orients the equation to $t^0 \Rightarrow s^0$. Note that after an application of Orient, its

premise converges, so the rule becomes inapplicable the second time. The second inference rule Superpose is applied to overlapping rewrite rules and produces the critical pair of these rewrite rules. After Orient is applied to this equation, the critical pair of these rewrite rules converges.

If the inference procedure terminates, i.e., if no new equation or a rewrite rule can be produced, then we have computed a set of rewrite rules R for E. Speaking in general terms, we say that in this case, the set of equations and rewrite rules is *saturated* w.r.t. $\mathcal{KB}^0$ and that we have computed a *saturation* for E. There are several possible rewrite systems that can be computed for the same input E, because of nondeterministic nature of the inference process. If a computed rewrite system R is terminating, then by Lemma 2.11 it is convergent, since every critical pair is derived by Superpose and converges by Orient. In this case we have computed R such that $\overset{*}{\Leftrightarrow}_R$ coincides with $\approx_E$. Unfortunately, there are two assumptions in this proposition that do not necessary hold for arbitrary derivations: (***i***) termination of the inference procedure and (***ii***) termination of the computed rewrite system. We demonstrate this in the following example:

Example 2.12. Let us compute a rewrite system for the set E consisting of equations 1. $\mathtt{f(a)} \simeq \mathtt{a}$ and 2. $\mathtt{g(f(a))} \simeq \mathtt{f(g(a))}$. The inference rules from $\mathcal{KB}^0$ can be applied in several different ways:

Saturation A:	Saturation B:	Saturation C:
$\mathtt{O}[1]$: 3. $\mathtt{a} \Rightarrow \mathtt{f(a)}$	$\mathtt{O}[1]$: 3. $\underline{\mathtt{f(a)}} \Rightarrow \mathtt{a}$	$\mathtt{O}[1]$: 3. $\mathtt{f(a)} \Rightarrow \mathtt{a}$
$\mathtt{O}[2]$: 4. $-\,n.t.-$	$\mathtt{O}[2]$: 4. $\mathtt{g}(\underline{\mathtt{f(a)}}) \Rightarrow \mathtt{f(g(a))}$	$\mathtt{O}[2]$: 4. $\mathtt{f(g(a))} \Rightarrow \mathtt{g(f(a))}$
	$\mathtt{S}[3;4]$: 5. $\mathtt{g(a)} \simeq \mathtt{f(g(a))}$	
	$\mathtt{O}[5]$: 6. $\mathtt{g(a)} \Rightarrow \mathtt{f(g(a))}$	

Saturation A is an example of non-termination of the procedure. Here we have oriented the first equation $\mathtt{f(a)} \simeq \mathtt{a}$ into the rewrite rule $\mathtt{a} \Rightarrow \mathtt{f(a)}$. This has been a wrong decision, since now we cannot execute the rule Orient for the second equations, since the normalization procedure for the terms $\mathtt{g(f(a))}$ and $\mathtt{f(g(a))}$ in the condition of the rule does not terminate.

In Saturation B we have oriented the first equation in a right way: $\mathtt{f(a)} \Rightarrow \mathtt{a}$ and have successfully computed all inferences (we have underlined the matched terms). Although the computed rewrite system is confluent, it is not terminating because of the last rewrite rule: 6. $\mathtt{g(a)} \Rightarrow \mathtt{f(g(a))}$. If we had oriented equation 5 to the opposite direction, we would have obtained a convergent rewrite system.

In Saturation C we obtained a terminating rewrite system immediately, by orienting the initial equations in such a way that no critical pair is produced. $\lll$

Example 2.12 shows that orienting the equations in a right way is crucial for termination of Knuth-Bendix completion and for producing a terminating rewrite

system. It seems that the right way to orient equations is to make rewriting from "larger" to "smaller" terms. In the next section we will see how to force the completion procedure terminate for ground equations using certain orderings.

2.4 Orderings

Ordering restrictions were shown to be useful not only in the context of term rewriting, but also as refinements in many saturation-based theorem proving procedures.

A *(strict partial) ordering (or order)* $\succ$ on a set $\boldsymbol{D}$ is a transitive and irreflexive binary relation on $\boldsymbol{D}$. If $\succ$ is a strict ordering then its reflexive closure is denoted by $\succeq$.

A *quasi-ordering* $\succsim$ on $\boldsymbol{D}$ is any reflexive and transitive relation on $\boldsymbol{D}$. An *equivalence relation induced* by a quasi-ordering $\succsim$ is the *symmetrical part* $\sim$ of $\succsim$: $d_1 \sim d_2$ *iff* $d_1 \succsim d_2$ and $d_2 \succsim d_2$. A *strict part* $\succ$ of a quasi-ordering $\succsim$ is the difference between $\succsim$ and $\sim$: $d_1 \succ d_2$ *iff* $d_1 \succsim d_2$ and $d_2 \not\succsim d_1$. Note that the strict part $\succ$ of $\succsim$ is the greatest ordering contained in $\succsim$.

An ordering $\succ$ is *total* or *linear* if every two different elements are *comparable* by $\succ$, i.e., for every $d_1, d_2 \in \boldsymbol{D}$, $d_1 \neq d_2$ implies that either $d_1 \succ d_2$ or $d_2 \succ d_1$. An ordering $\succ$ is *well-founded* or *Noetherian* if there is no infinite descending chain $d_1 \succ d_2 \succ \cdots$ of elements $d_i \in \boldsymbol{D}$, $i \geq 1$. A total well-founded order is called a *well-order*.

2.4.1 Multiset and lexicographic extensions of orderings

A *multiset* of elements from $\boldsymbol{D}$ is a function $M : \boldsymbol{D} \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. The number $M(d)$ is called the *multiplicity* of an element d in M, $d \in \boldsymbol{D}$. We say that an element d *belongs* to a multiset M (in symbols $d \in M$) if $M(d) > 0$. The *size* $|M|$ of a multiset M is defined by $|M| := \sum_{d \in D} M(d)$. A multiset M is *finite* if $|M| < \infty$.

Note that every subset D of $\boldsymbol{D}$ is uniquely represented by a multiset M_D that is the *characteristic function of* D: $M_D(d) = 1$ *iff* $d \in D$. Conversely a multiset can be seen as a set in which several occurrences of the same element are allowed. Given this correspondence, it is not difficult to extend some set-theoretic operations to multisets: the *multiset union* $M_1 \cup M_2$ of two multisets M_1 and M_2 is defined by $(M_1 \cup M_2)(d) := M_1(d) + M_2(d)$; the *multiset intersection* $M_1 \cap M_2$ is given by $(M_1 \cap M_2)(d) := \min(M_1(d), M_2(d))$ and the *multiset difference* $M_1 \setminus M_2$ is defined by $(M_1 \setminus M_2)(d) = \max(0, M_1(d) - M_2(d))$ for all $d \in \boldsymbol{D}$. We say that M_1 is a *submultiset* of M_2 (notation $M_1 \subseteq M_2$), if $M_1 \setminus M_2 = \{\}_m$, where the last is the *empty multiset*, that is given by $\{\}_m(d) = 0$ for every $d \in \boldsymbol{D}$.

Let $\succ$ be an ordering on $\boldsymbol{D}$ and D be a multiset of elements from $\boldsymbol{D}$. An element $d \in \boldsymbol{D}$ is called *maximal (strictly maximal) w.r.t. D* if $d' \succ d$ ($d' \succeq d$) for no $d' \in D$. An element $d \in \boldsymbol{D}$ is *greatest (strictly greatest) w.r.t. D* if $d \succeq d'$ ($d \succ d'$) for all elements $d' \in D$. An element $d \in D$ is *(strictly) maximal or (strictly) greatest in D*, if it is respectively (strictly) maximal or (strictly) greatest w.r.t. $D \setminus \{d\}$. The notions of *(strictly) minimal* and *(strictly) least* elements w.r.t. D (in D) are defined analogously by inverting respective relations ($\succ$ to $\prec$ and $\succeq$ to $\preceq$).

Any ordering $\succ$ on $\boldsymbol{D}$ can be extended to an ordering $\succ_{mul}$ on finite multisets of $\boldsymbol{D}$ as follows: $M_1 \succ_{mul} M_2$ *iff* (***i***) $M_1 \neq M_2$ and (***ii***) for every element $d \in \boldsymbol{D}$, either $M_1(d) \geq M_2(d)$, or, otherwise there exists for some $d' \succ d$, $d' \in \boldsymbol{D}$, such that $M_1(d') > M_2(d')$. The ordering $\succ_{mul}$ is called the *multiset extension of* of the ordering $\succ$.

Example 2.13. Let $\boldsymbol{D} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $\mathtt{M}_1 = \{\mathtt{a}, \mathtt{b}\}_m$, $\mathtt{M}_2 = \{\mathtt{a}, \mathtt{c}, \mathtt{c}\}_m$, $\mathtt{M}_3 = \{\mathtt{b}, \mathtt{b}, \mathtt{b}\}_m$ be multisets over $\boldsymbol{D}$. Formally, this notation means that $\mathtt{M}_1(\mathtt{a}) = 1$, $\mathtt{M}_1(\mathtt{b}) = 1$, $\mathtt{M}_1(\mathtt{c}) = 0$; $\mathtt{M}_2(\mathtt{a}) = 1$, $\mathtt{M}_2(\mathtt{b}) = 0$, $\mathtt{M}_2(\mathtt{c}) = 2$ and $\mathtt{M}_3(\mathtt{a}) = 0$, $\mathtt{M}_3(\mathtt{b}) = 3$, $\mathtt{M}_3(\mathtt{c}) = 0$. Then $\mathtt{M}_1 \cup \mathtt{M}_2 = \{\mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{c}\}_m$, $\mathtt{M}_1 \cap \mathtt{M}_3 = \{\mathtt{b}\}_m$, $\mathtt{M}_2 \setminus \mathtt{M}_1 = \{\mathtt{c}, \mathtt{c}\}_m$. Let $\succ$ be an ordering on $\boldsymbol{D}$ such that $\mathtt{a} \succ \mathtt{b} \succ \mathtt{c}$ and $\succ_{mul}$ be the multiset extension of $\succ$. Then $\mathtt{M}_1 \succ_{mul} \mathtt{M}_2 \succ_{mul} \mathtt{M}_3$. ⋘

The following proposition states that the multiset extension of an ordering inherits many properties of the ordering it is based on:

Proposition 2.14. *Let $\succ$ be a strict partial ordering on $\boldsymbol{D}$. Then its multiset extension $\succ_{mul}$ is a strict partial ordering and:*
(***i***) *$\succ_{mul}$ is total iff $\succ$ is total and*
(***ii***) *$\succ_{mul}$ is well-founded iff $\succ$ is well-founded.*

Any ordering $\succ$ on $\boldsymbol{D}$ can be extended to an ordering $\succ^n_{lex}$ on $\boldsymbol{D}^n$ called the *lexicographic extension of* of $\succ$ as follows: $(d_1,.., d_n) \succ^n_{lex} (d'_1,.., d'_n)$ *iff* there exists i with $1 \leq i \leq n$ such that $d_i \succ d'_i$, and for all j with $1 \leq j < i$, we have $d_j = d'_j$. Continuing Example 2.13, it is easy to show by this definition that $(\mathtt{a}, \mathtt{b}, \mathtt{c}) \succ^3_{lex} (\mathtt{b}, \mathtt{c}, \mathtt{c}) \succ^3_{lex} (\mathtt{c}, \mathtt{b}, \mathtt{a}) \succ^3_{lex} (\mathtt{c}, \mathtt{b}, \mathtt{b}) \succ^3_{lex} (\mathtt{c}, \mathtt{c}, \mathtt{a})$. Often we omit the index n in $\succ^n_{lex}$, when the number of elements in vectors is clear from the context.

The idea of the lexicographic extension of an ordering can be further extended to combine several orderings. Let $(\boldsymbol{D}_1, \succ_1)$, $(\boldsymbol{D}_2, \succ_2)$,... $(\boldsymbol{D}_n, \succ_n)$ be ordered sets. We can define a *lexicographic combination* of orderings $\succ_1, \succ_2,.., \succ_n$ as a binary relation $\succ^{1..n}_{lex}$ (also denoted by $(\succ_1,.., \succ_n)$) on $\boldsymbol{D}_1 \times \cdots \times \boldsymbol{D}_n$ satisfying: $(d_1, d_2,.., d_n) \succ^{1..n}_{lex} (d'_1, d'_2,.., d'_n)$ *iff* there exists i with $1 \leq i \leq n$ such that $d_i \succ_i d'_i$, and for all j with $1 \leq j < i$, we have $d_j = d'_j$. Note that the lexicographic extension $\succ^n_{lex}$ of an ordering $\succ$ is a lexicographic combinations of n copies of $\succ$.

The following is a variant of Proposition 2.14 for lexicographic combinations of orderings:

Proposition 2.15. *Let $\succ_i$ be strict partial orderings on $\boldsymbol{D}_i$ for i with $1 \leq i \leq n$ and let $\succ_{lex}^{1..n}$ be its lexicographic combination. Then $\succ_{lex}^{1..n}$ is a strict partial ordering and:*

(***i***) *$\succ_{lex}^{1..n}$ is total iff every $\succ_i$ with $1 \leq i \leq n$ is total and*

(***ii***) *$\succ_{lex}^{1..n}$ is well-founded iff every $\succ_i$ with $1 \leq i \leq n$ is well-founded.*

2.4.2 Reduction orders

In automated deduction, orders are used to constrain inferences and are applied to syntactical objects of clause logic: terms, literals and clauses. For these domains, additional classification of orders is possible.

Let $\succ$ be an ordering on ground expressions over a signature Σ (i.e., on ground terms and ground literals). We say that $\succ$ is a *rewrite ordering* if $\succ$ is a rewrite relation, i.e., it admits the monotonicity property: for every ground terms s^0, t^0 and a ground expression E^0 with $s^0 \lhd E^0$, $s^0 \succ t^0$ implies $E^0[s^0] \succ E^0[t^0/s^0]$. A *reduction ordering* is a well-founded rewrite ordering. An ordering $\succ$ has the *subterm property* if $E^0 \succ t^0$ for every $t^0 \lhd E^0$. A *simplification ordering* is any reduction ordering with the subterm property.

It is easy to show that every *total* reduction ordering must have the subterm property for ground *terms*. Indeed, otherwise $s^0 \succ t^0[s^0]$ for some terms $s^0 \lhd t^0$ and by monotonicity we obtain an infinite descending chain: $s^0 \succ t^0[s^0] \succ t^0[t^0[s^0]] \succ \cdots$. It is not, however, true that a total reduction ordering must have the subterm property: $L^0[t^0] \succ t^0$ for a subterm t^0 of a *literal* L^0, because literals cannot strictly occur in other expressions. In the remaining part of this section we restrict ourselves to *ground term orderings*, i.e., orderings over the domain $\boldsymbol{D} = \mathrm{Tm}^0_\Sigma$.

Most term orderings used in applications nowadays are variations of either Knuth-Bendix ordering [Knuth & Bendix, 1970] or a lexicographic path ordering [Kamin & Lévy, 1980]. Both orderings are based on a *precedence* $\gg$, which is a strict order on functional symbols Fun of a signature Σ.

The Knuth-Bendix ordering A *weight function* is any function $\mathit{weight} : \mathrm{Fun} \to \mathbb{N}$ that assigns a non-negative integer[2] to every functional symbol from Fun. A weight function $\mathit{weight}(\cdot)$ is *admissible* for a precedence $\gg$ *iff* (***i***) $\mathit{weight}(c) > 0$ for every constant c and (***ii***) for every *unary* functional symbol $f \in \mathrm{Fun}$, $\mathit{weight}(f) = 0$ implies that f is $\gg$-greatest element in Fun (i.e., for every for every $g \in \mathrm{Fun} \setminus \{f\}$, we have $f \gg g$). The weight function is recursively extended to the set of ground

[2]some definitions, e.g., in [Baader & Nipkow, 1998], allow for non-negative real weights, however the advantage of this is not very clear

terms Tm^0_Σ as follows: $\mathit{weight}(f(t^0_1,..,t^0_n)) := \mathit{weight}(f)+\mathit{weight}(t^0_1)+\cdot+\mathit{weight}(t^0_n)$. Note, that if $\mathit{weight}(f) = 1$ for every functional symbol $f \in \mathrm{Fun}$, then $\mathit{weight}(t) = |t|$, where $|t|$ is the size of t.

Definition 2.16. The *Knuth-Bendix ordering* (short *KBO*), induced by a precedence $\gg$ and an admissible weight function $\mathit{weight}(\cdot)$ is defined as follows: For every pair of ground terms $s^0 = f(s^0_1,..,s^0_n)$ and $t^0 = g(t^0_1,..,t^0_m)$ we have $s^0 \succ_{kbo} t^0$ *iff* one of the following conditions holds:

(**1**) $\mathit{weight}(s^0) > \mathit{weight}(t^0)$, or
(**2**) $\mathit{weight}(s^0) = \mathit{weight}(t^0)$, but $f \gg g$, or
(**3**) $\mathit{weight}(s^0) = \mathit{weight}(t^0)$, $f = g$ (and hence $m = n$), and $(s^0_1,..,s^0_n) \succ^{kbo}_{lex} (t^0_1,..,t^0_n)$,

where $\succ^{kbo}_{lex}$ *is the lexicographic extension of* $\succ_{kbo}$. ⋘

Note that the Knuth-Bendix ordering is, in some sense, a recursive lexicographic combination of (***i***) the ordering $>$ on integers, (***ii***) the ordering $\gg$ on functional symbols and (***iii***) the lexicographic extension $\succ_{lex}$ of the ordering itself. It can be shown that for any precedence and admissible weight function, Knuth-Bendix ordering is a simplification order:

Theorem 2.17 ([see Baader & Nipkow, 1998, Theorem 5.4.20]). *Let* $\gg$ *be a precedence on functional symbols* Fun *of a signature* Σ *and* $\mathit{weight}(\cdot)$ *be an admissible weight function for* $\gg$. *Then the Knuth-Bendix order* $\succ_{kbo}$ *induced by* $\gg$ *and* $\mathit{weight}(\cdot)$ *is a simplification ordering.*

Note, that if the precedence $\gg$ on functional symbols is total, then $\succ_{kbo}$ based on this precedence is total as well. Indeed, by Definition 2.16, if $s^0 \not\succ_{kbo} t^0$ and $t^0 \not\succ_{kbo} s^0$ for some ground terms $s^0 = f(s^0_1,..,s^0_n)$ and $t^0 = g(t^0_1,..,t^0_m)$, then (**1**) $\mathit{weight}(s^0) = \mathit{weight}(t^0)$; (**2**) $f = g$ and (**3**) $(s^0_1,..,s^0_n) = (t^0_1,..,t^0_n)$ (by induction hypothesis, since the lexicographic extension of a total order is a total order by Proposition 2.15).

Proposition 2.18. *Let* $\succ_{kbo}$ *be the Knuth-Bendix order induced by a total precedence* $\gg$ *and a weight function* $\mathit{weight}(\cdot)$. *Then* $\succ_{kbo}$ *is a total ordering.*

The lexicographic path ordering A reduction ordering can be defined based on a precedence of functional symbols only:

Definition 2.19. The *lexicographic path ordering* (short *LPO*), induced by a precedence $\gg$ is defined as follows: For every pair of ground terms $s^0 = f(s^0_1,..,s^0_n)$ and $t^0 = g(t^0_1,..,t^0_m)$ we have $s^0 \succ_{lpo} t^0$ *iff* one of the following conditions holds:

(**1**) $s^0_i \succeq_{lpo} t^0$ for some i with $1 \le i \le n$, or

(**2**) $f \gg g$ and $s^0 \succ_{lpo} t_j^0$ for all j with $1 \leq j \leq m$, or
(**3**) $f = g$ (and hence $m = n$), and $(s_1^0,..,s_n^0) \succ_{lex}^{lpo} (t_1^0,..,t_n^0)$,
where $\succ_{lex}^{lpo}$ *is the lexicographic extension of* $\succ_{lpo}$.
◈

Analogs of Theorem 2.17 and Proposition 2.18 can be shown for *LPO*-orderings:

Theorem 2.20 ([see Baader & Nipkow, 1998, Theorem 5.4.14]). *For any precedence* $\gg$ *on functional symbols* Fun*, the ordering* $\succ_{lpo}$ *induced by* $\gg$ *is a simplification ordering on* Tm_Σ^0.

Proposition 2.21. *Let* $\gg$ *be a total precedence on* Fun*, then the ordering* $\succ_{lpo}$ *induced by* $\gg$ *is a total ordering on* Tm_Σ^0.

Proof. The proof of this proposition is slightly more involved than for the case of *KBO*-orderings. By induction on $|s^0|+|t^0|$ we prove that either $s^0 = t^0$ or $s^0 \succ_{lpo} t^0$ or $t^0 \succ_{lpo} s^0$. Let $s^0 = f(s_1^0,..,s_n^0)$ and $t^0 = g(t_1^0,..,t_m^0)$.

We may assume that $s_i^0 \not\succeq_{lpo} t^0$ and $t_j^0 \not\succeq_{lpo} t^0$ for all i with $1 \leq i \leq n$ and all j with $1 \leq j \leq m$ (otherwise $s^0 \succ_{lpo} t^0$ or, respectively $t^0 \succ_{lpo} s^0$ by condition (**1**) from Definition 2.16). By induction hypothesis, we should have $t^0 \succ_{lpo} s_i^0$ and $s^0 \succ_{lpo} t_j^0$ for all these pairs of terms since $|t^0| + |s_i^0| < |s^0| + |t^0|$ and $|s^0| + |t_j^0| < |s^0| + |t^0|$ for all i with $1 \leq i \leq n$ and all j with $1 \leq j \leq m$.

Consider first the case when $f \neq g$. Since $\gg$ is total, then either $f \gg g$ or $g \gg f$. In this case, condition (**2**) from Definition 2.16 implies that $s^0 \succ_{lpo} t^0$ or $t^0 \succ_{lpo} s^0$ respectively.

If $f = g$ then condition (**3**) from Definition 2.16 is applied, by which the terms t^0 and s^0 should be comparable, since the lexicographic extension of a total ordering is a total ordering by Proposition 2.15. ◫

There are variations of *KBO* and *LPO*-orderings, where the condition (**3**) is extended for functional symbols with status (for *LPO* ordering this extension is known as the *recursive path ordering with status*, or short *RPOS* [see Dershowitz, 1987]). A *status* $status(f)$ of a functional symbol $f \in$ Fun is an element from {`left`, `right`, `multiset`}. Definition 2.16 and Definition 2.19 can be extended to take the status of symbols into account by replacing the condition (**3**) with:

(**3′**) $\cdots$ $f = g$ (and hence $m = n$), and either
($\boldsymbol{l}$) $status(f) =$ `left` and $(s_1^0,..,s_n^0) \succ_{lex}^{*} (t_1^0,..,t_n^0)$, or
($\boldsymbol{r}$) $status(f) =$ `right` and $(s_n^0,..,s_1^0) \succ_{lex}^{*} (t_n^0,..,t_1^0)$, or
($\boldsymbol{m}$) $status(f) =$ `multiset` and $\{s_1^0,..,s_n^0\}_m \succ_{mul}^{*} \{t_1^0,..,t_n^0\}_m$,
where $* \in \{kbo, lpo\}$ *and* $\succ_{lex}^{*}$, $\succ_{mul}^{*}$ *are the lexicographic and the multiset extensions for respective orderings.*

KBO and *LPO* orders can be used for ground expressions by treating predicate symbols and negation as functional symbols (i.e., by defining precedence and weight functions on them). In the following example, we demonstrate a difference between *KBO* and *LPO*-orderings, that is be important for saturation-based decision procedures.

Example 2.22. Consider two atoms: $\mathtt{p}(t^0, t^0)$ and $\mathtt{q}(\mathtt{f}(t^0))$, where $\mathtt{p}$ and $\mathtt{q}$ are predicate symbols, $\mathtt{f}$ is a functional symbol and t^0 is some ground term. Let $\gg$ be a precedence on predicate and functional symbols such that $\mathtt{f} \gg \mathtt{p}$ and let $\succ_{lpo}$ be the *LPO*-ordering induced by $\gg$. Then we have $\mathtt{q}(\mathtt{f}(t^0)) \succ_{lpo} \mathtt{p}(t^0, t^0)$. Indeed, $t^0 \succeq_{lpo} t^0$, therefore by condition (**1**) from Definition 2.19 we have $\mathtt{f}(t^0) \succ_{lpo} t^0$. Since $\mathtt{f} \gg \mathtt{p}$, by condition (**2**) we have $\mathtt{f}(t^0) \succ_{lpo} \mathtt{p}(t^0, t^0)$ which yields again by condition (**1**) that $\mathtt{q}(\mathtt{f}(t^0)) \succ_{lpo} \mathtt{p}(t^0, t^0)$. Note, that this holds for *every* ground term t^0.

However it is not possible to have $\mathtt{q}(\mathtt{f}(t^0)) \succ_{kbo} \mathtt{p}(t^0, t^0)$ for all terms t^0 using a *KBO*-ordering $\succ_{kbo}$. Indeed, for every admissible weight function $\mathit{weight}(\cdot)$, one can construct a term t^0 with a large enough weight so that $\mathit{weight}(\mathtt{p}(t^0, t^0)) = \mathit{weight}(\mathtt{p}) + 2{\cdot}\mathit{weight}(t^0) > \mathit{weight}(\mathtt{q}) + \mathit{weight}(\mathtt{f}) + \mathit{weight}(t^0) = \mathit{weight}(\mathtt{q}(\mathtt{f}(t^0)))$. ◈

2.4.3 The ordered Knuth-Bendix completion

Now we return to the congruence closure procedure described in subsection 2.3 and refine the calculus $\mathcal{KB}^0$ from System 1. The idea is to use a total reduction ordering $\succ$ to guide orientation of ground equations.

Orient	**Superpose**	**Superposition**
$\mathtt{O}: \dfrac{s^0 \simeq t^0}{s^0\Downarrow_R \Rightarrow t^0\Downarrow_R}$	$\mathtt{S}: \dfrac{s^0 \Rightarrow t^0 \quad w^0[s^0] \Rightarrow v^0}{w^0[t^0] \simeq v^0}$	$\mathtt{SP}: \dfrac{s^0 \simeq t^0 \quad w^0[s^0] \simeq v^0}{w^0[t^0] \simeq v^0}$
$\big[\mathit{if} \quad s^0\Downarrow_R \succ t^0\Downarrow_R. \big]$		$\big[\mathit{where}\ (\boldsymbol{i})\ s^0 \succ t^0\ \mathit{and}\ (\boldsymbol{ii})\ w^0[s^0] \succ v^0 \big]$

System 2: Ground ordered Knuth-Bendix completion $\mathcal{KB}^0_\succ$ and Superposition

In System 2 we formulated two variants of *ordered Knuth-Bendix completion*. The left variant is a refinement of $\mathcal{KB}^0$, where we have restricted applications of the rule Orient to produce only rewrite rules from larger terms to smaller ones (w.r.t. $\succ$).

Since a rewrite rule correspondent to an equation is completely determined by an ordering, one could join the Orient and Superpose rules into a single Superposition rule, shown in the right part of System 2. This rule is the main rule for equality handling in almost all saturation-based calculi for equational clause logic.

Application of Superposition always terminates for ground equalities when a total reduction ordering $\succ$ of ω-type is used. An ordering $\succ$ on $\boldsymbol{D}$ is of ω-type, if for every element $d \in \boldsymbol{D}$ there are at most *finitely many* smaller elements in $\boldsymbol{D}$ w.r.t. $\succ$, i.e., the set $\{d' \in \boldsymbol{D} \mid d \succ d'\}$ is finite. *KBO*-ordering with positive weights ($\mathit{weight}(f) > 0$ for all $f \in \mathrm{Fun}$) is an instance of such orderings as there are only finitely many ground terms of a bounded weight. *LPO* is an example of a simplification ordering which is not of ω-type. We shell see it in a moment below.

If an ordering is of ω-type, then Superposition can generate only finitely many equations, since the greatest (w.r.t. $\succ$) term in the conclusion is strictly smaller then the greatest term in the premise of this rule: $w^0[s^0] \succ w^0[t^0]$, $w^0[s^0] \succ v^0$, because of the conditions (***i***) and (***ii***) of the rule respectively. Since there are only finitely many terms that are smaller than a given term w.r.t. an ordering $\succ$, then only finitely many equations can be generated by the procedure.

Unfortunately, the completion procedure, as it is given by the Superposition rule, does not always terminate for ground equations if a total reduction order $\succ$ is not of ω-type. This is demonstrated in the following example:

Example 2.23. Let us perform completion for set $E = \{\mathtt{g}(\mathtt{f}(\mathtt{a})) \simeq \mathtt{f}(\mathtt{a}), \mathtt{f}(\mathtt{a}) \simeq \mathtt{a}\}$ of equations, using an *LPO*-ordering $\succ_{lpo}$ based on a precedence $\mathtt{f} \gg \mathtt{g} \gg \mathtt{a}$:

given : 1. $\mathtt{g}(\underline{\mathtt{f}(\mathtt{a})}) \simeq \mathtt{f}(\mathtt{a})$
given : 2. $\underline{\mathtt{f}(\mathtt{a})} \simeq \mathtt{a}$
SP[2; 1]: 3. $\underline{\mathtt{f}(\mathtt{a})} \simeq \mathtt{g}(\mathtt{a})$
SP[3; 1]: 4. $\underline{\mathtt{f}(\mathtt{a})} \simeq \mathtt{g}(\mathtt{g}(\mathtt{a}))$
SP[3; 2]: 5. $\underline{\mathtt{g}(\mathtt{a})} \simeq \mathtt{a}$
SP[4; 1]: 6. $\underline{\mathtt{f}(\mathtt{a})} \simeq \mathtt{g}(\mathtt{g}(\mathtt{g}(\mathtt{a})))$
SP[4; 2]: 7. $\mathtt{g}(\mathtt{g}(\mathtt{a})) \simeq \mathtt{a}$
SP[4; 3]: 8. $\mathtt{g}(\mathtt{g}(\mathtt{a})) \simeq \mathtt{g}(\mathtt{a})$
SP[6; 1]: 9. $\underline{\mathtt{f}(\mathtt{a})} \simeq \mathtt{g}(\mathtt{g}(\mathtt{g}(\mathtt{g}(\mathtt{a}))))$
SP[6; 2]: 10. $\mathtt{g}(\mathtt{g}(\mathtt{g}(\mathtt{a}))) \simeq \mathtt{a}$
SP[6; 3]: 11. $\mathtt{g}(\mathtt{g}(\mathtt{g}(\mathtt{a}))) \simeq \mathtt{g}(\mathtt{a})$
. . . etc . . .

It is easy to see that the inference procedure does not terminate. This is because there are infinitely many ground terms smaller than $\mathtt{f}(\mathtt{a})$ w.r.t. $\succ_{lpo}$ (in particular, all terms constructed from g and a only). Termination can be regained for arbitrary total reduction ordering [see Exercise 6.7 in Baader & Nipkow, 1998], when deletion of unnecessary equations is applied. Note, that the right premise of the Superposition rule is no longer needed after this rule is applied, since it follows from the other two equations containing smaller terms. Speaking in general terms, such an equation is *redundant* and can be *deleted* from the set. In the example above, equation 1 becomes redundant after the step 3, thus, all inferences, except 5 are not needed and the procedure terminates. The resulted rewrite system consists of the rules obtained by orienting equations 2 and 5 into $R := \{\mathtt{f}(\mathtt{a}) \Rightarrow \mathtt{a},\ \mathtt{g}(\mathtt{a}) \Rightarrow \mathtt{a}\}$. ⋘

2.5 Substitutions And Unification

In clause logic we deal with expressions that may possibly contain variables. In this section we introduce some necessary notions and prove important facts regarding

non-ground expressions.

A *substitution* is a function that maps variables to terms $\sigma : \mathrm{Var} \to \mathrm{Tm}_\Sigma$, which is denoted by $\sigma = \{x_1/t_1, x_2/t_2,.., x_n/t_n,..\}$, or, shortly $\sigma = \{\overline{x}/\overline{t}\}$ (hereby $\sigma(x_i) = t_i$). The *domain* of a substitution $Dom(\sigma) := \{x_i \in \mathrm{Var} \mid \sigma(x_i) \neq x_i\}$. The *range* of a substitution $Ran(\sigma) := \{\sigma(x_i) \mid x_i \in Dom(\sigma)\}$. The *identity substitution* is a substitution such that $id(x) = x$ for every $x \in \mathrm{Var}$. In other terms, $Dom(id) = \{\}$. A substitution σ is a *renaming*, if (***i***) $Ran(\sigma) \subseteq \mathrm{Var}$ and (***ii***) $x \neq y$ implies $\sigma(x) \neq \sigma(y)$. A substitution σ^0 is *ground iff* $\sigma(x) \in \mathrm{Tm}^0_\Sigma$ for every $x \in \mathrm{Var}$.[3] Given a set of variables $V \subseteq \mathrm{Var}$, a *restriction of* a substitution σ to V is a substitution $\sigma_{|V}$, that is defined as follows:

$$\sigma_{|V}(x) := \begin{cases} \sigma(x) & \text{if } x \in V; \\ x & \text{if } x \notin V. \end{cases}$$

The *application* of a substitution to expressions and formulas is defined recursively in Figure 1. A *composition of substitutions* σ_1 and σ_2 is a new substitution denoted

Figure 1 Application of a substitution

$$\begin{array}{l|l} t\cdot\sigma := \quad x\cdot\sigma = \sigma(x) & F\cdot\sigma := A\cdot\sigma = A\cdot\sigma \\ f(t_1,t_2,..,t_n)\cdot\sigma = f(t_1\cdot\sigma,t_2\cdot\sigma,..,t_n\cdot\sigma)\ . & (F_1 \between F_2)\cdot\sigma = (F_1\cdot\sigma) \between (F_2\cdot\sigma) \\ \hline A\cdot\sigma := & (\neg F_1)\cdot\sigma = \neg(F_1\cdot\sigma) \\ a(t_1,t_2,..,t_n)\cdot\sigma = a(t_1\cdot\sigma,t_2\cdot\sigma,..,t_n\cdot\sigma)\ . & (Qy.F_1)\cdot\sigma = Qy.(F_1\cdot\sigma_{|\mathrm{Var}\setminus\{y\}})\ . \end{array}$$

by $\sigma_1{\cdot}\sigma_2$ that is defined as follows: $(\sigma_1{\cdot}\sigma_2)(x) := (\sigma_1(x)){\cdot}\sigma_2$. It can be shown by induction over the definitions of Tm_Σ and Fm_Σ that for every $t \in \mathrm{Tm}_\Sigma$ and $F \in \mathrm{Fm}_\Sigma$: $t{\cdot}(\sigma_1{\cdot}\sigma_2) = (t{\cdot}\sigma_1){\cdot}\sigma_2$ and $F{\cdot}(\sigma_1{\cdot}\sigma_2) = (F{\cdot}\sigma_1){\cdot}\sigma_2$. Hence, composition of substitutions is an associative operation. Because of this, we will often omit braces in sequences of substitution applications: $F{\cdot}\sigma_1{\cdot}\sigma_2{\cdot}\cdots\sigma_n$. Note that $\sigma_1{\cdot}\sigma_2 = id$, implies that both σ_1 and σ_2 are renamings.

An expression E_2 is an *instance* of an expression E_1 (notation: $E_2 \gtrsim_i E_1$ and, symmetrically $E_1 \lesssim_i E_2$), *iff* there exists a substitution σ_1 such that $E_2 = E_1{\cdot}\sigma_1$. Without loss of generality, one can assume that $Dom(\sigma_1) \subseteq free[E_1]$ since otherwise, one can take $\sigma_{1|free[E_1]}$ instead of σ_1. Note that $\gtrsim_i$ is a quasi-ordering (see subsection 2.4), since: (***i***) $E = E{\cdot}id$ implies reflexivity of $\gtrsim_i$ and (***ii***) $E_2 = E_1{\cdot}\sigma_1$ and $E_3 = E_2{\cdot}\sigma_2$ imply $E_3 = E_1{\cdot}\sigma_3$, where $\sigma_3 = \sigma_1{\cdot}\sigma_2$, which implies transitivity of $\gtrsim_i$.

Let $\sim_i$ denote the equivalence relation induced by $\gtrsim_i$ (i.e., $E_1 \sim_i E_2$ *iff* $E_1 \gtrsim_i E_2$ and $E_2 \gtrsim_i E_1$). Then $E_1 \sim_i E_2$ if there exists a renaming σ_1 such that $E_2 = E_1{\cdot}\sigma_1$. Indeed, from $E_1 = E_2{\cdot}\sigma_2$ and $E_2 = E_1{\cdot}\sigma_1$ (w.l.o.g. $Dom(\sigma_1) = free[E_1]$), one implies

[3] It is generally assumed that every substitution has a finite domain, which we do not require in this report, since otherwise it is more tricky to define the notion of a ground substitution

that $E_1 = E_1{\cdot}\sigma_1{\cdot}\sigma_2$, so, $\sigma_1{\cdot}\sigma_2 = \sigma_{1|_{free[E_1]}}{\cdot}\sigma_2 = id_{|_{free[E_1]}} = id$, which implies that both σ_1 and σ_2 are renaming substitutions. The *strict instance* ordering $>_i$ is the difference between $\gtrsim_i$ and $\sim_i$.

The instance ordering can be extended on substitutions by defining $\sigma_2 \gtrsim_i \sigma_1$ *iff* $\sigma_2 = \sigma_1{\cdot}\sigma$ for some substitution σ. In this case σ_2 is called an *instance of* (or *more specific than*) σ_1, and σ_1 is called *more general* than σ_2. The relation $\gtrsim_i$ is a quasi-ordering on substitutions, which can be shown similarly to $\gtrsim_i$ on expressions. Note also that *id* is the greatest, i.e., the *most specific* element w.r.t. $\gtrsim_i$.

A *unification problem*[4] is a set $P = \{E_1{=}E'_1,.., E_n{=}E'_n\}$ of equations between expressions, $n \geq 0$. A *solution* for a unification problem P, called a *unifier* is a substitution σ such that $E_i{\cdot}\sigma = E'_i{\cdot}\sigma$ for every i with $1 \leq i \leq n$. Note that any substitution is a unifier for the *empty* unification problem (i.e., when $n = 0$). A *most general unifier* (or, shortly *mgu*) for the unification problem P is a unifier σ such that it is more general than any other unifier σ', i.e., $\sigma' \gtrsim_i \sigma$. Note that this definition implies that a most general unifier is unique up to a renaming, i.e., if σ_1 and σ_2 are two *mgu*'s for P, then $\sigma_1 \sim_i \sigma_2$. Below we give an algorithm that computes a most general unifier for a unification problem.

Figure 2 A unification procedure

$mgu(P) :=$	$mgu(\{\})$	$= id$	|	(Empty)
	$mgu(P \sqcup \{e(s_1,.., s_n){=}e(t_1,.., t_n)\})$	$= mgu(P \sqcup \{s_1{=}t_1,.., s_n{=}t_n\})$	|	(Decompose)
	$mgu(P \sqcup \{e(s_1,.., s_n){=}e'(t_1,.., t_m)\})$	$= \bot$, if $e \neq e'$	|	(Clash)
	$mgu(P \sqcup \{x{=}t\})$	$= \bot$, if $x \lhd t$	|	(Occurs-Check)
	$mgu(P \sqcup \{s{=}y\})$	$= mgu(P \sqcup \{y{=}s\})$, if $s \notin \mathrm{Var}$	|	(Orient)
	$mgu(P \sqcup \{x{=}t\})$	$= \{x/t\}{\cdot}mgu(P{\cdot}\{x/t\})$, if $x \not\lhd t$.		(Eliminate)

In Figure 2 we define by induction a *unification function*, that given a unification problem P, returns an *mgu* for it, if there exists one. We introduce a special *undefined substitution* $\bot$ for the case when no *mgu* exists for P to make the function $mgu(\cdot)$ total. We adopt a convention, according to which $\bot$ maps every term to a special *undefined constant* **?**. So, $\bot{\cdot}\sigma = \bot$ for any substitution σ and $\bot$ is a unifier for every unification problem (the *most specific* unifier). For a unification problem $P = \{E_1{=}E'_1,.., E_n{=}E'_n\}$ and a substitution σ, $P{\cdot}\sigma$ denotes the unification problem $\{E_1{\cdot}\sigma{=}E'_1{\cdot}\sigma,.., E_n{\cdot}\sigma{=}E'_n{\cdot}\sigma\}$. Throughout this report we assume that $mgu(P)$ is the most general unifier returned by the unification procedure in Figure 2. When P consists of one equation $P = \{E{=}E'\}$, we usually write $mgu(E, E')$ instead of $mgu(\{E{=}E'\})$.

Proposition 2.24. *The unification procedure defined in Figure 2 terminates for every input P and produces a substitution σ which is a most general unifier for P.*

[4]In this report we are concerned only with the *syntactic* unification problems

Proof. First we show that the unification procedure terminates. Let $|P|$ be the *size* of P and $vars[P]$ be the *set of variables* in P for a unification problem P that are defined in the following way:

$$\begin{array}{rcl|rcl} |P| := |\{\}| & = & 0 & vars[P] := vars[\{\}] & = & \{\} \\ |\{s{=}t\}| & = & |s| + |t|, \text{ if } t \notin \text{Var} & vars[\{s{=}t\}] & = & vars[s] \cup vars[t] \\ |\{s{=}y\}| & = & |s| + |y| + 1 & vars[P_1 \sqcup P_2] & = & vars[P_1] \cup vars[P_2] \; . \\ |P_1 \sqcup P_2| & = & |P_1| + |P_2| \; . & & & \end{array}$$

Note that in every recursion call of function $mgu(\cdot)$ in Figure 2, either the number of variables $\sharp vars[P]$ decreases, or $\sharp vars[P]$ remains unchanged, but $|P|$ decreases. Therefore the function $mgu(P)$ always terminates. Now we can use induction over it to prove properties for its returned value. It is not difficult to show the following properties for $\sigma = mgu(P)$ by induction over $mgu(\cdot)$: ($\boldsymbol{i}$) σ is a unifier for P and ($\boldsymbol{ii}$) for every unifier σ' for P, $\sigma' \gtrsim_i \sigma$. So, σ is a most general unifier for P. ⧉

Remark 2.25. The unification algorithm given in Figure 2 is not optimal for the unification problem. There are examples [see Example 4.6.11 in Baader & Nipkow, 1998], where this algorithm requires an exponential space (and time). It is possible to implement a unification procedure in almost linear time and in linear space [for discussion of complexity issues for unification algorithms, please, refer to Baader & Nipkow, 1998]. We use the simple unification algorithm above to study local properties of unification for some classes of expressions that we define below. ⋘

2.6 Covering Expressions and Atomic Substitutions

The notion of covering expressions has been introduced by Fermüller et al. [1993] to describe resolution decision procedures for certain clause classes. The following notions and results originate from [Fermüller et al., 1993].

Definition 2.26 (Atomic). A term is *atomic* if it is either a constant or a variable. A substitution σ is *atomic on a set of variables* V, if for every $x \in V$, $\sigma(x)$ is atomic. A substitution σ is *atomic* if σ is atomic on Var (equivalently on $Dom(\sigma)$). ⋘

The nice property of atomic substitutions is that they do not change the depth of the expression when applied to it:

Proposition 2.27. *Let E be an expression and σ be a substitution that is atomic on $vars[E]$. Then $depth(E{\cdot}\sigma) = depth(E)$.*

Definition 2.28 (Covering). An expression E *covers a set of variables* V (notation: $E \propto V$) *iff* every non-atomic subterm t of E contains all variables form V ($V \subseteq vars[t]$). An expression E_1 *covers an expression* E_2 (or a clause C) (notation:

$E_1 \propto E_2$, $E_1 \propto C$) *iff* E_1 covers *vars*$[E_2]$ (*vars*$[C]$). An expression E is *covering iff* $E \propto E$. A clause C is covering *iff* all literals from C cover C. ⋘

Note that from Definition 2.28 it follows that for every expression E that covers a set of variables V, every subterm t of E also covers V. In symbols: $t \trianglelefteq E \propto V$ implies $t \propto V$. Note that every expression containing only atomic subterms covers every set of variables and every expression. Hence the relation $\propto$ is not transitive on expressions, since, in particular $\mathtt{f(x)} \propto \mathtt{x} \propto \mathtt{f(y)}$, but $\mathtt{f(x)} \not\propto \mathtt{f(y)}$.

Example 2.29. The following expressions cover the set of variables $V = \{\mathtt{x}, \mathtt{y}\}$:
$\mathtt{a(c, f(x, y))}$; $\mathtt{a(h(x, x, c, y), y)}$; $\mathtt{a(h(x, x, f(x, y), f(y, x)), c)}$; $\mathtt{a(x, c)}$; $\mathtt{x}$; $\mathtt{c}$; $\mathtt{z}$; $\mathtt{a(x, z)}$; $\mathtt{a(c, h(x, z, c, y), y)}$ and $\mathtt{a(f(x, y), z)}$.
All expressions in the first line cover each other. Every expression in the second line covers every expression in the first line. The last expression covers none of the expressions in the last line. The expression before, covers all expressions. All but the last expressions are covering.

The following expressions do not cover $V = \{x, y\}$: $\mathtt{a(g(x), x)}$; $\mathtt{a(c, f(y, c))}$; $\mathtt{a(x, f(x, z))}$; $\mathtt{a(f(x, y), g(z))}$; $\mathtt{a(g(c), f(x, y))}$; and $\mathtt{a(f(x, x), y)}$.
However first three of them are covering. First two and last two expressions are covered by every expression from the first group. ⋘

Covering expression often appear in the result of Skolemization for relational first-order formulas. Skolemization is a transformation of first-order formulas that introduces new functions for existential quantified variables of a formula according to the *Axiom of Choice* which can be concisely formulated as follows: "If $\forall\overline{x}.\exists y.F[\overline{x}, y]$ holds then there exists a function $\mathtt{f}_{\exists y.F}(\overline{x})$ such that $\forall\overline{x}.F[\overline{x}, \mathtt{f}_{\exists y.F}(\overline{x})]$ holds". Skolemization procedure(s) shall be discussed in details in subsection 6.3. Here we just note, that the function $\mathtt{f}_{\exists y.F}(\overline{x})$ (called the *Skolem function*) that replaces existentially quantified variable, usually cover the subformula followed by the existential quantifier for which it has been introduced (i.e., it contains all its variables). This results in covering expressions.

The class of covering expressions can be extended to so-called *weakly covering* expressions. This notion has been used for defining many decidable clause classes, including $\mathcal{E}^+$ [Fermüller et al., 1993] and a clause class for the guarded fragment [de Nivelle, 1998*b*; de Nivelle & de Rijke, 2003]. We will not make use of weakly covering expressions in our proofs, but we include their definition for the sake of completeness.

Definition 2.30 (Weakly Covering). An expression E *weakly covers* a set of variables V, *iff* every non-ground subterm t of E contains all variables from V ($V \subseteq$ *vars*$[t]$). An expression E_1 *weakly covers* an expression E_2 (or a clause C), *iff* E_1 weakly covers *vars*$[E_2]$ (*vars*$[C]$). An expression E is *weakly covering iff* E

weakly covers E. A clause C is *weakly covering iff* all literals in C weakly cover C. ⋘

Every covering expression is obviously, a weakly covering expression. The converse does not hold in general: the expression $\mathtt{a}(\mathtt{x}, \mathtt{f}(\mathtt{c}))$ is weakly covering, but not covering. Weakly covering expressions have similar properties to covering expressions, if one think of ground subterms (such as $\mathtt{f}(\mathtt{c})$) as new constant names.

The following simple property relates covering expressions and atomic substitutions, and can bee seen a s a converse of Proposition 2.27:

Proposition 2.31. *Let E be an expression that is covering for a set of variables V ($E \propto V$). Then for any substitution σ, $depth(E{\cdot}\sigma) = depth(E)$ implies that σ is atomic on V.*

Proof. The property can be shown by a simple induction on $depth(E)$. ⧉

Now we are going to give a lemma that plays an important rôle for proving decidability of some clause classes by resolution.

Lemma 2.32. *Let E_1 and E_2 be covering expressions such that $depth(E_1) \geq depth(E_2)$. Let $\sigma = mgu(E_1{=}E_2)$. Then σ is atomic on vars$[E_1]$.*

The lemma essentially says that unification of covering expressions does not enlarge their maximal depth, since by Proposition 2.27, $depth(E_2{\cdot}\sigma) = depth(E_1{\cdot}\sigma) = depth(E_1)$. This argument can be used in saturation-based decision procedures for showing that the depth of generated clauses does not grow beyond a certain limit. To give a more concise proof of Lemma 2.32, we extend the covering relation and formulate a more general result for unification problems, which will be proven by induction over the function $mgu(P)$ given in Figure 2.

Definition 2.33 (Variable Superset). A *variable superset* $\overline{V}$ is a set of variable sets which is denoted by $\overline{V} = \{V_1, .., V_p\}$, where $p \geq 0$ and V_i with $1 \leq i \leq p$ are sets of variables ($V_i \subseteq$ Var, it is not required that $V_i \cap V_j = \{\}$ for $i \neq j$). We say that an expression E *covers a variable superset* $\overline{V}$ (notation: $E \propto \overline{V}$), if every non-atomic subterm t of E contains all variables from *some* $V \in \overline{V}$. We say that a *unification problem* $P := \{E_1{=}E'_1, .., E_n{=}E'_n\}$ *covers a variable superset* $\overline{V}$ (notation: $P \propto \overline{V}$), if $E_i \propto \overline{V}$ and $E'_i \propto \overline{V}$ for every i with $1 \leq i \leq n$. ⋘

Example 2.34. Consider a variable superset $\overline{V} := \{\{\mathtt{x}, \mathtt{y}\}, \{\mathtt{y}, \mathtt{z}\}\}$. The following expressions cover $\overline{V}$:

$\mathtt{a}(\mathtt{f}(\mathtt{x}, \mathtt{y}), \mathtt{f}(\mathtt{y}, \mathtt{z}))$; $\mathtt{a}(\mathtt{c}, \mathtt{f}(\mathtt{y}, \mathtt{x}))$; $\mathtt{a}(\mathtt{y}, \mathtt{z})$; $\mathtt{a}(\mathtt{x}, \mathtt{z})$; $\mathtt{a}(\mathtt{c}, \mathtt{y})$; $\mathtt{x}$.

The following expressions do not cover $\overline{V}$:

$\mathtt{a}(\mathtt{c}, \mathtt{f}(\mathtt{c}, \mathtt{c}))$; $\mathtt{a}(\mathtt{z}, \mathtt{f}(\mathtt{y}, \mathtt{c}))$; $\mathtt{a}(\mathtt{f}(\mathtt{x}, \mathtt{y}), \mathtt{f}(\mathtt{x}, \mathtt{z}))$. ⋘

Proposition 2.35. *Let E be an expression and $\overline{V}$ be a variable superset. Then $E \propto \overline{V}$ iff $t \propto \overline{V}$ for every term $t \trianglelefteq E$.*

Let $\overline{V}$ be a variable superset, and x be a variable. We denote by $\overline{V}\backslash\backslash x$ a variable superset obtained from $\overline{V}$ by removing all sets $V \in \overline{V}$ that contain the variable x: $\overline{V}\backslash\backslash x := \{V \in \overline{V} \mid x \notin V\}$.

Proposition 2.36. *Let E_1 be an expression, t be a non-atomic term and $\overline{V}_1$, $\overline{V}_2$ be variable supersets such that $E_1 \propto \overline{V}_1$ and $t \propto \overline{V}_2$. Then for every $x \in \mathrm{Var}$, $E_1{\cdot}\{x/t\} \propto ((V_1\backslash\backslash x) \cup V_2)$.*

Proof. Consider any non-atomic subterm t' of $E_1{\cdot}\{x/t\}$. Clearly, there are two cases possible:
(***i***) $t' = t'_1{\cdot}\{x/t\}$, where t'_1 is a subterm of E_1. If $x \notin \mathit{vars}[t'_1]$ then $t' = t'_1 \propto \overline{V}_1\backslash\backslash x$ and there exists $V_1 \in (\overline{V}_1\backslash\backslash x)$ such that $V_1 \subseteq \mathit{vars}[t'_1] = \mathit{vars}[t']$. If $x \in \mathit{vars}[t'_1]$ then t is a subterm of t', hence there exists $V_2 \in \overline{V}_2$ such that $V_2 \subseteq \mathit{vars}[t] \subseteq \mathit{vars}[t']$.
(***ii***) t' is a subterm of t. Then $V_2 \subseteq \mathit{vars}[t']$ for some $V_2 \in \overline{V}_2$. ⧉

Lemma 2.37 (Covering Lemma). *Let P be a unification problem and $\overline{V}$ be a variable superset such that $P \propto \overline{V}$. Let $\sigma := \mathit{mgu}(P)$. Then σ is atomic on some $V \in \overline{V}$.*

Proof. We prove the lemma by induction over the definition of the function $\mathit{mgu}(P)$ given in Figure 2. For the base case (Empty), the lemma holds, since id is an atomic substitution. For the cases (Clash) and (Occurs-Check), the lemma also holds, since $\bot$ is an atomic substitution as well (it maps every variable to the undefined *constant*). The transformations of the unification problem in cases (Decompose) and (Orient) preserve the conditions of the lemma (see Proposition 2.35), and do not change the *mgu* that is returned, so these cases are trivial. The only non-trivial step is (Eliminate) which we consider in more detail. Assume that the conditions of the lemma holds for the unification problem $P \cup \{x{=}t\}$. Consider two cases:
(***i***) If t is not an atomic term, then $x \notin \mathit{vars}[t]$ (otherwise $x \lhd t$), thus $t \propto (\overline{V}\backslash\backslash x)$. By Proposition 2.36, $P{\cdot}\{x/t\} \propto ((\overline{V}\backslash\backslash x) \cup (\overline{V}\backslash\backslash x)) = (\overline{V}\backslash\backslash x)$. So, we can apply the induction hypothesis, by which we obtain that $\sigma' = \mathit{mgu}(P{\cdot}\{x/t\})$ is atomic on some $V \in (\overline{V}\backslash\backslash x)$. Since $x \notin V$, the substitution $\sigma = \{x/t\}{\cdot}\sigma'$ is atomic on V.
(***ii***) If t is atomic, then $P{\cdot}\{x/t\} \propto \overline{V}'$, where $\overline{V}' := \{((V \setminus \{x\}) \cup \mathit{vars}[t]) \mid V \in \overline{V}\}$. By induction hypothesis, $\sigma' = \mathit{mgu}(P{\cdot}\{x/t\})$ is atomic on some $V' = ((V \setminus \{x\}) \cup \mathit{vars}[t]) \in \overline{V}'$. Thus $\sigma = \{x/t\}{\cdot}\sigma'$ is atomic on V. ⧉

Proof of Lemma 2.32. Given a unification problem $P = \{E_1{=}E_2\}$, we apply Lemma 2.37 for variable superset $\overline{V} := \{\mathit{vars}[E_1], \mathit{vars}[E_2]\}$. As a conclusion,

we obtain that $\sigma = mgu(P)$ is atomic on either $vars[E_1]$ or on $vars[E_2]$. Suppose σ is *not* atomic on $vars[E_1]$. Then σ is atomic on $vars[E_2]$ and $depth(E_2) = depth(E_2{\cdot}\sigma) = depth(E_1{\cdot}\sigma) > depth(E_1)$ (see Propositions 2.27 and Theorem 2.31), which contradicts the assumption $depth(E_1) \geq depth(E_2)$. ⧉

At the end, we prove one more useful property for the covering relation that describes its monotonicity w.r.t. substitutions.

Proposition 2.38. *Let E_1, E_2 be expressions such that $E_1 \propto E_2$ and $vars[E_2] \subseteq vars[E_1]$. Then for every substitution σ:*

(***a***) *$depth(E_1) > depth(E_2)$ implies $depth(E_1{\cdot}\sigma) > depth(E_2{\cdot}\sigma)$;*

(***b***) *$depth(E_1) = depth(E_2)$ implies $depth(E_1{\cdot}\sigma) \geq depth(E_2{\cdot}\sigma)$;*

Proof. The proposition can be easily proven by induction on $depth(E_1)$. ⧉

2.7 Saturation-Based Theorem Proving

A *saturation-based (= refutational) theorem prover* works with a set of clauses by applying *inference rules* from a certain parametrised inference system called a *calculus*. We have already seen examples of calculi in subsection 2.3 and subsection 2.4 which describe the Knuth-Bendix completion. Finding a completion which we call a *saturation* w.r.t. an inference system is not the primary goal of calculi used in theorem provers. Their objective is to find a *refutation* of an input clause set which is a derivation of a basic contradiction, and thereof to establish unsatisfiability of the input clause set. On the other hand, (finite) saturations can be used as an effective witness for satisfiability of an input clause set.

Starting from this section, we refer to a clause $C = L_1 \vee \cdots \vee L_k$ as to the multiset consisting of its literals $L = \{L_1, .., L_k\}_m$. In other words, we do not distinguish clauses that differ only in permutation of literals. We could have represented a clause by a set of literals, however, as will be seen later, the multiset representation is more suitable for completeness proofs. The *empty clause*, i.e., the disjunction of no literals is denoted by $\Box$. Semantically, the empty clause is a neutral element w.r.t. disjunction and hence is always `false`. The empty clause plays the rôle of a basic contradiction that one needs to derive to establish unsatisfiability of a clause set.

Definition 2.39 (Inference, Inference System, Derivation). An *inference (from k clauses)* is an element $\pi \in \mathrm{Cl}_\Sigma^{k+1}$ which is written in the form $C_1,..,C_k \vdash C$, where $k \geq 0$ and $C_i \in \mathrm{Cl}_\Sigma$, $C \in \mathrm{Cl}_\Sigma$ are first-order clauses for $1 \leq i \leq k$. The set of all inferences (with k premises) for a signature Σ is denoted by Inf_Σ (Inf_Σ^k). An *inference rule (with k premises)* is a subset $R_k \subseteq \mathrm{Inf}_\Sigma^k$ of inferences with k premises.

We will write $R_k : C_1,..,C_k \vdash C$ or $R[C_1,..,C_k] : C$ instead of $(C_1,..,C_k \vdash C) \in R$. An *inference system* $\boldsymbol{\mathcal{S}}$ is a finite set of inference rules.

Given an inference $\pi = (C_1,..,C_k \vdash C) \in \mathrm{Inf}_\Sigma$, we write $N \vdash_\pi C$ if $\{C_1,..,C_k\} \subseteq N$. In this case we say that the clause C is *π-derived in one step from N*. This relation is straightforwardly extended to inference rules and inference systems: $N \vdash_R C$ if $N \vdash_\pi C$ for some $\pi \in R$; $N \vdash_{\boldsymbol{\mathcal{S}}} C$ if $N \vdash_R C$ for some $R \in \boldsymbol{\mathcal{S}}$. ⋘

We often refer to an inference system $\boldsymbol{\mathcal{S}}$ as to the set of its inferences: $\{\pi \in \mathrm{Inf}_\Sigma \mid R : \pi \in \boldsymbol{\mathcal{S}}\}$. For a set of inferences $S \subseteq \mathrm{Inf}_\Sigma$ we denote $S(N) := \{C \in \mathrm{Cl}_\Sigma \mid N \vdash_\pi C, \pi \in S\}$ to be the set of all clauses that are S-derivable from N in one step. Similar notation is applied to any inference π, inference rule R and inference system $\boldsymbol{\mathcal{S}}$, where $\pi(N)$, $R(N)$ and $\boldsymbol{\mathcal{S}}(N)$ denote the set of clauses that are respectively π-derivable, R-derivable or $\boldsymbol{\mathcal{S}}$-derivable from N in one step.

Definition 2.40 (Saturation, Soundness and Completeness). A clause set N is called *$\boldsymbol{\mathcal{S}}$-saturated* if $\boldsymbol{\mathcal{S}}(N) \subseteq N$. The *$\boldsymbol{\mathcal{S}}$-saturation* of a clause set N_0 is the smallest saturated set $\boldsymbol{\mathcal{S}}^*(N_0)$ containing N_0. For a clause $C \in \boldsymbol{\mathcal{S}}^*(N_0)$ we also say that *C is derivable from N_0 (in several steps)*.

An inference system $\boldsymbol{\mathcal{S}}$ is *sound* if $\square \notin \boldsymbol{\mathcal{S}}^*(N_0)$ for no satisfiable clause set N_0. An inference system $\boldsymbol{\mathcal{S}}$ is *(refutationally) complete for a clauses set $\boldsymbol{\mathcal{N}} \subseteq \mathrm{Cl}_\Sigma$* if $\square \in \boldsymbol{\mathcal{S}}^*(N_0)$ for every unsatisfiable clause set $N_0 \subseteq \boldsymbol{\mathcal{N}}$. ⋘

Remark 2.41. Note that the saturation of a clause set N_0 is the minimal fixed-point w.r.t. the operator $N \mapsto N \cup \boldsymbol{\mathcal{S}}(N)$. Hence, by the fixed-point theorem, for every $N_0 \subseteq \mathrm{Cl}_\Sigma$, its saturation $\boldsymbol{\mathcal{S}}^*(N_0)$ always exists and unique. Moreover, every $C \in \boldsymbol{\mathcal{S}}^*(N_0)$ belongs to the result of some *finite* iteration of this operator: $N_0' \cup \boldsymbol{\mathcal{S}}(N_0' \cup \cdots \cup \boldsymbol{\mathcal{S}}(N_0'))$, where N_0' is a *finite* subset of N_0, i.e., $C \in \boldsymbol{\mathcal{S}}^*(N_0')$. In other words, every clause that is $\boldsymbol{\mathcal{S}}$-derivable from N_0 can be derived from some *finite* subset of N_0 in *finitely* many steps. ⋘

In most inference systems that we introduce, conclusion of every rule is a logical consequences of the premises of this rule.[5] Therefore, soundness of such inference system is trivial. Although any sound inference systems can be used for proof search, calculi that are *refutationally complete* are especially useful in automated deduction, since they (theoretically) allow one to find a proof for every provable first-order formula. Therefore a considerable effort is put towards finding efficient refutationally complete saturation strategies.

Every non-ground clause can be essentially seen as a representation for the set of all its ground instances. Indeed, the Herbrand theorem (Theorem 2.6) implies the following:

[5]However in section 7 we introduce inference rules that do not have this property

Lemma 2.42 (Lifting Lemma). *Let N be a clause set over a signature Σ containing at least one constant symbol and let N^{gr} be the set of all* ground instances *of N. Then N is satisfiable iff N^{gr} is satisfiable.*

Lemma 2.42 provides a main theoretical foundation for saturation-based theorem proving. It allows one to design a calculus operating on the *ground level*, i.e., with ground clauses only, and then, to apply the lifting lemma for obtaining a calculus for non-ground clauses. For the last step, unification is often employed as an effective means of computing a representation for the set of common ground instances of expressions.

The most commonly used technique nowadays for proving completeness of saturation-based calculi is the *model generation method* developed by Bachmair & Ganzinger [1990, 1994]. Using this method, simple completeness proofs for many calculi can be obtained in a uniform way. Moreover, this method allows one to formulate a quite general notion of redundancy that gives rise to numerous refinements of saturation procedures.

3 The Ordered Resolution Calculus

Resolution calculus was invented by Robinson [1965] and later became the most successful method for automated deduction in first-order logic. The *Resolution calculus* in its basic form is given in System 3. This calculus consists of two

(Binary) Resolution	**(Positive) Factoring**
$\mathtt{R}: \dfrac{C \vee \underline{A} \quad D \vee \neg \underline{B}}{C\sigma \vee D\sigma}$	$\mathtt{F}: \dfrac{C \vee \underline{B} \vee \underline{A}}{C\sigma \vee A\sigma}$
$\big[$*where* $\sigma = mgu(A, B)$. $\big]$	$\big[$*where* $\sigma = mgu(A, B)$. $\big]$

System 3: The Resolution calculus $\mathcal{R}$

inference rules: Resolution and Factoring. These rules are applied to first-order clauses and produce new clauses using a unifier for a pair of its literals. We underline expressions that are to be unified in inference rules to improve their readability.

In this section we demonstrate the main principles of saturation-based calculi and the model generation method. We give a completeness proof for the resolution calculus by showing that it admits a so-called *reduction property for counterexamples*. We see how this completeness proof allows one to restrict the calculus and to formulate a notion of redundancy which potentially improves efficiency of the saturation procedure.

Remark 3.1. The resolution calculus $\mathcal{R}$ is a sound inference system. More generally, the conclusion of every inference rule is a logical consequences of its premises. Indeed, since all variables in clauses are implicitly universally quantified, every instance of a clause is a logical consequence of this clause. Hence, for the Resolution rule we have: $C \vee A \vDash C\sigma \vee A\sigma$, $D \vee \neg B \vDash D\sigma \vee \neg B\sigma$. Since $A\sigma = B\sigma$, this gives us $C \vee A$, $D \vee \neg B \vDash (C\sigma \vee A\sigma) \wedge (D\sigma \vee \neg A\sigma) \equiv C\sigma \vee D\sigma$. For the Factoring rule, similarly $C \vee A \vee B \vDash C\sigma \vee A\sigma \vee B\sigma \equiv C\sigma \vee A\sigma$ since $A\sigma = B\sigma$. ⋘

Recall, that a calculus is refutationally complete if every unsatisfiable clause set has a refutation, that is a derivation of the empty clause. Dually, if there is no way to derive the empty clause from a clause set, then this clause set must be satisfiable. In other words, a calculus is complete if every clause set whose saturation does not contain the empty clause is satisfiable. Since the saturation of any clause set is a superset of this set, refutational completeness for the resolution calculus $\mathcal{R}$ can be equivalently formulated as follows:

Theorem 3.2 (Completeness of $\mathcal{R}$). *Let N be a clause set that is saturated in inference system of $\mathcal{R}$. Then N is satisfiable iff N does not contain the empty clause $\square$.*

3.1 Refutational Completeness for the Ground Version

In this section we prove completeness for the ground (also called propositional) version of resolution calculus $\mathcal{R}^0$, which is given in System 4. This calculus is

Resolution	**Factoring**
$\mathtt{R}: \dfrac{C \vee \underline{A} \quad D \vee \neg \underline{A}}{C \vee D}$	$\mathtt{F}: \dfrac{C \vee \underline{A} \vee \underline{A}}{C \vee A}$

System 4: The propositional resolution calculus $\mathcal{R}^0$

applied to a set of ground clauses Cl^0_Σ, thus unification is replaced here by identity checking (two unifiable ground expressions are equal). Unless stated otherwise, all clauses and expressions in this section are ground. The following theorem is a restriction of Theorem 3.2 to ground clauses:

Theorem 3.3 (Completeness of $\mathcal{R}^0$). *Let N be a set of ground clauses that is saturated w.r.t. $\mathcal{R}^0$. Then N is satisfiable iff N does not contain the empty clause $\Box$.*

Proof. The *"only if"* part of this theorem is trivial, since every clause set containing the empty clause is unsatisfiable. The non-trivial part of the theorem is the *"if"* direction.

Let N be a clause set saturated w.r.t. $\mathcal{R}^0$ and not containing the empty clause $\Box$. We assume this set to be fixed in this proof. We are going to construct a model for N inductively over subsets of N.

Let $\succ$ be an ordering on ground literals. We extend $\succ$ to an ordering on ground clauses by taking the multiset extension of $\succ$ (which shall be also denoted by $\succ$). Note that the empty clause $\Box$ is the least clause w.r.t. $\succ$. We introduce additional notation: $N_C := \{C' \in N \mid C' \prec C\}$, $N^C := \{C' \in N \mid C' \preceq C'\}$ denote the set of clauses from N that are smaller, respectively smaller or equal then the clause C.

We describe a procedure that assigns to every subset N_C and N^C of N a Herbrand model I_C and I^C respectively. The Herbrand models I_C and I^C are viewed as sets consisting of ground atoms (see Remark 2.5). The models are constructed by induction over $C \in \mathrm{Cl}^0_\Sigma$ w.r.t. the ordering $\succ$. In order to make induction work, this ordering should be well-founded. Since $\succ$ is initially defined on ground literals, we require that this ordering enjoys the following properties:

Definition 3.4 (Admissible Order). An ordering $\succ$ is *admissible for resolution* if:

(**W**) $\succ$ is a well-order on ground literals;
(**R1**) $\neg A \succ A$ for every ground atom A. ⋘

Admissible orders do exist. By the well-ordering theorem, every set can be well-ordered.[6] It suffices to take a well-order on all negative literals and a well-order on all positive literals and set every negative literal greater then every positive one.

Definition 3.5 (Productive Clause). A clause C is called *productive* w.r.t. an interpretation I if (***i***) $I \not\models C$ and (***ii***) $C = C' \vee A$, where A is a positive literal which is strictly greatest element w.r.t. C'. In this case we say that C *produces* the atom A w.r.t. I and we assign $\Delta^I C := \{A\}$. If C is not productive we assign $\Delta^I C := \{\}$. ⟡

Definition 3.6 (Candidate Models). Let $N \subseteq \mathrm{Cl}^0_\Sigma$ and $C \in \mathrm{Cl}^0_\Sigma$. The *candidate models* I_N for N, I_C for N_C and I^C for N^C are defined inductively as follows: $I_N := \bigcup_{C \in N} I^C$; $I_C := \bigcup_{C' \in N_C} I^{C'}$, and $I^C := I_C \cup \Delta^{I_C} C$. ⟡

Note that if C is a minimal clause w.r.t. N then $I_C = \{\}$ (the empty set is the neutral element w.r.t. the set union operation). Since $\succ$ is a well-founded ordering on Cl^0_Σ, the candidate model I_N is well-defined for every clause set N. So Definition 3.6 is correct.

We usually check productiveness of a clause C w.r.t. the model I_C. For this special case we shorten $\Delta^{I_C} C$ to ΔC and say that C *produces* ΔC. We use the notation $C = C' \vee \boldsymbol{A}^\star$ to indicate that $A \in \Delta C$.

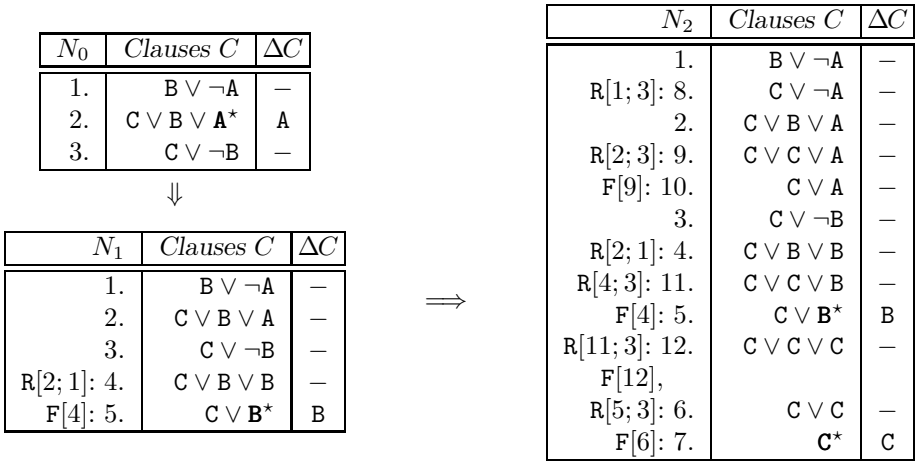
The candidate models are not necessarily models for respective clause sets (they are only "candidates" to be so). This shall be demonstrated in the next example. However, for saturated sets it is indeed the case, which we are going to show in the rest of the proof.

Example 3.7. Let us find the candidate model for the clause set N_0 in Figure 3. We fix an admissible ordering $\succ$ such that$\neg\mathtt{A} \succ \mathtt{A} \succ \neg\mathtt{B} \succ \mathtt{B} \succ \mathtt{C}$. So the clauses in this table are listed in ascending order w.r.t. $\succ$. Clause 3 is the least clause, hence $I_3 = \{\}$. Since $I_3 \models \neg\mathtt{B}$, then clause 3 is true in I_3. So it produces no atom, which is indicated in the last column of the table. Continuing, we have $I_2 = I^3 = I_3 \cup \Delta 3 = \{\}$. Clause 2 is false in I_2, so it produces its greatest atom $\mathtt{A}$. Finally, $I_1 = I^2 \cup I^3 = \{\mathtt{A}\}$. Although clause 1 is false in I_1, it may not be productive, since its greatest literal $\neg\mathtt{A}$ is negative. Therefore, no atom is produced. We have obtained a candidate model $I_{N_0} = I^1 \cup I^2 \cup I^3 = \{A\}$ for clause set N_0 which is not a model of N_0, since the clause 1 is false in I_{N_0}.

The reason why the candidate model for N_0 is not a model for N_0 is hidden in the "conflict" between clauses 1 and 2 which have complementary literals $\neg\mathtt{A}$

[6] This theorem is equivalent to Axiom of Choice. However for Herbrand universes over a finite signature, that are used for lifting of calculi, one can use *KBO* or *LPO*-orderings and not rely on Axiom of Choice

Figure 3 Candidate models for clause sets



and A respectively. Consider the clause set N_1, where we have added the resolvent R[2; 1]: 4 between these clauses and its factor F[4]: 5. In this new clause set, the clause 2 yielding a conflict before, is not productive anymore, since the clause 5 produces the atom B which makes clause 2 true in I_2. However again, the candidate model $I_{N_1} = \{\mathtt{B}\}$ for N_1 is not a model for N_1 (neither is for N_0) since clause 3 is false in I_{N_1}. The reason is again in that clauses 3 and 5 have complementary literals.

Finally, consider a clause set N_2 which is obtained from N_1 by producing all remaining resolution and factoring inferences between the clauses, i.e., N_2 is a saturation of N_0 w.r.t. $\mathcal{R}$. The candidate model $I_{N_2} = \{\mathtt{B}, \mathtt{C}\}$ for this clause set is indeed a model for N_2 and hence for N_0. ⋘

We confirm our conjecture and prove that the candidate model for every saturated clause set is a model for this set. This result is a consequence of the following two lemmas. The proofs of these lemmas are supplied with small tables similar to those given in Example 3.7. These tables demonstrate situations for different cases that we consider.

Lemma 3.8 (Properties of Candidate Models). *Let N be a clause set for which candidate models are constructed according to Definition 3.6. Then for every $C \in N$ (**i**) $I^C\Downarrow \models C$ implies that $I_N\Downarrow \models C$ and (**ii**) If $C = (C' \vee \boldsymbol{A}^\star)$ is productive w.r.t. I_C then $I_N\Downarrow \not\models C'$.*

Proof. (i) If $I^C\Downarrow \models A$ for some positive atom A in C, then $I_N\Downarrow \models A \models C$. Otherwise, there should be a clause $D = (D' \vee \boldsymbol{A}^\star) \succ C$ producing an atom A that occurs *negatively* in C. But this situation is not possible, since otherwise $C \succeq \neg A \succ A \succ D$ (here we have used condition (R1) for admissible orderings from Definition 3.4). This contradicts the assumption $D \succ C$.

N	*Clauses* C	ΔC
D	$D' \vee \boldsymbol{A}^\star$	A
Υ		
C	$C' \vee \neg A$	?

(ii) Assume that $I_N \models C'$. First, note that $I_C \not\models C'$ since C is a productive clause. Hence there should be a clause $C_1 \succeq C$ that produces some atom B which occurs positively in C'. But this situation is not possible, since, otherwise $B \succeq A \succ C' \succeq B$. ⧉

N	*Clauses* C	ΔC
C_1	$C_1' \vee \boldsymbol{B}^\star$	B
Υ	C'	Υ
C	$\overbrace{C'' \vee B} \vee \boldsymbol{A}^\star$	A

Note that Lemma 3.8 holds for arbitrary clause sets, not only for saturated ones. The fact that N is saturated is used in the next lemma which is the main ingredient of the model generation method. If I_N is not a model for a clause set N then there must be a *counterexample* for I_N, that is a clause $D \in N$ which is false in I_N. The lemma claims that if N is saturated, then any counterexample D for I_N, except for the empty clause $\square$, can be *reduced*, i.e., there must be an inference rule from D that produces a *smaller* counterexample $D_1 \in N$ than D.

Lemma 3.9 (Counterexample-Reduction Lemma). *Let N be a set of ground clauses which is saturated w.r.t. $\mathcal{ORS}^{0\succ}_{Sel}$. Then for every $D \in N$ with $I_N \not\models D \neq \square$ there exists $D_1 \in N$ such that $D_1 \prec D$ and $I_N \not\models D_1$.*

Proof. Let $I_N\Downarrow \not\models D$ for some clause $D \in N$. Then $I^D\Downarrow \not\models D$, since otherwise by Lemma 3.8 (i) we would have $I_N\Downarrow \models D$. In particular, clause D cannot be productive. So, we have $I_D\Downarrow = I^D\Downarrow \not\models D$. According to the definition of productive clauses (see Definition 3.5), this situation is possible only in the following cases: either (**1**) D has at least one negative literal and for every such literal $L = \neg A$ we have $I_D \models A$, or (**2**) the maximal literal in D is positive, but not strictly maximal (these cases are not mutually exclusive).

In case (1), $D = (D' \vee \neg\boldsymbol{A})$, where $I_D \models A$ so there exists a clause $C = (C' \vee \boldsymbol{A}^\star) \in N$ that produces the atom A. This means that the Resolution rule can be applied to the clauses C and D which yields the clause $\mathtt{R}[C; D]$: $D_1 = (C' \vee D') \in N$. The conclusion of this inference D_1 is smaller than the clause D, since $\neg A \succ A \succ C'$. Moreover, $I_N \not\models D_1$, since $I_N \not\models D'$ (from $I_N \not\models D$) and $I_N \not\models C'$ (by Lemma 3.8 (ii)). Thus, we have found a smaller counterexample D_1 for I_N.

N	*Clauses* C	ΔC
D	$D' \vee \neg\boldsymbol{A}$	$-$
Υ		
D_1	$C' \vee D'$	?
C	$C' \vee \boldsymbol{A}^\star$	A

In case (2), $D = (D' \vee A \vee \boldsymbol{A})$, where A is the greatest literal in D. In this case, the Factoring rule can be applied to D which produces the clause $\mathtt{F}[D]$: $D_1 = (D' \vee A) \in N$. Obviously, $D \succ D_1$ and $I_N \not\models D_1$, since $I_N \not\models D$. So D_1 is a smaller counterexample for I_N than D.

N	*Clauses* C	ΔC
D	$D' \vee A \vee \boldsymbol{A}$	$-$
γ		
D_1	$D' \vee A$	?

The proof of Theorem 3.3 now can be concluded as follows. Suppose I_N is not a model for N. Then the set $X = \{D \mid I_N \not\models D\}$ of counterexamples for I_N is not empty. Since $\succ$ is a well-order on N, there is a least element $D_0 \in X$ w.r.t. $\succ$ (i.e., $D \succeq D_0$ for every $D \in X$). But this is not possible, since by Lemma 3.9 there exists a smaller counterexample than D_0. Hence I_N is a model for N and N is a satisfiable clause set.

Theorem 3.3 has the following rather unexpected consequence:

Corollary 3.10 (Compactness for Propositional Clause Logic). *Let N_0 be a set of ground clauses. Then N_0 is unsatisfiable iff some finite subset of N_0 is unsatisfiable.*

Proof. The *"if"* part of the corollary is trivial. To prove the *"only if"* part, assume that N_0 is an unsatisfiable clause set. Let N be the saturation of N_0 under $\mathcal{R}^0$, i.e., $N := \mathcal{R}^{0*}(N_0)$. By Theorem 3.3, N must contain the empty clause $\Box$. This clause is derivable from a finite subset of N_0 (see Remark 2.41) using a sound inference system. Therefore this *finite* subset of clauses is unsatisfiable.

3.2 Refinements of the Resolution Calculus

Let us analyze the proof of Theorem 3.3, in particular, arguments used in Counterexample -Reduction Lemma (Lemma 3.9). Note that in this proof we have used that the set N is closed only w.r.t. restricted forms of the Resolution and Factoring inferences:

- the resolved atom in the left premise of the Resolution rule is a strictly maximal in this clause and
- the factored atom in the premise of the Factoring rule is maximal in this clause.

Moreover, note that cases (1) and (2) in the proof of Lemma 3.9 are not mutually exclusive, and any negative literal $\neg A$ of the clause D can be chosen in case (1) to reduce the counterexample. This freedom can be restricted using so-called *selection strategies*.

Definition 3.11 (Selection Function, Eligible Literals). A *selection function* is a mapping *Sel* that assigns to every clause C a (possibly empty) sub-multiset $Sel(C)$ of negative literals from C. These literals are then called the *selected literals* in C. A literal L is *maximal* w.r.t. C, if $L' \succ L$ for no literal $L' \in C$. Additionally, if $L \notin C$, then L is *strictly maximal w.r.t.* C. A literal L is called *eligible w.r.t.* C (or *eligible in* $C \vee L$) if either $L \in Sel(C \vee L)$, or otherwise $Sel(C \vee L) = \{\}$ and L is maximal w.r.t. C. ⋘

It can be shown that applications of the Resolution and Factoring rules can be restricted to eligible literals w.r.t. any selection function *Sel*. The resulted restricted inference system is called *Ordered Resolution with Selection*. The ground version of this calculus $\mathcal{OR}^{0\succ}_{Sel}$ is formulated in System 5. The *Ordered Resolution Calculus* $\mathcal{OR}^{0\succ}$ is an instance of $\mathcal{OR}^{0\succ}_{Sel}$ for the *trivial* selection function $Sel = Sel_0$, that assigns the empty multiset of literals to every clause.

Ordered Resolution

$$\mathtt{OR}: \frac{C \vee \underline{\boldsymbol{A}}^{\star} \quad D \vee \neg\underline{\boldsymbol{A}}}{C \vee D}$$

where (i) A is eligible strictly maximal w.r.t. C and (ii) $\neg A$ is eligible w.r.t. D.

Ordered Factoring

$$\mathtt{OF}: \frac{C \vee \underline{A} \vee \underline{\boldsymbol{A}}}{C \vee A}$$

where (i) A is eligible w.r.t. C.

System 5: The propositional version of the ordered resolution calculus with selection $\mathcal{OR}^{0\succ}_{Sel}$

Remark 3.12. In System 5 we have used a new notation. When formulating inference rules, we indicate eligible literals in clauses by writing them in boldface type. That does *not* mean that these are the *only* eligible literals in clauses. For instance, both literals A in the premise of the Ordered Factoring rule are eligible. Additionally, by writing $C = C' \vee \boldsymbol{A}^{\star}$, we indicate that the eligible atom A is *strictly maximal* in clause C. A clause C of this form is called *reductive (for A)*. Intuitively C is a candidate for a productive clause in model construction. Given this notation, we will not duplicate its meaning in the conditions of rules starting from now on. ⋘

The inference system $\mathcal{OR}^{0\succ}_{Sel}$, has an additional property that did not have the system $\mathcal{R}^0$, namely monotonicity. We say that an inference π is *monotone (w.r.t. $\succ$)*, if some premise C' of π is greater then the conclusion C of π: $C' \succ C$. An inference system $\boldsymbol{\mathcal{S}}$ is *monotone* (w.r.t. $\succ$), if every inference $\pi \in \boldsymbol{\mathcal{S}}$ is monotone w.r.t. $\succ$. It can be shown that $\mathcal{OR}^{0\succ}_{Sel}$ is monotone:

Lemma 3.13 (Monotonicity for $\mathcal{OR}^{0\succ}_{\boldsymbol{Sel}}$). *$\mathcal{OR}^{0\succ}_{Sel}$ is monotone w.r.t. every admissible ordering $\succ$.*

Proof. We must show that the conclusion of the Ordered Resolution and Ordered Factoring rules from ground clauses are smaller then some premise of these rules:

(**1**) $\mathtt{OR}: C \vee \underline{\boldsymbol{A}}^{\star},\ D \vee \neg\underline{\boldsymbol{A}} \vdash C \vee D$ (Ordered Resolution):
We show that the conclusion of the Ordered Resolution rule is always smaller than its right premise. Indeed, by condition (R1) of admissible ordering (see Definition 3.4), we must have $\neg A \succ A$. Additionally, by condition (i) of this inference rule, we have $A \succ C$. This implies $(D \vee \neg A) \succ (C \vee D)$, what was required to show.

(**2**) $\mathtt{OF}: C \vee \underline{A} \vee \underline{\boldsymbol{A}} \vdash C \vee A$ (Ordered Factoring):
Monotonicity of the Ordered Factoring rule is obvious, since $\succ$ is monotone w.r.t. multiset inclusion of clauses: $(C \vee A \vee A) \succ (C \vee A)$. ⧉

Theorem 3.14 (Completeness of $\mathcal{OR}^{0\succ}_{\boldsymbol{Sel}}$). *Let N be a set of ground clauses that is saturated in the inference system $\mathcal{OR}^{0\succ}_{Sel}$ based on an admissible ordering $\succ$ and a selection function Sel. Then N is satisfiable iff N does not contain the empty clause $\square$.*

Proof. The proof of this theorem goes in the same way as for Theorem 3.3, except for a couple of moments below.

($\boldsymbol{a}$) We need to modify the notion of a productive clause (see Definition 3.5). A clause $C \in N$ is *productive* w.r.t. an interpretation I if ($\boldsymbol{i}$) $I \not\models C$ and ($\boldsymbol{ii}$) $C = C' \vee \boldsymbol{A}^{\star}$, i.e., A is an eligible literal which is $\succ$-strictly greatest w.r.t. C'. In this case, C *produces* A.

($\boldsymbol{b}$) If $I \not\models D$ but D is not productive w.r.t. I, then this is only possible if either D is the empty clause, or (1) $D = D' \vee \neg\boldsymbol{A}$ (i.e., $\neg A$ is eligible in D) for some A such that $I \models A$, or there are no negative eligible literals and hence (2) $D = D' \vee \boldsymbol{A} \vee A$ (i.e., A is eligible in D but not strictly maximal). The proof of Counterexample-Reduction Lemma (Lemma 3.9). must be adjusted accordingly for these cases. In both cases either the Ordered Resolution rule or the Ordered Factoring rule can be applied which reduces a counterexample. ⧉

In fact, ordered resolution is complete for a larger class of orderings than orderings admissible according to Definition 3.4. Condition (W) (well-foundedness and totality) can be dropped without affecting refutational completeness. The resulted class of orderings is called *A-orderings* [Kowalski & Hayes, 1968], [see also Chapter 5 in Fermüller et al., 1993]. We define an extension of A-orderings for ground literals called L-orderings:

Definition 3.15 (L-ordering). An ordering $\succ$ on ground literals is called *L-ordering* if:

(**R1**) $\neg A \succ A$ for every ground atom A. ⟐

Corollary 3.16 (Completeness of $\mathcal{OR}^{0\succ}_{Sel}$ for L-orderings). *Let N be a set of ground clauses that is saturated in $\mathcal{OR}^{0\succ}_{Sel}$ based on a selection function Sel and an L-ordering $\succ$. Then N is satisfiable iff N does not contain the empty clause $\Box$.*

Proof. Again, the *"if"* direction is the only non-trivial part of this corollary. In order to show refutational completeness, suppose N is an unsatisfiable clause set. Then by compactness of propositional clause logic (Corollary 3.10), there is a finite subset N_0 of N that is unsatisfiable. Let $\succ_{|N_0}$ be the restriction of the ordering $\succ$ on literals from N_0 and let $\succ_0$ be a total well-founded extension of $\succ_{|N_0}$ such that $\neg A \succ_0 A$ for every atom A. Such an extension always exists since N_0 contains only finitely many atoms. Hence $\succ_0$ is an admissible ordering according to Definition 3.4.

Let N' be the saturation of N_0 under $\mathcal{OR}^{0\succ}_{Sel}$ based on the ordering $\succ_0$. Then N' is a subset of N since all inferences involving literals from N_0 that are possible in w.r.t. $\succ_0$ are also possible w.r.t. $\succ$. Since N' is unsatisfiable (because N_0 is unsatisfiable), by Theorem 3.14 applied to $\mathcal{OR}^{0\succ}_{Sel}$ based on $\succ_0$, we conclude that N' and hence N contain the empty clause. ⧉

There is a freedom left in what eligible negative literal to choose in case (1) of Counterexample-Reduction Lemma if there are several of those. Instead of choosing one negative eligible literal, one can apply an inference on all of them simultaneously to produce a smaller counterexample. This justifies an extension of the Ordered Resolution rule called the Ordered Hyper-resolution rule. For the purpose of this rule, we say that a multiset $D' = \{\neg B_1,..,\neg B_n\}_m$ of negative literals is *eligible w.r.t. D (for hyper-resolution)* if either (1) $D' = Sel(D \vee D') \neq \{\}_m$, or otherwise (2) $Sel(D \vee D') = \{\}_m$, $n = 1$ and $\neg B_1$ is maximal w.r.t. D. The Ordered Hyper-resolution rule is given in Figure 4. This inference rule can be simulated

Figure 4 The hyper-resolution rule for ground clauses

Ordered Hyper-resolution

$$\mathtt{HR}: \frac{C_1 \vee \underline{\underline{A_1}}^{\star} \quad \ldots \quad C_n \vee \underline{\underline{A_n}}^{\star} \qquad \neg\underline{\underline{A_1}} \vee \cdots \vee \neg\underline{\underline{A_n}} \vee D}{C_1 \vee \cdots \vee D}$$

where (i) A_i are eligible strictly maximal literals w.r.t. C_i, $i = 1,..,n$, and (ii) $\{\neg A_1,..,\neg A_n\}_m$ is eligible w.r.t. D.

by an application of n ordered resolution inferences. However, the advantage of a simultaneous inference is that the intermediate clauses are not retained in the clause set.

3.3 Redundancy: the Static View

Ordering restrictions, selection functions and hyper-inferences are especially useful for theorem proving since in practice they lead to a huge decrease in the number of generated clauses. If we reconsider the inferences in the set N_2 from Example 3.7, we notice that most of the inferences violate the *ordering restrictions* in $\mathcal{OR}^{0\succ}$ and therefore can be eliminated. Indeed, the clauses $8-12$ can be deleted from the set N_2 without any change in candidate models.

The inferences producing the clauses $8-12$ are not needed because they are not used for reduction of counterexamples. This observation is extended to the general notion of redundancy introduced by Bachmair & Ganzinger [1990, 1994]. The idea is to identify clauses that may never be minimal counterexamples, and inferences that may be not used for reduction of counterexamples. These are so-called redundant clauses and redundant inferences.

Definition 3.17 (Standard Redundancy for the Ground Case). Let $\succ$ be a total order on ground clauses, N be a subset of ground clauses and $N_C := \{C' \mid C' \prec C\}$ be defined like in the proof of Theorem 3.2. A ground clause $C \in \mathrm{Cl}^0_\Sigma$ is called *redundant w.r.t. N* if $N_C \models C$. A ground *inference* $\pi \in \mathrm{Inf}^0_\Sigma$ with the maximal premise C' and the conclusion C is *redundant w.r.t. N* in an inference system $\boldsymbol{\mathcal{S}}$, if $\pi \in \boldsymbol{\mathcal{S}}$ implies that $N_{C'} \models C$. ⋘

In words, a ground clause C is redundant if it follows logically from some smaller clauses from N. An inference $\pi \in \boldsymbol{\mathcal{S}}$ is redundant if its conclusion follows from clauses in N that are smaller than its maximal premise. Note that it is neither required that $C \in N$, nor that the inference π is applicable to the clause set N. Note also that every inference outside $\boldsymbol{\mathcal{S}}$ is redundant (these inferences are not needed for refutation). Definition 3.17 in particular implies that tautologies are redundant w.r.t. every clause set (since they follow from no clauses) and monotone inferences whose conclusion is in N or is redundant w.r.t. N, are redundant.

Continuing Example 3.7, we can notice that the clauses 2 and 4 are redundant w.r.t. the clause sets N_1 and N_2 since they follow form the smaller clause 5. The inference $\mathtt{R}[1,3]$ is redundant w.r.t. both N_1 and N_2 in $\boldsymbol{\mathcal{S}} := \mathcal{R}$, since its conclusion (the clause 8) follows logically from the smaller clauses 3 and 5.

Remark 3.18. A clause $C \in N$ that is redundant w.r.t. N may not be a minimal counterexample w.r.t. I_N, even if N is *not* saturated, since otherwise $I_N \not\models N_C$, so there must be some $C' \in N_C$ such that $I_N \not\models C'$ which is then a smaller counterexample than C. Similarly, a redundant inference may be not used for reducing counterexamples in N: If a counterexample C' may be reduced to C via a redundant inference $C_1,..,C_k \vdash C$, where C' is maximal in $\{C_1,..,C_k\}$, then $C' \succ C$, $N_{C'} \models C$, and hence there is a counterexample in $N_{C'}$ that is smaller than C'. So, the counterexample C' can be reduced in N without having C in N. ⋘

Redundancy notions allow one to weaken the conditions of Theorem 3.14, so that the conclusion of this theorem still holds. We say that a clause set N is *saturated up to redundancy* w.r.t. an inference system $\boldsymbol{\mathcal{S}}$, if every $\boldsymbol{\mathcal{S}}$-inference $\pi = C_1,..,C_k \vdash C$ from N (i.e. when $\pi \in \boldsymbol{\mathcal{S}}$ and $C_i \in N$, $i = 1,..,k$) is redundant w.r.t. N. The following is a stronger version of Theorem 3.14:

Theorem 3.19 (Completeness for $\mathcal{OR}^{0\succ}_{\boldsymbol{Sel}}$-Saturated Sets up to Redundancy).
Let N be a set of ground clauses that is saturated up to redundancy in $\mathcal{OR}^{0\succ}_{Sel}$ based on an admissible ordering $\succ$ and a selection function Sel. Then N is satisfiable iff N does not contain the empty clause $\Box$.

Proof. The proof of this theorem proceeds in the same way as for Theorem 3.14. The only modification that has to be done, is the case in Counterexample-Reduction Lemma, when an inference reducing a counterexample is possible but has not been made because of redundancy. In this case one applies the argument given in Remark 3.18 to find a smaller counterexample. ⧉

3.4 Redundancy: the Dynamic View

In order to use Theorem 3.19 effectively, one should come up with a *derivation strategy* such that for every clause set N either (***i***) the empty clause is eventually derived and hence N is unsatisfiable, or otherwise, (***ii***) the process can be continued (forever) and a saturated set is obtained ("in the limit"). Modern saturation-based theorem provers implement a combination of two interleaving processes:

1. **Deduction**, *during which conclusions of (non-redundant) inferences are produced and*
2. **Deletion**, *during which unnecessary (redundant) clauses are removed from a clause set.*

The precise definition of this process can be given using an abstract notion of redundancy:

Definition 3.20 (Redundancy Criterion).
An *(abstract) redundancy criterion* is a pair $\mathbf{R} = (\mathsf{R}_{\mathrm{Cl}}(\cdot), \mathsf{R}_{\mathrm{Inf}}(\cdot))$ of functions, that assign for every clause set $N \subseteq \mathrm{Cl}_\Sigma$, a set $\mathsf{R}_{\mathrm{Cl}}(N) \subseteq \mathrm{Cl}_\Sigma$ of $\mathbf{R}$-*redundant clauses* w.r.t. N and a set $\mathsf{R}_{\mathrm{Inf}}(N) \subseteq \mathrm{Inf}_\Sigma$ of $\mathbf{R}$-*redundant inferences* w.r.t. N, such that for all sets of clauses N' and N with $N \setminus N' \subseteq \mathsf{R}_{\mathrm{Cl}}(N)$ the following properties hold:

- **(R1)** if N is unsatisfiable, then N' is unsatisfiable;
- **(R2)** $\mathsf{R}_{\mathrm{Cl}}(N) \subseteq \mathsf{R}_{\mathrm{Cl}}(N')$;
- **(R3)** $\mathsf{R}_{\mathrm{Inf}}(N) \subseteq \mathsf{R}_{\mathrm{Inf}}(N')$;

A redundancy criterion $\mathbf{R}$ is *effective*, if in addition:

(R4) for every inference $\pi \in \mathrm{Inf}_\Sigma$ such that $\pi(N) \subseteq N$, we have $\pi \in \mathsf{R}_{\mathrm{Inf}}(N)$. ◈

Conditions (R1) – (R4) for a redundancy criterion $\mathbf{R}$ have the following meaning: If a clause set N' is obtained from a clause set N by removing some redundant clauses and adding some other clauses, then this transformation preserves (R1): unsatisfiability of clause sets as well (R2): redundancy of clauses and (R3): redundancy of inferences. Finally, condition (R4) implies that an inference becomes redundant as long as its conclusion has been added to a clause set.

The *trivial* but effective redundancy criterion is the "smallest" $\mathsf{R}^0 = (\mathsf{R}^0_{\mathrm{Cl}}(\cdot), \mathsf{R}^0_{\mathrm{Inf}}(\cdot))$ that admits all properties (R1) – (R4), which can be defined as follows: $\mathsf{R}^0_{\mathrm{Cl}}(N) := \{\}$; $\mathsf{R}^0_{\mathrm{Inf}}(N) := \{(C_1,..,C_k \vdash C) \in \mathrm{Inf}_\Sigma \mid C \in N\}$.

The redundancy criterion given in Definition 3.17 is called the *standard redundancy criterion (for ground clauses)* $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{gr}} = (\mathsf{R}^{\succ}_{\mathrm{Cl}^0}(\cdot), \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}^0}(\cdot))$ w.r.t. an inference system $\boldsymbol{\mathcal{S}}$ parametrized by an ordering $\succ$. When we speak about redundancy without mentioning $\mathbf{R}$, we usually refer to this notion. We show that this redundancy criterion admits all properties of Definition 3.20.

Lemma 3.21 (Standard Redundancy Criterion). *The notion of standard redundancy* $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{gr}} = (\mathsf{R}^{\succ}_{\mathrm{Cl}^0}(\cdot), \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}^0}(\cdot))$ *is a redundancy criterion for ground clauses. If in addition the inference system* $\boldsymbol{\mathcal{S}}$ *is monotone, then* $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{gr}}$ *is effective.*

Proof. For showing the properties (R1) – (R3) for $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{gr}}$, it suffices to prove that for every sets N, N' of ground clauses and a ground clause C we have that $N \setminus N' \subseteq \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N)$ implies $N' \models N$ and $N'_C \models N_C$. Indeed, $N' \models N$ guarantees the property (R1). Since $C \in \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N)$ *iff* $N_C \models C$, we have that $N'_C \models N_C$ implies (R2): $\mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N) \subseteq \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N')$. Similarly, $\pi \in \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}^0}(N)$ *iff* $N_{C'} \models C$ for the maximal premise C' of $\pi \in \boldsymbol{\mathcal{S}}$, so $N'_C \models N_C$ implies (R3): $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}^0}(N) \subseteq \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}^0}(N')$.

Note that $N_C \setminus N'_C \subseteq \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N) \cap N_C \subseteq \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N_C)$, so it suffices to show that $N \setminus N' \subseteq \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N)$ implies $N' \models N$. This can be done as follows. Suppose this property does not hold and let C be a minimal w.r.t. $\succ$ clause in N such that $N' \not\models C$, so $N' \models N_C$. Then $C \in N \setminus N' \subseteq \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N)$, hence $N_C \models C$ and we have obtained a contradiction $N' \models N_c \models C$ with $N' \not\models C$.

To show (R4), let $\pi \in \boldsymbol{\mathcal{S}}$. Since $\boldsymbol{\mathcal{S}}$ is monotone, we have $C' \succ C$ for some premise C' and the conclusion C of π. If $C \in N$ or $C \in \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N)$ then $N_{C'} \models N^C \models C$, and so $\pi \in \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}^0}(N)$. ⧉

Remark 3.22. The proof of Lemma 3.21 uses the fact that the ordering $\succ$ is well-founded, when we claim existence of a minimal w.r.t. $\succ$ clause C satisfying certain

properties. Actually, this lemma does not hold for general L-orderings. For example, in the clause set $N = \{\mathtt{A}_1, \neg\mathtt{A}_1, \mathtt{A}_2, \neg\mathtt{A}_2, \ldots, \mathtt{A}_i, \neg\mathtt{A}_i, \ldots\}$ with $\neg\mathtt{A}_1 \succ \mathtt{A}_1 \succ \neg\mathtt{A}_2 \succ \mathtt{A}_2 \succ \cdots$, all clauses are redundant w.r.t. N (i.e., $N \subseteq \mathsf{R}_{\mathrm{Cl}}(N)$). Indeed, every clause $\mathtt{A}_i$ or $\neg\mathtt{A}_i$ with $i \geq 1$, follows from the pair of *smaller* clauses $\mathtt{A}_j, \neg\mathtt{A}_j$ with $j = i + 1$. So, $N \setminus \{\} \subseteq \mathsf{R}_{\mathrm{Cl}}(N)$ but $\{\} \not\models N$, since N is inconsistent. This example shows that the notion of standard redundancy is not fully compatible with L-orderings. However, some redundancy elimination techniques, in particular, *tautology deletion* can be justified also for L-orderings [see Kowalski & Hayes, 1968]. ◈

An abstract model of a saturation procedure can be defined using the binary relation $\Rightarrow$ on clause sets called the *(theorem-proving) derivation relation* $\Rightarrow$, that is defined in Figure 5 using two rules. The Deduction rule allows one to extend

Figure 5 Theorem-proving derivations "$\Rightarrow$" for $\boldsymbol{\mathcal{S}}$ and $\mathbf{R}$

Deduction
$N \;\Rightarrow\; N \cup N' \quad \text{if } N' \subseteq [\boldsymbol{\mathcal{S}} \setminus \mathsf{R}_{\mathrm{Inf}}(N)](N)$

Deletion
$N \;\Rightarrow\; N \setminus N' \quad \text{if } N' \subseteq \mathsf{R}_{\mathrm{Cl}}(N)$

a clause set by adding conclusions of non-$\mathbf{R}$-redundant $\boldsymbol{\mathcal{S}}$-inferences. Using the Deletion rule a clause set can be filtered by removing $\mathbf{R}$-redundant clauses. A (finite or countably infinite) sequence $N_0 \Rightarrow N_1 \Rightarrow N_2 \Rightarrow \cdots$ is called a *(theorem-proving) derivation* from N_0 (based on $\boldsymbol{\mathcal{S}}$ and $\mathbf{R}$). The set $N_\infty := \bigcup_{i\geq 0}\bigcap_{j\geq i} N_j$ of all *persisting clauses* is called the *limit of* the derivation. Sometimes we say that N_∞ is an $(\boldsymbol{\mathcal{S}}, \mathbf{R})$-*saturation* of N_0 for the reasons that will be clear in a moment.

Lemma 3.23 (Properties of the Limit Set). *Let $N_0 \Rightarrow N_1 \Rightarrow \cdots$ be a derivation based on $\boldsymbol{\mathcal{S}}$ and $\mathbf{R}$. Then (**i**) if N_0 is unsatisfiable then N_∞ is unsatisfiable; (**ii**) $\mathsf{R}_{\mathrm{Cl}}(\bigcup_{i\geq 0} N_i) \subseteq \mathsf{R}_{\mathrm{Cl}}(N_\infty)$ and (**iii**) $\mathsf{R}_{\mathrm{Inf}}(\bigcup_{i\geq 0} N_i) \subseteq \mathsf{R}_{\mathrm{Inf}}(N_\infty)$.*

Proof. Let $N' := (\bigcup_{i\geq 0} N_i)$. For every clause $C \in N' \setminus N_\infty$ there exists some $i \geq 0$ such that $C \in N_i \setminus N_{i+1} \subseteq \mathsf{R}_{\mathrm{Cl}}(N_i)$. Indeed, otherwise either $C \notin N'$, or $C \in \bigcap_{i\geq j} N_j \in N_\infty$ for some $j \geq 0$. By condition (R2) of redundancy criterion (see Definition 3.20), $C \in N_i \setminus N_{i+1} \subseteq \mathsf{R}_{\mathrm{Cl}}(N_i) \subseteq \mathsf{R}_{\mathrm{Cl}}(N')$ and so $N' \setminus N_\infty \subseteq \mathsf{R}_{\mathrm{Cl}}(N')$.

Now claim (**i**) of the lemma follows from condition (R1) of redundancy criterion, since $N_0 \setminus N_\infty \subseteq N' \setminus N_\infty \subseteq \mathsf{R}_{\mathrm{Cl}}(N')$. Claims (**ii**) and (**iii**) follow respectively from conditions (R2) and (R3) of redundancy criterion. ⧉

Definition 3.24 (Fair Derivation). A theorem-proving derivation $N_0 \Rightarrow N_1 \Rightarrow \cdots$ based on $\boldsymbol{\mathcal{S}}$ and $\mathbf{R}$ is called *fair*saturation, if for every inference $\pi \in \boldsymbol{\mathcal{S}}$ from N_∞ (i.e., when $\pi(N_\infty) \neq \{\}$), there exists some $j \geq 0$ such that π is redundant w.r.t. N_j. ◈

In other words, a derivation is fair if every inference that can be applied to all clause sets N_j, $j \geq i$ starting from some $i \geq 0$, must become redundant eventually (for instance, by adding the conclusion of the inference to a clause set). The following lemma claims that a fair derivation from every clause set exists for every *effective* redundancy criterion.

Proposition 3.25 (Existence of a Fair Derivation). *Let $\boldsymbol{\mathcal{S}}$ be an inference system and $\mathbf{R}$ be an effective redundancy criterion. Then from every clause set N_0 there exists a fair derivation based on $\boldsymbol{\mathcal{S}}$ and $\mathbf{R}$.*

Proof. Consider the derivation $N_0 \Rightarrow N_1 \Rightarrow \cdots \Rightarrow N_i \Rightarrow \cdots$, where $N_{i+1} = N_i \cup \boldsymbol{\mathcal{S}}(N_i)$, $i \geq 0$. We claim that this is a fair derivation. Indeed, otherwise there exists an inference $\pi \in \boldsymbol{\mathcal{S}}$ and an infinite sequence $N_j \Rightarrow N_{j+1} \Rightarrow \cdots$ for some $j \geq 0$ such that for all $i \geq j$ we have $\pi(N_i) \neq \{\}$ and $\pi \notin \mathsf{R}_{\mathrm{Inf}}(N_i)$. But this is not possible since, in particular, $\pi(N_j) \subseteq N_{j+1}$ and by condition (R4) of effective redundancy criterion (see Definition 3.20), $\pi \in \mathsf{R}_{\mathrm{Inf}}(N_{j+1})$. ⧉

The notion of a saturated set up to redundancy can be extended to arbitrary redundancy criteria. Using this definition one can show that the limit of every theorem proving derivation is a saturated set up to redundancy:

Definition 3.26 (Saturation up to Redundancy, Completeness). A set of clauses N is called *saturated up to redundancy* w.r.t. an inference system $\boldsymbol{\mathcal{S}}$ and a redundancy criterion $\mathbf{R}$ (or, shortly $(\boldsymbol{\mathcal{S}}, \mathbf{R})$-*saturated*), if every $\boldsymbol{\mathcal{S}}$-inference π from N (i.e., when $\pi \in \boldsymbol{\mathcal{S}}$ and $\pi(N) \neq \{\}$) is redundant w.r.t. N: $\pi \in \mathsf{R}_{\mathrm{Inf}}(N)$.

An inference system $\boldsymbol{\mathcal{S}}$ with a redundancy criterion $\mathbf{R}$ is *(refutationally) complete* (or short, $(\boldsymbol{\mathcal{S}}, \mathbf{R})$ *is complete*) *for a set* $\boldsymbol{\mathcal{N}} \subseteq \mathrm{Cl}_\Sigma$ of clauses, if every fair derivation based on $\boldsymbol{\mathcal{S}}$ and $\mathbf{R}$ from an unsatisfiable clause set $N_0 \subseteq \boldsymbol{\mathcal{N}}$ contains the empty clause $\square$. ⧈

Lemma 3.27 (Limit of a Fair Derivation). *Let $N_0 \Rightarrow N_1 \Rightarrow \cdots$ be a fair derivation based on $\boldsymbol{\mathcal{S}}$ and $\mathbf{R}$. Then the limit N_∞ of this derivation is $(\boldsymbol{\mathcal{S}}, \mathbf{R})$-saturated.*

Proof. Let π be an inference from N_∞ (i.e., $\pi(N_\infty) \neq \{\}$). We need to show that $\pi \in \mathsf{R}_{\mathrm{Inf}}(N_\infty)$. Since the derivation is fair, by Definition 3.24 there is some $j \geq 0$ such that $\pi \in \mathsf{R}_{\mathrm{Inf}}(N_j)$. By condition (R3) of redundancy criterion, $\mathsf{R}_{\mathrm{Inf}}(N_j) \subseteq \mathsf{R}_{\mathrm{Inf}}(\bigcup_{i \geq 0} N_i)$. By Lemma 3.23 (iii), $\mathsf{R}_{\mathrm{Inf}}(\bigcup_{i \geq 0} N_i) \subseteq \mathsf{R}_{\mathrm{Inf}}(N_\infty)$. So, we conclude that $\pi \in \mathsf{R}_{\mathrm{Inf}}(N_\infty)$. ⧉

Corollary 3.28 (Completeness Criterion). *Let $\boldsymbol{\mathcal{S}}$ be an inference system, $\mathbf{R}$ be a redundancy criterion and $\boldsymbol{\mathcal{N}} \subseteq \mathrm{Cl}_\Sigma$ be a clause set such that $\boldsymbol{\mathcal{S}}(\boldsymbol{\mathcal{N}}) \subseteq \boldsymbol{\mathcal{N}}$. Then $(\boldsymbol{\mathcal{S}}, \mathbf{R})$ is complete for $\boldsymbol{\mathcal{N}}$, iff every unsatisfiable $(\boldsymbol{\mathcal{S}}, \mathbf{R})$-saturated clause set $N \subseteq \boldsymbol{\mathcal{N}}$ contains the empty clause $\square$.*

Proof. The "*only if*" part of the corollary is simple: If $(\boldsymbol{\mathcal{S}}, \mathbf{R})$ is complete for $\boldsymbol{\mathcal{N}}$ and $N \subseteq \boldsymbol{\mathcal{N}}$ is an unsatisfiable $(\boldsymbol{\mathcal{S}}, \mathbf{R})$-saturated clause set, then the trivial derivation consisting only of N is fair: all $\boldsymbol{\mathcal{S}}$-inferences from N are $\mathbf{R}$-redundant w.r.t. N. Hence, from completeness of $(\boldsymbol{\mathcal{S}}, \mathbf{R})$, we obtain that N contains the empty clause $\Box$.

To prove the "*if*" part, suppose that $(\boldsymbol{\mathcal{S}}, \mathbf{R})$ is not complete. Then there exists a fair derivation $N_0 \Rightarrow N_1 \Rightarrow \cdots$ from an unsatisfiable clause set $N_0 \subseteq \boldsymbol{\mathcal{N}}$ based on $\boldsymbol{\mathcal{S}}$ and $\mathbf{R}$ such that $\Box \notin N_i$ for every $i \geq 0$. Note that for every $i \geq 0$, we have $N_i \subseteq \boldsymbol{\mathcal{N}}$. Hence the limit N_∞ of the derivation is a subset of $\boldsymbol{\mathcal{N}}$. By Lemma 3.23 (i), N_∞ is unsatisfiable, since N_0 is unsatisfiable. Moreover, $N_\infty \subseteq \bigcup_{i \geq 0} N_i$, hence $\Box \notin N_\infty$. But this is not possible, since by Lemma 3.27, N_∞ is $(\boldsymbol{\mathcal{S}}, \mathbf{R})$-saturated, and by our assumption, $\Box \in N_\infty$. ⧉

Remark 3.29. Note that the "*only if*" part of Corollary 3.28, holds without the assumption that $\boldsymbol{\mathcal{S}}(\boldsymbol{\mathcal{N}}) \subseteq \boldsymbol{\mathcal{N}}$. ⋈

Corollary 3.30 (Completeness for $\mathcal{OR}^{0\succ}_{\boldsymbol{Sel}}$ with Redundancy). $(\mathcal{OR}^{0\succ}_{Sel}, \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{gr}})$ *is complete for the set* Cl^0_Σ *of ground clauses.*

Proof. By Theorem 3.19, every $(\mathcal{OR}^{0\succ}_{Sel}, \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{gr}})$-saturated unsatisfiable subset $N \subseteq \mathrm{Cl}^0_\Sigma$ of ground clauses contains the empty clause $\Box$. Hence the corollary is a consequence of Corollary 3.28 because $\mathcal{OR}^{0\succ}_{Sel}(\mathrm{Cl}^0_\Sigma) \subseteq \mathrm{Cl}^0_\Sigma$. ⧉

Note 3.31. *Definitions of redundancy criteria and fair derivations vary in literature. Our definition of redundancy criteria is equivalent to the one given in [Bachmair & Ganzinger, 2001]. This definition can be weaken, in particular condition (R2) can be replaced with condition (R2)′: "*$N \subseteq N'$ *implies* $\mathsf{R}_{\mathrm{Cl}}(N) \subseteq \mathsf{R}_{\mathrm{Cl}}(N')$*", so that Lemma 3.27 still holds. For an even weaker definition of redundancy criterion see [Bachmair, Ganzinger & Waldmann, 1994].*

We have also strengthened the definition of fair derivation, compared to the one given in [Bachmair & Ganzinger, 2001; Bachmair et al., 1994], since it appears to be more handy for showing fairness of saturation strategies that we describe in section 7). In this form it is more close to the definition of fairness given in [Nieuwenhuis & Rubio, 2001]. The weakest, although completely useless definition would be to say that a derivation is fair if N_∞ *is* $(\boldsymbol{\mathcal{S}}, \mathbf{R})$*-saturated.* ⋈

A limit of a theorem proving derivation for a given clause set N_0 is not unique in general. Even a *finite* clause set may have several different *minimal* saturations w.r.t. the *standard* redundancy criterion, as demonstrates the example below. However for the trivial redundancy criterion R^0, the saturation is always unique and consists of all clauses $\boldsymbol{\mathcal{S}}^*(N_0)$ that are derivable from the initial clause set, as no deletion step takes place.

Example 3.32. Let $\succ$ be an admissible ordering such that $\neg\mathtt{A} \succ \mathtt{A} \succ \neg\mathtt{B} \succ \mathtt{B} \succ \neg\mathtt{C} \succ \mathtt{C}$. Consider the clause set $N_0 := \{\neg\mathtt{A}^\star,\ \mathtt{A}^\star \vee \mathtt{C},\ \mathtt{B} \vee \mathtt{B},\ \mathtt{B}^\star \vee \neg\mathtt{C}\}$. This clause set has two derivations based on $\mathcal{OR}^{0\succ}$ and $\mathsf{R}_{\mathrm{gr}}^{\mathcal{S}\succ}$ that produce different minimal saturated set up to redundancy given in Figure 6. The (non-redundant) inferences that can be applied to N_0 are $\mathtt{OR}[2;1]$ and $\mathtt{OF}[3]$. Depending on the order in which these inferences are applied we obtain two different derivations.

Figure 6 Computing saturated sets with redundancy

$$
\begin{array}{ll}
N_0: & \{\overset{①}{\neg\mathtt{A}^\star},\ \overset{②}{\mathtt{A}^\star \vee \mathtt{C}},\ \overset{③}{\mathtt{B} \vee \mathtt{B}},\ \overset{④}{\mathtt{B}^\star \vee \neg\mathtt{C}}\} \\
& \Downarrow \mathtt{OR}[2;1] \\
N_1: & \{\overset{①}{\neg\mathtt{A}^\star},\ \overset{②}{\mathtt{A}^\star \vee \mathtt{C}},\ \overset{③}{\mathtt{B} \vee \mathtt{B}},\ \overset{④}{\mathtt{B}^\star \vee \neg\mathtt{C}},\ \overset{⑤}{\mathtt{C}^\star}\} \\
& \Downarrow \mathit{DelRed} \\
N_2: & \{\overset{①}{\neg\mathtt{A}^\star},\ \overset{④}{\mathtt{B}^\star \vee \neg\mathtt{C}},\ \overset{⑤}{\mathtt{C}^\star}\}
\end{array}
\qquad
\begin{array}{ll}
N_0: & \{\overset{①}{\neg\mathtt{A}^\star},\ \overset{②}{\mathtt{A}^\star \vee \mathtt{C}},\ \overset{③}{\mathtt{B} \vee \mathtt{B}},\ \overset{④}{\mathtt{B}^\star \vee \neg\mathtt{C}}\} \\
& \Downarrow \mathtt{OF}[3] \\
N_1': & \{\overset{①}{\neg\mathtt{A}^\star},\ \overset{②}{\mathtt{A}^\star \vee \mathtt{C}},\ \overset{③}{\mathtt{B} \vee \mathtt{B}},\ \overset{④}{\mathtt{B}^\star \vee \neg\mathtt{C}},\ \overset{⑥}{\mathtt{B}^\star}\} \\
& \Downarrow \mathit{DelRed} \\
N_2': & \{\overset{①}{\neg\mathtt{A}^\star},\ \overset{②}{\mathtt{A}^\star \vee \mathtt{C}},\ \overset{⑥}{\mathtt{B}^\star}\} \\
& \Downarrow \mathtt{OR}[2;1] \\
N_3': & \{\overset{①}{\neg\mathtt{A}^\star},\ \overset{②}{\mathtt{A}^\star \vee \mathtt{C}},\ \overset{⑥}{\mathtt{B}^\star},\ \overset{⑤}{\mathtt{C}^\star}\} \\
& \Downarrow \mathit{DelRed} \\
N_4': & \{\overset{①}{\neg\mathtt{A}^\star},\ \overset{⑥}{\mathtt{B}^\star},\ \overset{⑤}{\mathtt{C}^\star}\}
\end{array}
$$

In the first derivation, we have applied inference $\mathtt{OR}[2;1]$ which produced clause 5. $\mathtt{C}^\star$. This clause renders clause 2 redundant since the last clause contains literal $\mathtt{C}$. Moreover, clause 3 and hence inference $\mathtt{OF}[3]$ become redundant w.r.t. N_1 since clause 3 follows from the *smaller* clauses 4 and 5. After we delete all redundant clauses in N_1 (2 and 3 are the only clauses in N_1 that are redundant w.r.t. N_1), we obtain a clause set N_2 which is saturated in $\mathcal{OR}^{0\succ}$ (up to redundancy), since no further inferences apply.

If we apply inference $\mathtt{OF}[3]$ first (see the right derivation in Figure 6), we obtain clause 6. $\mathtt{B}^\star$, which renders clauses 3 and 4 redundant. After the deletion step, inference $\mathtt{OR}[2;1]$ remains possible, and at the end, a different saturated set N_4' is produced. ◈

One can argue that the set N_4' is a "better" saturated set than the set N_2, since $N_2 \setminus N_4' \subseteq \mathsf{R}_{\mathrm{Cl}}(N_4')$ but not vice versa. Note that the clause 6 can be obtained from the clauses 4 and 5 by applying an (unordered) resolution inference. The result of this inference, i.e., the clause 6 makes the clause 4 redundant. This inference is an instance of a so-called Subsumption Resolution inference rule [see Bachmair & Ganzinger, 2001]. To allow such useful inferences, a calculus is usually enhanced with additional *simplification rules*. We return to this point in section 7.

3.5 Lifting

By Lifting Lemma (Lemma 2.42), every clause set N represents the *set of its ground instances* N^{gr} (here *represents* means *"is equisatisfiable with"*). Hence, for *lifting* a refutationally complete calculus from ground clauses it suffices to find a sound inference system for general clauses such that for every set N that is saturated w.r.t. this system, the set of its ground instances N^{gr} is saturated w.r.t. the ground calculus.

In this section we describe the lifted version $\mathcal{OR}^{\succ}$ of the ordered resolution calculus, and in the next section, we show how selection functions can be integrated into this model. To describe $\mathcal{OR}^{\succ}$, we need to "lift" the ordering restrictions from the ground calculus $\mathcal{OR}^{0\succ}$.

Definition 3.33 (Liftable Orders, Lifting). An order $\succ$ on expressions is called *liftable* if there exists an order $\succ_0$ on ground expressions such that $E_1 \succ E_2$ implies that $E_1\sigma \succeq_0 E_2\sigma$ for every ground substitution σ. The *lifting of* an order $\succ$ defined on ground expressions is a liftable order containing $\succ$ (which we denote by the same letter) that is defined as follows: $E_1 \succ E_2$ *iff* for every ground substitution σ it holds that $E_1\sigma \succ E_2\sigma$. ⋘

Note that the lifting of a total order on ground expressions may not be a total order: two unifiable different expressions (e.g., $\mathtt{p}(\mathtt{x},\mathtt{c})$ and $\mathtt{p}(\mathtt{c},\mathtt{x})$) may not be comparable by $\succ$, since they have a common instance. To give another example, consider an *LPO*-ordering based on the precedence $\mathtt{f} \gg \mathtt{p} \gg \mathtt{q} \gg \mathtt{c}$ (see Definition 2.19). The atoms $\mathtt{p}(\mathtt{x},\mathtt{x})$ and $\mathtt{q}(\mathtt{y})$ are not comparable by $\succ_{lpo}$, since $\mathtt{p}(\mathtt{c},\mathtt{c}) \succ_{lpo} \mathtt{q}(\mathtt{c})$, but $\mathtt{q}(\mathtt{f}(\mathtt{c})) \succ_{lpo} \mathtt{p}(\mathtt{c},\mathtt{c})$. On the other hand, $\mathtt{p}(\mathtt{x},\mathtt{y}) \succ_{lpo} \mathtt{q}(\mathtt{y})$ and $\mathtt{q}(\mathtt{f}(\mathtt{x})) \succ_{lpo} \mathtt{p}(\mathtt{x},\mathtt{x})$. From Example 2.22, it also follows that $\mathtt{q}(\mathtt{f}(x)) \succ_{kbo} \mathtt{p}(\mathtt{x},\mathtt{x})$ *never* holds.

An order $\succ$ on atoms is usually extended to literals by setting $\neg A \succ A \succ \neg B \succ B$ for all atoms $A \succ B$. By *LPO* (or *KBO*) order on literals we mean this extension of the respective lifted orders from ground clauses.

Definition 3.34 (Admissible Orders for Ordered Resolution). An order $\succ$ on literals is called *admissible for resolution*, if (***i***) $\succ$ is liftable and (***ii***) $\succ$ is admissible for ground literals according to Definition 3.4. ⋘

Note that by this definition, an admissible order should be total on ground literals. Recall, that a literal L is *maximal* w.r.t. a clause C, if $L' \succ L$ for no literal $L' \in C$. If in addition $L \notin C$, then L is called *strictly maximal* w.r.t. C. The lifted version of the *Ordered Resolution calculus* $\mathcal{OR}^{\succ}$ (without selection) is given in System 6. The inference rules are applied to clauses whose variables are *renamed apart*, so that they do not have variables in common. This can be always done, since all variables in clauses are implicitly universally quantified. The ordered

Ordered Resolution

$$\mathtt{OR}: \frac{C \vee \underline{\boldsymbol{A}}^{\star} \quad D \vee \neg\underline{\boldsymbol{B}}}{C\sigma \vee D\sigma}$$

[*where (i) $\sigma = mgu(A, B)$; (ii) $A\sigma$ is strictly maximal w.r.t. $C\sigma$ and (iii) $\neg B\sigma$ is maximal w.r.t. $D\sigma$*]

Ordered Factoring

$$\mathtt{OF}: \frac{C \vee \underline{B} \vee \underline{\boldsymbol{A}}}{C\sigma \vee A\sigma}$$

[*where (i) $\sigma = mgu(A, B)$; (ii) $A\sigma$ is maximal w.r.t. $C\sigma \vee B\sigma$.*]

System 6: The ordered resolution calculus $\mathcal{OR}^{\succ}$

resolution calculus $\mathcal{OR}^{\succ}$ defines a sound inference system, since it is a restriction of resolution calculus $\mathcal{R}$, which is sound (see Remark 3.1).

Inference rules of System 6 are organized in such a way that, if an ordered resolution inference is possible from some ground instances of clauses, then the inference should be possible from these (non-ground) clauses that captures the result of the ground inference (see Figure 7).

Figure 7 The lifting diagram for $\mathcal{OR}^{\succ}$

$$\begin{array}{lll} \mathcal{OR}^{\succ}: & C \vee \underline{\boldsymbol{A}}^{\star},\ D \vee \neg\underline{\boldsymbol{B}} \overset{\mathtt{OR}}{\dashrightarrow} C\sigma \vee D\sigma & C \vee \underline{B} \vee \underline{\boldsymbol{A}} \overset{\mathtt{OF}}{\dashrightarrow} C\sigma \vee A\sigma \\ & \Downarrow \mathrm{gr} \quad \vdots \mathrm{gr} & \Downarrow \mathrm{gr} \quad \vdots \mathrm{gr} \\ \mathcal{OR}^{0\succ}: & C^0 \vee \underline{\boldsymbol{A}}^{\mathbf{0}\star},\ D^0 \vee \neg\underline{\boldsymbol{A}}^{\mathbf{0}} \overset{\mathtt{OR}}{\Longrightarrow} C^0 \vee D^0 & C^0 \vee \underline{A}^0 \vee \underline{\boldsymbol{A}}^{\mathbf{0}} \overset{\mathtt{OF}}{\Longrightarrow} C^0 \vee A^0 \end{array}$$

The way how the ordering restrictions (ii) and (iii) are applied in the conditions of $\mathcal{OR}^{\succ}$-inference rules in System 6, is called *a-posteriori*, i.e., after an inference is made. This way the inferences are more restrictive than with *a-priori* ordering restrictions, i.e., for $\sigma = id$. The a-priori ordering restrictions for the inference rules of $\mathcal{OR}^{\succ}$ are indicated by our notation for (strictly maximal) eligible literals (see Remark 3.12).

Remark 3.35. Note that even a-posteriori ordering restrictions do not guarantee that there is always a ground instance of the respective inference that satisfies the ordering restrictions. For example, a simple inference:

$$\mathtt{OR}: \frac{\mathtt{a}(\mathtt{x},\mathtt{x}) \vee \underline{\mathtt{c}(\mathtt{x},\mathtt{y})^{\star}} \quad \neg\underline{\mathtt{c}(\mathtt{u},\mathtt{v})} \vee \neg\mathtt{b}(\mathtt{v},\mathtt{v})}{\mathtt{a}(\mathtt{x},\mathtt{x}) \vee \neg\mathtt{b}(\mathtt{y},\mathtt{y})} \tag{15}$$

with the unifier $\sigma := \{\mathtt{u}/\mathtt{x}, \mathtt{v}/\mathtt{y}\}$, satisfies all a-posteriori (and a-priori) ordering restrictions of the ordered resolution rule for *KBO*-ordering $\succ_{kbo}$ based on a precedence with $\mathtt{a} \gg \mathtt{b} \gg \mathtt{c}$. However, for any inference from ground instances of the premises of (15):

$$\mathtt{OR}: \frac{\mathtt{a}(s^0, s^0) \vee \underline{\mathtt{c}(\boldsymbol{s}^{\mathbf{0}}, \boldsymbol{t}^{\mathbf{0}})^{\star}} \quad \neg\underline{\mathtt{c}(\boldsymbol{s}^{\mathbf{0}}, \boldsymbol{t}^{\mathbf{0}})} \vee \neg\mathtt{b}(t^0, t^0)}{\mathtt{a}(s^0, s^0) \vee \neg\mathtt{b}(t^0, t^0)} \tag{16}$$

we must have $\mathit{weight}(t^0) > \mathit{weight}(s^0)$ from the ordering restrictions for the left premise, and $\mathit{weight}(s^0) > \mathit{weight}(t^0)\mathit{weight}$ from the restrictions for the right premise, which is not possible. Hence, the inference rule (15) has no ground instances. ⋄

The problem spotted in Example 3.35 can be addressed using an extension of the standard redundancy criterion (see Definition 3.17) for general clauses:

Definition 3.36 (Standard Redundancy). Let $\succ$ be an admissible order according to Definition 3.34. A *clause C is redundant w.r.t. N* (notation: $C \in \mathsf{R}^{\succ}_{\mathrm{Cl}}(N)$), if every ground instance $C\sigma$ of C is redundant w.r.t. N^{gr} (according to Definition 3.17). An *inference $\pi = C_1,..,C_k \vdash C$ is redundant w.r.t. N (in $\boldsymbol{\mathcal{S}}$)* (notation: $\pi \in \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}}(N)$), if every ground instance $\pi\sigma := C_1\sigma,..,C_k\sigma \vdash C\sigma$ (i.e., when all $C_i\sigma$ with $i = 1,..,k$ and $C\sigma$ are ground), is redundant w.r.t. N^{gr} in $\boldsymbol{\mathcal{S}}^0$, where $\boldsymbol{\mathcal{S}}^0$ is the restriction of $\boldsymbol{\mathcal{S}}$ to ground clauses. ⋄

Note that an instance $\pi\sigma$ may be not an inference of $\boldsymbol{\mathcal{S}}^0$ even if $\pi \in \boldsymbol{\mathcal{S}}$, since the unified expressions might not match, or ordering restrictions may be violated. For example, the inference (15) in Remark 3.35 has no valid ground instances. In this case the inference π is *vacuously* redundant, since all its ground inferences are not in $\boldsymbol{\mathcal{S}}^0$ and hence redundant (see remark after Definition 3.17). Inferences that are outside $\boldsymbol{\mathcal{S}}$ are redundant as well, since according to the lifting diagram (see Figure 7) they may not have valid ground instances.

Lemma 3.37 (Standard Redundancy Criterion). *The notion of standard redundancy $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ} = (\mathsf{R}^{\succ}_{\mathrm{Cl}}(\cdot), \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}}(\cdot))$ given in Definition 3.36 is a redundancy criterion for general clauses Cl_Σ. In addition, if $\boldsymbol{\mathcal{S}}^0$ is monotone, then $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}$ is effective.*

Proof. Lemma 3.37 is a consequence of Lemma 3.21 since by Definition 3.36, for every clause sets N' and N, we have $N \subseteq \mathsf{R}^{\succ}_{\mathrm{Cl}}(N)$ *iff* $N^{\mathrm{gr}} \subseteq \mathsf{R}^{\succ}_{\mathrm{Cl}^0}(N^{\mathrm{gr}})$, and for every set of inferences S, we have $S \subseteq \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}}(N)$ *iff* $S^{\mathrm{gr}} \subseteq \mathsf{R}^{\boldsymbol{\mathcal{S}}^0\succ}_{\mathrm{Inf}^0}(N^{\mathrm{gr}})$, where S^{gr} is the set of all ground instances of inferences in S. ⊡

To demonstrate a non-trivial case of redundant inferences, we consider an example from [Nieuwenhuis & Rubio, 2001], that shows how resolution inferences between transitivity axioms can be avoided.

Example 3.38. Consider the set consisting of the single clause $\mathtt{T} : \neg\mathtt{p}(\mathtt{x},\mathtt{y}) \vee \neg\mathtt{p}(\mathtt{y},\mathtt{z}) \vee \mathtt{p}(\mathtt{x},\mathtt{z})$ that expresses the *transitivity axiom* for $\mathtt{p}$: "$\mathtt{p}(\mathtt{x},\mathtt{y})$ *and* $\mathtt{p}(\mathtt{y},\mathtt{z})$ *implies* $\mathtt{p}(\mathtt{x},\mathtt{z})$". Let $\succ$ be some admissible ordering that is a *rewrite ordering* for ground expressions (see subsection 2.4, p. 21), say *KBO* or *LPO* ordering.

Every negative literal of $\mathtt{T}$ is maximal in $\mathtt{T}$, since it is maximal in its instance where all variables are replaced by the same ground term. For many orderings $\succ$

(again, in particular for *KBO* or *LPO*), the positive literal is also maximal, hence the Ordered Resolution rule can be applied to (two copies of) $\mathtt{T}$:

$$\mathtt{OR}: \frac{\neg \mathtt{p}(\mathtt{x},\mathtt{y}) \vee \neg \mathtt{p}(\mathtt{y},\mathtt{z}) \vee \underline{\mathbf{p(x,z)}}^{\star} \quad \underline{\neg \mathbf{p(u,v)}} \vee \neg \mathtt{p}(\mathtt{v},\mathtt{w}) \vee \mathtt{p}(\mathtt{u},\mathtt{w})}{\neg \mathtt{p}(\mathtt{x},\mathtt{y}) \vee \neg \mathtt{p}(\mathtt{y},\mathtt{z}) \vee \neg \mathtt{p}(\mathtt{z},\mathtt{w}) \vee \mathtt{p}(\mathtt{x},\mathtt{w})} \tag{17}$$

with the unifier is $\sigma := \{\mathtt{u}/\mathtt{x}, \mathtt{v}/\mathtt{z}\}$. The conclusion of this inference is the clause:

$\mathtt{OR}[\mathtt{T};\mathtt{T}]: \ \mathtt{T}'. \ \neg \mathtt{p}(\mathtt{x},\mathtt{y}) \vee \neg \mathtt{p}(\mathtt{y},\mathtt{z}) \vee \neg \mathtt{p}(\mathtt{z},\mathtt{w}) \vee \mathtt{p}(\mathtt{x},\mathtt{w});$

which can be seen as a weak transitivity axiom. For similar reasons as for $\mathtt{T}$, the clause $\mathtt{T}'$ can be further resolved with $\mathtt{T}$ producing even longer clauses:

$\mathtt{OR}[\mathtt{T}';\mathtt{T}]: \ \mathtt{T}''. \ \neg \mathtt{p}(\mathtt{x},\mathtt{y}) \vee \neg \mathtt{p}(\mathtt{y},\mathtt{z}) \vee \neg \mathtt{p}(\mathtt{z},\mathtt{u}) \vee \neg \mathtt{p}(\mathtt{u},\mathtt{w}) \vee \mathtt{p}(\mathtt{x},\mathtt{w});$
$\mathtt{OR}[\mathtt{T}'';\mathtt{T}]: \ \mathtt{T}'''. \ \neg \mathtt{p}(\mathtt{x},\mathtt{y}) \vee \neg \mathtt{p}(\mathtt{y},\mathtt{z}) \vee \neg \mathtt{p}(\mathtt{z},\mathtt{u}) \vee \neg \mathtt{p}(\mathtt{u},\mathtt{v}) \vee \neg \mathtt{p}(\mathtt{v},\mathtt{w}) \vee \mathtt{p}(\mathtt{x},\mathtt{w});$
$\ldots$ etc.

Now we show how the notion of redundancy can be used to avoid all these dangerous inferences. First we show that the clause $\mathtt{T}'$ is *not* always redundant w.r.t. the clause set $\{\mathtt{T}\}$ when the inferences above are possible.

Consider any *LPO* ordering $\succ_{lpo}$ and suppose that the clause $\mathtt{T}'$ is redundant w.r.t. $\{\mathtt{T}\}$. By Definition 3.36, this in particular means that the following ground instance of $\mathtt{T}'$:

$$\mathtt{T}'_0. \ \neg \mathtt{p}(s^0, r^0) \vee \neg \mathbf{p}(\boldsymbol{r^0}, \boldsymbol{t^0})^{\star} \vee \neg \mathtt{p}(t^0, h^0) \vee \mathtt{p}(s^0, h^0) \tag{18}$$

with $r^0 \succ_{lpo} s^0 \succ_{lpo} t^0 \succ_{lpo} h^0$, follows from the set $\{\mathtt{T}'\}^{\mathrm{gr}}_{\mathtt{T}'_0}$ of smaller ground instances of $\mathtt{T}$. We claim that $\{\mathtt{T}'\}^{\mathrm{gr}}_{\mathtt{T}'_0}$ contains either the left clause from (19) or the left clause from (20) below:

$$\neg \mathtt{p}(s^0, r^0) \vee \neg \mathtt{p}(r^0, t^0) \vee \underline{\mathtt{p}(s^0, t^0)}; \qquad \neg \underline{\mathtt{p}(s^0, t^0)} \vee \neg \mathtt{p}(t^0, h^0) \vee \mathtt{p}(s^0, h^0); \tag{19}$$

$$\neg \mathtt{p}(r^0, t^0) \vee \neg \mathtt{p}(t^0, h^0) \vee \underline{\mathtt{p}(r^0, h^0)}; \qquad \neg \mathtt{p}(s^0, r^0) \vee \neg \underline{\mathtt{p}(r^0, h^0)} \vee \mathtt{p}(s^0, h^0). \tag{20}$$

Indeed, otherwise $I_0 := \{\mathtt{p}(s^0, r^0), \mathtt{p}(r^0, t^0), \mathtt{p}(t^0, h^0)\}$ is a model of $\{\mathtt{T}'\}^{\mathrm{gr}}_{\mathtt{T}'_0}$: the only instances of $\mathtt{T}$ in which both negative literals are false are the left clauses from (19) and (20), which we assume to be not in $\{\mathtt{T}'\}^{\mathrm{gr}}_{\mathtt{T}'_0}$. But I_0 is not a model of $\mathtt{T}'_0$, hence $\mathtt{T}'_0$ does not follow from $\{\mathtt{T}'\}^{\mathrm{gr}}_{\mathtt{T}'_0}$.

On the other hand, none of these left clauses may be in $\{\mathtt{T}'\}^{\mathrm{gr}}_{\mathtt{T}'_0}$, because they are $\succ_{lpo}$-greater than $\mathtt{T}'_0$, since $\mathtt{p}(s^0, t^0) \succ_{lpo} \neg \mathtt{p}(t^0, h^0) \vee \mathtt{p}(s^0, h^0)$ and $\mathtt{p}(r^0, h^0) \succ_{lpo} \neg \mathtt{p}(s^0, r^0) \vee \mathtt{p}(s^0, h^0)$. Hence $\mathtt{T}'_0$ does not follow from $\{\mathtt{T}'\}^{\mathrm{gr}}_{\mathtt{T}'_0}$ and so $\mathtt{T}'$ is not redundant w.r.t. $\{\mathtt{T}\}$.

Although the clause $\mathtt{T}'$ is not redundant w.r.t. $\{\mathtt{T}\}$, it is possible to show that *inference* (17) that has produced $\mathtt{T}'$, is redundant w.r.t. $\{\mathtt{T}\}$ for *every monotone*

admissible ordering $\succ$ *that is total on ground terms.*[7] To show redundancy, according to Definition 3.36, we should consider arbitrary ground instance (19) of the Ordered Resolution inference (17) that produces instance (18) of the conclusion of this inference (we discard the *LPO*-ordering considered before). We need to demonstrate that clause (18) follows from some instances of $\mathtt{T}$ that are smaller than the maximal premise of this ground inference, namely the right clause from (19). We show that the clauses from (20) are those instances that we need.

Obviously, clause (18) is a consequence of (20). What remains to be shown, is that both clauses from (20) are smaller than the right clause from (19). To prove this, we use the conditions of the Ordered Resolution rule that is applied to (19). The ordering restrictions for the left premise yield: $\mathtt{p}(,t^0) \succ \neg\mathtt{p}(,r^0) \succ \mathtt{p}(,r^0)$ and $\mathtt{p}(,t^0) \succ \neg\mathtt{p}(r^0,t^0) \succ \mathtt{p}(r^0,t^0)$. By monotonicity and totality of $\succ$ on ground terms, we obtain that $t^0 \succ r^0$ and $\succ r^0$. This together with the ordering restrictions for the right premise yield $\neg\mathtt{p}(,t^0) \succeq \neg\mathtt{p}(t^0,h^0) \succ \neg\mathtt{p}(r^0,h^0) \succ \mathtt{p}(r^0,h^0)$. Now it is easy to check that the right clause from (19) is $\succ$-larger than both clauses from (20). Hence, the inference (17) is redundant w.r.t. $\{\mathtt{T}\}$.

Similarly, it can be shown that the resolution inference with the second negative literal of $\mathtt{T}$ is redundant. Here we need to consider an arbitrary instance (20) of the ordered resolution inference and, using the ordering restrictions demonstrate, that the conclusion of this inference can be obtained by (unordered) resolution from the *smaller* clauses (19). ⋘

It is hard to come up with a general algorithm using which redundancy of clauses and inferences can be shown. In the Saturate system [Ganzinger, Nieuwenhuis & Nivela, 2002], some concrete techniques for proving redundancy are implemented. For the particular example above with $\succ = \succ_{lpo}$, this prover finds redundancy automatically by employing *clausal rewriting* combined with *LPO* constraint solving. In [Kazakov, 2005] we demonstrate how the proof model described in Example 3.38 can be adapted to obtain decision procedures for variety of non-trivial fragments of first-order logic with transitive predicates and related theories.

The notion of saturation up to redundancy (see Definition 3.26) can be applied to the ordered resolution calculus $\mathcal{OR}^{\succ}$ with the (extended) standard redundancy criterion from Definition 3.36. Our goal now is to show refutational completeness of this calculus.

Theorem 3.39 (Completeness of $\mathcal{OR}^{\succ}$ with Redundancy). $(\mathcal{OR}^{\succ}, \mathsf{R}^{s\succ})$ *is refutationally complete for* Cl_{Σ}.

Proof. Since $\mathsf{R}^{s\succ}$ is a redundancy criterion (see Lemma 3.37), by the completeness criterion (see Corollary 3.28), it suffices to show that every unsatisfiable set N that

[7] In [Nieuwenhuis & Rubio, 2001] redundancy of this inference has been shown only for *LPO*

is saturated in $\mathcal{OR}^{\succ}$ up to redundancy, contains the empty clause $\square$. Furthermore, it suffices to prove that (***i***) if N is $(\mathcal{OR}^{\succ}, \mathsf{R}^{\boldsymbol{s}\succ})$-saturated then N^{gr} is $(\mathcal{OR}^{0\succ}, \mathsf{R}^{\boldsymbol{s}\succ}_{\mathrm{gr}})$-saturated. Indeed, this together with (***ii***) N is unsatisfiable implies that N^{gr} is unsatisfiable and (***iii***) $\square \notin N$ implies that $\square \notin N^{\mathrm{gr}}$, yield a contradiction if $\square \notin N$, since $(\mathcal{OR}^{0\succ}, \mathsf{R}^{\boldsymbol{s}\succ}_{\mathrm{gr}})$ is complete (see Corollary 3.30).

The fact (***i***) is a consequence of the following two lemmas that follow directly from the introduced definitions:

Lemma 3.40. *For every clause $C^0 \in N^{\mathrm{gr}}$, there exists a clause in $C \in N$ such that C^0 is a ground instance of C and for every literal L^0 that is (strictly) maximal in C^0, the correspondent literal L in C is (strictly) maximal in C.*

Lemma 3.41. *For every inference $\pi^0 = C^0_1,..,C^0_k \vdash C^0 \in \mathcal{OR}^{0\succ}$ from N^{gr} there exists an inference $\pi = C_1,..,C_k \vdash C \in \mathcal{OR}^{\succ}$ from N, such that π^0 is an instance of π. Moreover, if π is redundant w.r.t. N then π^0 is redundant w.r.t. N^{gr}.* ⧉

3.6 Selection Functions for General Clauses

Selection functions can be extended to clauses with variables in the same way as given in Definition 3.11, that is a selection function *Sel* selects a sub-multiset *Sel*(C) of negative literals in *every* clause C. To integrate selection functions into $\mathcal{OR}^{\succ}$ as it is done for $\mathcal{OR}^{0\succ}_{Sel}$, we extend the notion of eligible literals (see Definition 3.11) as follows: Given a clause C and a substitution σ, we say that a literal L *is eligible (strictly maximal) w.r.t. C and σ* if either $L \in Sel(C \vee L)$, or otherwise $Sel(C \vee L) = \{\}$ and $L\sigma$ is (strictly) maximal w.r.t. $C\sigma$. The *Ordered Resolution calculus with selection* $\mathcal{OR}^{\succ}_{Sel}$ parametrized by a selection function *Sel* and an admissible ordering $\succ$ is given in System 7.

Ordered Resolution

$$\mathtt{OR}: \frac{C \vee \underline{\boldsymbol{A}}^{\star} \quad D \vee \neg\underline{\boldsymbol{B}}}{C\sigma \vee D\sigma}$$

where (i) $\sigma = mgu(A, B)$; (ii) A is eligible strictly maximal w.r.t. C and σ and (iii) $\neg B$ is eligible w.r.t. D and σ.

Ordered Factoring

$$\mathtt{OF}: \frac{C \vee \underline{B} \vee \underline{\boldsymbol{A}}}{C\sigma \vee A\sigma}$$

where (i) $\sigma = mgu(A, B)$; (ii) A is eligible w.r.t. C and σ.

System 7: The ordered resolution calculus with selection $\mathcal{OR}^{\succ}_{Sel}$

There are some technical difficulties in proving refutational completeness of the ordered resolution with selection functions. In particular, the calculus in System 7 is no longer compatible with the lifting diagram in Figure 7, since the selected literals in a clause and its ground instances may not correspond. It is possible to

fix this problem by allowing only "liftable" selection functions in similar way how it is done for orderings. However this is not desirable in practice, where a selection functions are used to control the behaviour of a saturation procedure, like in the example below.

Example 3.42. To demonstrate the problems that arise with arbitrary selection functions, consider the clause set

$$N_0 = \{\mathtt{p(x, f(x))}^{\star},\ \mathtt{q(x, f(x))}^{\star},\ \neg\mathtt{p}(\mathtt{x},\mathtt{y}) \vee \neg\mathtt{q(x, x)}^{\sharp},\ \neg\mathtt{p(x, x)}^{\sharp} \vee \neg\mathtt{q}(\mathtt{x},\mathtt{y})\}.$$

First, it is reasonable to set the selection function *Sel* to select the negative literals with one variable in the last two clauses, since this ensures that no inferences can be drawn between the clauses (here and further we mark selected literals by $^{\sharp}$). Now, the question is, what should be selected in clause $\neg\mathtt{p}(\mathtt{c},\mathtt{c}) \vee \neg\mathtt{q}(\mathtt{c},\mathtt{c})$, which is a common ground instance of the last two clauses? Regardless which selection function we choose, the lifting diagram in Figure 7 will not work: If literal $\neg\mathtt{p}(\mathtt{c},\mathtt{c})$ becomes eligible in this ground instance, then the inference between clauses $\mathtt{p}(\mathtt{x},\mathtt{x})$ and $\neg\mathtt{p}(\mathtt{x},\mathtt{y}) \vee \neg\mathtt{q(x, x)}^{\sharp}$ is not possible, although it is possible between its ground instances $\mathtt{p}(\mathtt{c},\mathtt{c})$ and $\neg\mathtt{p(c, c)} \vee \neg\mathtt{q}(\mathtt{c},\mathtt{c})$. The situation when literal $\neg\mathtt{q}(\mathtt{c},\mathtt{c})$ is eligible in clause $\neg\mathtt{p}(\mathtt{c},\mathtt{c}) \vee \neg\mathtt{q}(\mathtt{c},\mathtt{c})$ is symmetric.

However for the *fixed* clause set N, it is always possible to extend the selection function to ground literals such that every inference between clauses from N^{gr} will correspond to some inferences between clauses from N. This idea is used for proving completeness of $\mathcal{OR}^{\succ}_{Sel}$ with non-liftable selection functions. Please find in Appendix A.1 the details of the proof. ◈

There are other related technical problems with general selection functions, in particular, how to define the notion of redundant inferences. This is also discussed in Appendix A.1. Using techniques discuss in this appendix, it is possible to justify so-called *subsumption deletion*:

Definition 3.43 (Subsumption). A clause C *subsumes* a clause D (or D is *subsumed by* C) if $C\sigma \subseteq D$ for some substitution σ (the inclusion between clauses is the inclusion between their multisets of literals). A clause C *strictly subsumes* a clause D (D is *strictly subsumed by* C) if C subsumes D but D does not subsume C. ◈

For example, the clause $\mathtt{a}(\mathtt{x}) \vee \mathtt{a}(\mathtt{y})$ subsumes the clauses $\mathtt{a}(\mathtt{x}) \vee \mathtt{a}(\mathtt{x})$, $\mathtt{a}(\mathtt{y}) \vee \mathtt{a}(\mathtt{x}) \vee \mathtt{a}(\mathtt{c})$, $\mathtt{a}(\mathtt{f}(\mathtt{x})) \vee \mathtt{a}(\mathtt{c})$, but not, say the clause $\mathtt{a}(\mathtt{x})$ or the clause $\mathtt{a}(\mathtt{f}(\mathtt{x}))$. Note also that clauses which subsume each other are variants of each other.

It is possible to extend the notion of redundancy in such a way that clauses that are *strictly subsumed* by C are *redundant* w.r.t. C and hence can be deleted during a saturation process. For details, please see Appendix A.2.

3.7 Hyper-Resolution Strategies

The extension of ground resolution calculus with the hyper-inference given in Figure 4 can be lifted in a similar way. To formulate the resulted inference system, we say that a multiset $D' = \{\neg B_1,..,\neg B_n\}_m$ of negative literals is *eligible w.r.t. D and σ (for hyper-resolution)* if either $(\boldsymbol{a})$ $D' = Sel(D \vee D') \neq \{\}_m$, or, otherwise $(\boldsymbol{b})$ $Sel(D \vee D') = \{\}_m$, $n = 1$ and $\neg B_1\sigma$ is maximal w.r.t. $D\sigma$. Now the *Ordered Hyper-Resolution calculus with selection* $\mathcal{HR}^{\succ}_{Sel}$ is defined by replacing the Ordered Resolution rule with the Ordered Hyper-resolution rule given in Figure 8.

Figure 8 The hyper-resolution rule

Ordered Hyper-resolution

$$\mathtt{HR}: \frac{C_1 \vee \underline{\boldsymbol{A_1}}^{\star} \quad \ldots \quad C_n \vee \underline{\boldsymbol{A_n}}^{\star} \qquad \neg\underline{\boldsymbol{B_1}} \vee \cdots \vee \neg\underline{\boldsymbol{B_n}} \vee D}{C_1\sigma \vee \cdots \vee D\sigma}$$

where (i) $\sigma = mgu(\{A_1{=}B_1, \ldots, A_n{=}B_n\})$; (ii) $A_i\sigma$ are eligible strictly maximal w.r.t. C_i and σ, $i = 1,..,n$ and (iii) $\{\neg B_1,..,\neg B_n\}_m$ is eligible w.r.t. D and σ.

The hyper-resolution rule can be further extended to a more powerful rule with a-posteriori selection strategy, which has been employed in [de Nivelle & de Rijke, 2003] for deciding the *loosely guarded fragment*. This rule can be described using the *a-posteriori selection function* Sel^a that assigns to a unification problem $P = \{A_1{=}B_1,.., A_n{=}B_n\}$ between first-order atoms, its non-empty subproblem: $Sel^a(P) \subseteq P$, $Sel^a(P) \neq \{\}$. The extended Ordered Hyper-resolution is given in Figure 9.

Figure 9 The hyper-resolution rule with a-posteriori selection

Ordered Hyper-resolution

$$\mathtt{HR}: \frac{C_1 \vee \underline{\boldsymbol{A_1}}^{\star} \quad \ldots \quad C_k \vee \underline{\boldsymbol{A_k}}^{\star} \quad \ldots \quad C_n \vee \boldsymbol{A_n}^{\star} \qquad \neg\underline{\boldsymbol{B_1}} \vee \cdots \vee \neg\underline{\boldsymbol{B_k}} \vee \cdots \vee \neg\boldsymbol{B_n} \vee D}{C_1\tau \vee \cdots \vee \neg B_{k+1}\tau \vee \cdots \vee \neg B_n\tau \vee D\tau}$$

where (i) $\sigma = mgu(\{A_1{=}B_1, \ldots, A_n{=}B_n\})$; (ii) A_i are eligible strictly maximal w.r.t. C_i and σ, $1 \leq i \leq n$; (iii) $\{\neg B_1,..,\neg B_n\}_m$ is eligible w.r.t. D and σ; (iv) $Sel^a(\{A_1{=}B_1,.., A_n{=}B_n\}) = \{A_1{=}B_1,.., A_k{=}B_k\}$ and (v) $\tau = mgu(\{A_1{=}B_1,.., A_k{=}B_k\})$.

4 Equational Reasoning

Reasoning with equality plays fundamental rôle in many applications of formal methods in mathematics and computer science. Hence, integration of equality into saturation-based theorem proving, became one of the central and hot topics of research in automated deduction shortly after introduction of the resolution calculus.

It is well-known that the theory of equality can be axiomatised in first-order logic using *axioms* of *congruence relations* (see subsection 2.1 on p. 12). However, resolution with congruence axioms appears to be highly inefficient. Problems arise already with the *transitivity axiom*, which produces many unnecessary inferences as has been demonstrated in Example 3.38. An alternative to the axiomatic approach have been found by Robinson & Wos [1969], who proposed a special inference rule called Paramodulation to handle equality. They proved that resolution augmented with the Paramodulation rule and special Functional Reflexivity axioms form a complete inference system for the first-order logic with equality (see System 8). Intuitively, the Paramodulation rule corresponds to transitivity and

Paramodulation

$$\mathtt{P}: \frac{C \vee \underline{s} \simeq t \quad D \vee L[\underline{s'}]}{C\sigma \vee D\sigma \vee L[t]\sigma}$$

where (i) $\sigma = mgu(s, s')$.

Functional Reflexivity

$$\mathtt{FR}: \frac{}{f(x_1, \ldots, x_n) \simeq f(x_1, \ldots, x_n)}$$

System 8: The Robinson & Wos's [1969] paramodulation calculus $\mathcal{P}$

monotonicity axioms for equality and the Functional Reflexivity axiom scheme, as can be guessed, corresponds to the reflexivity axiom for equality. Brand [1975] has proved later that the Functional Reflexivity axioms can be replaced with a more restricted Reflexivity Resolution rule and that *paramodulation into variables*, i.e., when s' is a variable is not necessary.

The basic paramodulation calculus evolved along with the resolution calculus: ordering restriction, selection strategies and redundancy criteria enhanced efficiency and flexibility of the calculus.[8] The completeness of the paramodulation calculus in its modern form, called the *ordered paramodulation calculus*, has been first demonstrated by Hsiang & Rusinowitch [1991] using a proof technique based on transfinite semantic trees. A refinement of the ordered paramodulation calculus called the *superposition calculus* [Bachmair & Ganzinger, 1990] is nowadays a standard method for equational reasoning and has been implemented in many

[8] See [Bachmair & Ganzinger, 1998*a*; Nieuwenhuis & Rubio, 2001; Degtyarev & Voronkov, 2001] for a historical exposition and comparison of techniques for equational reasoning

systems such as VAMPIRE [Riazanov & Voronkov, 2002], SPASS [Weidenbach et al., 2002], E [Schulz, 2002] and BLIKSEM [de Nivelle, 1998*a*].

In this section we review the *ordered paramodulation calculus* $\mathcal{OP}^{\succ}_{Sel}$ and the *superposition calculus* $\mathcal{SP}^{\succ}_{Sel}$. We present completeness proofs for these calculi according to the general model construction schema demonstrated for the resolution calculus. Although the superposition calculus is more restrictive than the paramodulation calculus, the last can be still used for deciding certain shallow clause classes, for example the one for the guarded fragment [see Kazakov, 2005].

4.1 The Ordered Paramodulation Calculus

The *ordered paramodulation calculus* $\mathcal{OP}^{\succ}_{Sel}$ is an extension of the ordered resolution calculus $\mathcal{OR}^{\succ}_{Sel}$ with two rules given in System 9. Like ordered resolution, this calculus is parametrized by an ordering $\succ$ on literals and a selection function *Sel*, but the ordering $\succ$ is now extended also to terms. In order to simplify the

Ordered Paramodulation

$$\texttt{OP}: \frac{C \vee \underline{s} \simeq t^{\star} \quad D \vee L[\underline{s'}]}{C\sigma \vee D\sigma \vee L[t]\sigma}$$

where (i) $\sigma = mgu(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $L[s']$ is eligible (strictly maximal if positive) w.r.t. D and σ; (iv) $(s\sigma \simeq t\sigma) \not\succeq L[s']\sigma$; (v) $t\sigma \not\succeq s\sigma$ and (vi) s' is not a variable.

Reflexivity Resolution

$$\texttt{RR}: \frac{C \vee \underline{s} \not\simeq \underline{s'}}{C\sigma}$$

where (i) $\sigma = mgu(s, s')$ and (ii) $s \not\simeq s'$ is eligible w.r.t. C and σ.

System 9: The ordered paramodulation calculus $\mathcal{OP}^{\succ}_{Sel}$

exposition of calculi with equality, we identify every *non-equational atom* $A \in \mathrm{At}^{-}_{\Sigma}$ with the equation $A \simeq \mathtt{T}$ (and its negation with $A \not\simeq \mathtt{T}$), where $\mathtt{T}$ is some fixed constant, which intuitively stands for "True". This allows us to deal only with equational atoms of the form $E_1 \simeq E_2$ over two sorts of expressions $E_1, E_2 \in \mathrm{At}^{-}_{\Sigma} \sqcup \mathrm{Tm}_{\Sigma}$.

Definition 4.1 (Admissible Order). An ordering $\succ$ is *admissible for parmodulation* if $\succ$ is admissible for resolution (see Definition 3.34) and additionally:

(**T**) $\succ$ is total on ground terms with the least element $\mathtt{T}$;
(**E1**) $t \prec s \lhd L$ implies $L[s] \succ L[t]$ (monotonicity);
(**E2**) $t \prec s \lhd E_1$ implies $(E_1[s] \simeq E_2) \succ (s \simeq t)$; ⋘

Condition (T) and (E1) of admissible ordering together with (L) from Definition 3.4 imply that $\succ$ is a total reduction ordering on ground expressions (see p. 21). Condition (E2) ensures that an equational atom $s \simeq t$ used in a paramodulation inference is always smaller than the atom into which paramodulation is

performed, except, perhaps, for equational atoms of the form $s \simeq h$ with $s \succ t \succ h$. In this case the last atom can be paramodulated into the first. Note that since $\succ$ is liftable, it suffices to require conditions (E1) – (E2) for ground expressions only.

An admissible ordering for paramodulation can be obtained, for example, by taking the lifting of any total reduction ordering $\succ$ on ground terms and non-equational ground atoms, like *KBO* or *LPO*, extended to equational literals by treating every positive literal $E_1 \simeq E_2$ as the multiset $\{E_1, E_2\}_m$ and every negative literal $E_1 \not\simeq E_2$ as the multiset $\{E_1, E_1, E_2, E_2\}_m$.

It is easy to show that Ordered Paramodulation and Reflexivity Resolution are sound and that the restriction of $\mathcal{OP}^{\succ}_{Sel}$ to ground clauses forms a monotone inference system:

Lemma 4.2 (Monotonicity for $\mathcal{OP}^{\succ}_{Sel}$). *$\mathcal{OP}^{\succ}_{Sel}$ is monotone for ground clauses w.r.t. every admissible ordering $\succ$.*

Proof. Monotonicity of the Ordered Resolution and Ordered Factoring rules has been already shown in Lemma 3.13 (recall that every ordering $\succ$ that is admissible for paramodulation is also admissible for resolution). It remains to show monotonicity of the Ordered Paramodulation and Reflexivity Resolution rules:

(**1**) $\mathtt{OP} : C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star},\ D \vee \boldsymbol{L}[\underline{\boldsymbol{s}}] \vdash C \vee D \vee L[t]$ (Ordered Paramodulation):
We show that the conclusion of the Ordered Paramodulation rule is always smaller than the right premise of this rule. Indeed, by conditions (iv) and (ii) of this rule, we must have $L[s] \succ (s \simeq t) \succ C$. By condition (v) of this rule and condition (E1) of admissible ordering, we have $L[s] \succ L[t]$. Hence, $D \vee L[s] \succ (C \vee D \vee L[t])$ what was required to show.

(**2**) $\mathtt{RR} : C \vee \underline{s} \not\simeq \underline{s} \vdash C$ (Reflexivity Resolution):
This inference rule is obviously monotone since $\succ$ is monotone w.r.t. multiset inclusion. ⧉

The standard redundancy criterion $\mathsf{R}^{\boldsymbol{s}\succ} = (\mathsf{R}^{\succ}_{\mathrm{Cl}}(\cdot), \mathsf{R}^{\boldsymbol{s}\succ}_{\mathrm{Inf}}(\cdot))$ is extended to equational calculi without new surprises. As usual, we say that a *ground clause C^0 is redundant w.r.t.* a set of ground clauses N^0, if C^0 follows from ${N^0}_{C^0}$. A *ground inference π^0 is redundant w.r.t. N^0 in $\mathcal{OP}^{0\succ}_{Sel'}$* if either $\pi^0 \notin \mathcal{OP}^{0\succ}_{Sel'}$ or, otherwise, the conclusion of π^0 follows from ${N^0}_{C^0_i}$ for some premise C^0_i of π^0. Note, that "follows" here means in the theory of equality. We will often say that such redundancy criterion is *based on semantical entailment*, in this case in equational theory.

A (general) *clause C is redundant* w.r.t. a clause set N if every ground instance $C^0 \in \{C\}^{\mathrm{gr}}$ of C is redundant w.r.t. N^{gr}. An *inference π is redundant w.r.t. N in $\mathcal{OP}^{\succ}_{Sel}$*, if every ground instance π^0 of π is redundant w.r.t. N^{gr} in $\mathcal{OP}^{0\succ}_{Sel'}$ for *every* projection *Sel′* of *Sel* from the premises of π.

4.1.1 Refutational completeness for the ground clauses

In this section we consider the restriction $\mathcal{OP}^{0\succ}_{Sel}$ of the ordered paramodulation calculus $\mathcal{OP}^{\succ}_{Sel}$ to the ground clauses Cl^0_Σ. Unless stated otherwise, all terms and clauses in this section are ground.

We extend the model construction given in subsection 3.1 to the paramodulation calculus. Given a set of *ground* clauses N that is saturated (up to redundancy), we are going to construct a candidate model I_N for N. Every model that we construct in our procedure is represented by a set of *oriented* ground equational atoms I, which form a rewrite system: $E_1 \simeq E_2$ with $E_1 \succ E_2$ is read as $E_1 \Rightarrow E_2$. The *(rewrite) model induced by* I is denoted by $I{\Downarrow}$ and is defined by setting an atom $A = (E_1 \simeq E_2)$ to be $\mathtt{true}$ in $I{\Downarrow}$ (notation: $I{\Downarrow} \vDash A$) *iff* the equation $E_1 \simeq E_2$ converges w.r.t. $\Rightarrow_I$, i.e., the expressions E_1 and E_2 are I-joinable: $E_1 \Downarrow_I E_2$ (recall the notations and definitions from subsection 2.3). In particular, a non-equational atom $A \in \mathrm{At}^-_\Sigma$ is $\mathtt{true}$ in $I{\Downarrow}$, *iff* $A{\Downarrow_I} = \mathtt{T}$.

The interpretation $I{\Downarrow}$ is equational only if the underlying rewrite system is convergent. By Critical Pair Lemma (Lemma 2.11), a rewrite system is convergent if and only if it is terminating and confluent. Termination of I is guaranteed by the ordering $\succ$, since every admissible ordering is well-founded. Confluence of I will be implied from a stronger property:

Definition 4.3 (Canonical Rewrite System). Given an atom A and a set of atoms I, let I_A (I^A) denote a set of atoms from I that are smaller (resp. smaller or equal) than A. We say that an atom $A = (E_1 \simeq E_2)$ is *irreducible* w.r.t. I if both E_1 and E_2 are irreducible w.r.t. I. A rewrite system (= set of atoms) I is *canonical (for paramodulation)* if every atom A from I is irreducible w.r.t. I_A. $\lll$

It will be shown below that every canonical rewrite system is confluent. In addition, canonical rewrite systems admit a nice property, namely that for proving an equation $E_1 \simeq E_2$, it suffices to use only those rewrite rules that are not greater than this equation:

Lemma 4.4 (Canonical Proofs). *Let I be a canonical rewrite system. Then ($\boldsymbol{i}$) I is confluent and ($\boldsymbol{ii}$) for every atom $A := (E_1 \simeq E_2)$, we have $I{\Downarrow} \vDash A$ iff $I^A{\Downarrow} \vDash A$.*

Proof. (i) In order to show that I is confluent, let $A = (E_1 \Rightarrow E_2)$ and $A' = (E'_1 \Rightarrow E'_2)$ be two different *overlapping* rewrite rules from I with $E'_1 \trianglelefteq E_1$, then either ($\boldsymbol{a}$) $E'_1 \triangleleft E_1$ and hence $A \succ A'$ (by condition (E2) of admissible ordering), which means that A is reducible w.r.t. I_A, or ($\boldsymbol{b}$) $E_1 = E'_1$ and either A is reducible w.r.t. I_A (if $A \succ A'$), or A' is reducible w.r.t. $I_{A'}$ (if $A' \succ A$). Both cases are not possible since I is canonical.

(*ii*) The "*if*" part of this case is trivial since $I^A \subseteq I$. To show the "*only if*" part, we employ induction on A over the well-ordered set $(\mathrm{At}^0_\Sigma, \succ)$. Recall, that $I\Downarrow \models (E_1 \simeq E_2)$ *iff* $E_1 \Downarrow_I E_2$. If both expressions in E_1 and E_2 from A are irreducible w.r.t. I, then $E_1 = E_2$, and the lemma is trivial, which provides the basis of induction.

W.l.o.g. assume that expression E_1 is reducible by some rewrite rule $(E'_1 \Rightarrow E'_2) \in I$. If $A \succeq (E'_1 \Rightarrow E'_2)$, then by the induction hypothesis, the equation $(E_1[E'_1/E'_2] \simeq E_2) \prec A$ (by condition (E1)) is `true` in $I^A\Downarrow$, and so $I^A\Downarrow \models A$, which was required to show.

Now suppose that $A \prec (E'_1 \Rightarrow E'_2)$. Since $E'_1 \trianglelefteq E_1$, this is only possible if $E'_1 = E_1$, because of condition (E2) for admissible orderings. For similar reasons, $E'_2 \succ E_2$ and E'_2 is irreducible w.r.t. I (otherwise it is reducible by some rewrite rule that is smaller than $E'_1 \Rightarrow E'_2$, which is not possible since I is canonical). Hence $E_1\Downarrow_I = E'_2 \succ E_2$ and the equation $E_1 \simeq E_2$ cannot converge. Therefore this case is not possible. ⧉

Now we describe a model construction for the paramodulation calculus.

Definition 4.5 (Productive Clause). A clause C is *productive* w.r.t. a set of atoms I, if (***i***) $I\Downarrow \not\models C$; (***ii***) $C = C' \vee A$, where $A = (E_1 \simeq E_2)$ is an eligible strictly greatest atom w.r.t. C' and $E_1 \succ E_2$,[9] and (***iii***) A is irreducible w.r.t. I. In this case we say that C *produces* a rewrite rule $\Delta^I C := (E_1 \Rightarrow E_2)$ w.r.t. I. If C is not productive, we assign $\Delta^I C := \{\}$. ⋄

Definition 4.6 (Candidate Models). Let $N \subseteq \mathrm{Cl}^0_\Sigma$ and $C \in \mathrm{Cl}^0_\Sigma$. The *candidate models* for N, N_C and N^C are induced respectively by the sets I_N, I_C and I^C that are defined as follows:
$I_N := \bigcup_{C\in N} I^C$; $\quad I_C := \bigcup_{C'\in N_C} I^{C'}$, and $I^C := I_C \cup \Delta C$, where $\Delta C := \Delta^{I_C} C$. ⋄

Lemma 4.7 (Properties of Candidate Models). *Let N be a clause set for which candidate models are constructed according to Definition 4.6. Then for every $C \in N$, (**i**) $I^C\Downarrow \models C$ implies that $I_N\Downarrow \models C$ and (**ii**) If $C = (C' \vee \boldsymbol{A}^\star)$ is productive w.r.t. I_C then $I_N\Downarrow \not\models C'$.*

N	*Clauses* C	ΔC
C	$C' \vee \neg B$	?

Proof. (*i*) If $I^C\Downarrow \models A$ for some positive atom A in C, then $I_N\Downarrow \models A \models C$. Otherwise, C should contain negatively some atom B such that $I^C\Downarrow \not\models B$, but $I_N\Downarrow \models B$. This situation is not possible, since by Lemma 4.4, $I^B\Downarrow \models B$, but $I^B \subseteq I^C \not\models B$, since $B \prec \neg B \preceq C$ (by condition (R1) of admissible ordering).

[9] Note that $E_1 = E_2$ is already not possible by condition (***i***), hence the maximal atom in C can be always oriented

(*ii*) Assume that for some productive clause $C = (C' \vee \boldsymbol{A}^\star)$, we have $I_N{\Downarrow} \models C'$. First, note that $I_C{\Downarrow} \not\models C'$ since C is a productive clause. Hence there exists an atom B which occurs positively in C', and becomes `true` in $I_N{\Downarrow}$. Since I_N forms a canonical rewrite system (by condition (*iii*) of productive clause), by Lemma 4.4 (*ii*), $I^B{\Downarrow} \models B$. Since $C \succeq A \succ B$, we have $I_C{\Downarrow} \models I^B{\Downarrow} \models B \models C'$. A contradiction. ⧉

N	*Clauses* C	ΔC
C	$\overbrace{C'' \vee B}^{C'} \vee \boldsymbol{A}^\star$	A

Lemma 4.8 (Counterexample-Reduction Lemma). *Let N be a set of ground clauses which is saturated w.r.t. $\mathcal{OP}^{\succ}_{Sel}$ up to redundancy. Then for every clause $D \in N$ with $I_N{\Downarrow} \not\models D \neq \square$, there exists $D_1 \in N$ such that $D_1 \prec D$ and $I_N{\Downarrow} \not\models D_1$.*

Proof. Let $I_N{\Downarrow} \not\models D$ for some clause $D \in N$. Then $I^D{\Downarrow} \not\models D$, since otherwise by Lemma 4.7 (*i*) we would have $I_N{\Downarrow} \models D$. In particular, clause D cannot be productive. So, we have $I_D{\Downarrow} = I^D{\Downarrow} \not\models D$. According to the definition of productive clauses (see Definition 4.14), this situation is possible only in the following cases:

(**1**) $D = (D' \vee \neg \boldsymbol{A})$ for some eligible literal $\neg A$ in D, and $I_D{\Downarrow} \models A = (E_1 \simeq E_2)$. W.l.o.g. assume that $E_1 \succeq E_2$. Then either (**1.1**) $E_1 = E_2 = s$ for some ground term $s \in \mathrm{Tm}^0_\Sigma$, or (**1.2**) the atom A is reducible w.r.t. I_D.

N	*Clauses* C	ΔC
D	$D' \vee \boldsymbol{s \not\simeq s}$	$-$
Υ		
D_1	D'	?

In case (1.1), the Reflexivity Resolution rule can be applied to D that produces a *smaller* clause $\mathtt{RR}[D]$: $D_1 = D' \in N$ than D which is `false` in $I_N{\Downarrow}$.

In case (1.2), A must be reducible by some rewrite rule produced by a clause $C = (C' \vee B) \prec D$. If this is a term rewrite rule $B = (s \Rightarrow t)$, then the Ordered Paramodulation rule can be applied to C and D that produces a clause $\mathtt{OP}[C, D]$: $D_1 = (C' \vee D' \vee \neg A[t]) \in N$ which is *smaller counterexample* than D. If $B = (P \Rightarrow \mathtt{T})$, where P is a non-equational atom, then $A = (P \Rightarrow \mathtt{T})$, hence the Ordered Resolution rule can be applied to C and D that produces a clause $\mathtt{OR}[C, D]$: $D_1 = (C' \vee D') \in N$ which is again a *smaller counterexample* than D.

N	*Clauses* C	ΔC
D	$D' \vee \neg \boldsymbol{A[s]}$	$-$
Υ		
D_1	$C' \vee D' \vee \neg A[t]$	?
C	$C' \vee \boldsymbol{s \simeq t^\star}$	$s \Rightarrow t$

(**2**) $D = (D' \vee A \vee \boldsymbol{A})$, where A is an eligible greatest literal in D. In this case, the Ordered Factoring rule is applied to D that produces a clause $\mathtt{OF}[D]$: $D_1 = (D' \vee A) \in N$ which is a *smaller counterexample* than D. ⧉

Counterexample-Reduction Lemma implies completeness of the ground ordered paramodulation calculus:

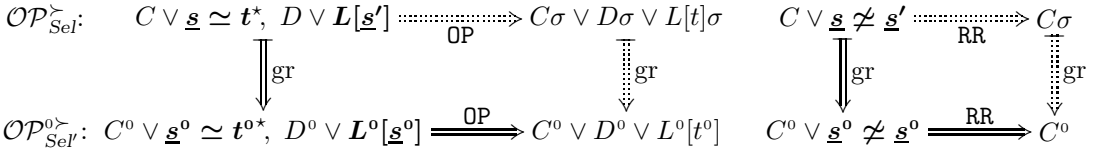
Theorem 4.9 (Completeness of $\mathcal{OP}^{0\succ}_{Sel}$ with Redundancy). *$(\mathcal{OP}^{0\succ}_{Sel}, \mathsf{R}^{\mathcal{S}\succ})$ is complete for the set Cl^0_Σ of ground clauses.*

Proof. Let $N \subseteq \mathrm{Cl}^0_\Sigma$ be an $(\mathcal{OP}^{0\succ}_{Sel}, \mathsf{R}^{\mathcal{S}\succ})$-saturated set of clauses that does not contain the empty clause $\square$. We show that N is `true` in $I_N\Downarrow$. Indeed, otherwise there exists a minimal w.r.t. $\succ$ counterexample $D_0 \in N$ for $I_N\Downarrow$, i.e., such that $I_N\Downarrow \not\models D_0$. By Counterexample-Reduction Lemma (Lemma 4.8), this may be only if $D_0 = \square$, which is not possible since $\square \notin N$. Hence $I_N\Downarrow$ is a model for N and by Completeness Criterion (Corollary 3.28), $(\mathcal{OP}^{0\succ}_{Sel}, \mathsf{R}^{\mathcal{S}\succ})$ is complete. ∎

4.1.2 Lifting

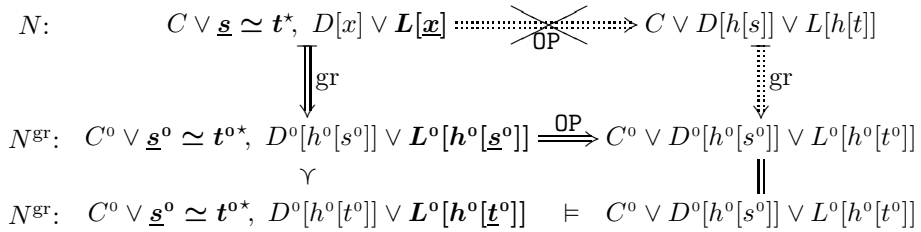
Completeness of the full paramodulation calculus $\mathcal{OP}^{\succ}_{Sel}$ can be shown by a similar lifting argument that has been employed for the ordered resolution calculus $\mathcal{OR}^{\succ}_{Sel}$. Given a clause set N that is saturated up to redundancy in $\mathcal{OP}^{\succ}_{Sel}$ one can show that the set of its ground instances N^{gr} is saturated up to redundancy in $\mathcal{OP}^{0\succ}_{Sel'}$ for every projection Sel' of Sel from N. This property follows from the lifting diagram for the ordered paramodulation calculus that is illustrated in Figure 10. According

Figure 10 The lifting diagram for $\mathcal{OP}^{\succ}_{Sel}$



to this lifting diagram, for every non-redundant inference from N^{gr} there exists a correspondent inference from N that captures the result of the ground inference. The only case when there is no correspondent inference from non-ground clauses is caused by a *non-liftable* condition (v) of the Ordered Paramodulation rule. This situation is illustrated in Figure 11. However, in this situation, it is possible to

Figure 11 A non-liftable paramodulation inference



show that this ground inference is redundant w.r.t. N^{gr}. Indeed, N^{gr} must contain a ground instance $D^0[h^0[t^0]] \vee L^0[h^0[t^0]]$ of clause $D[x] \vee L[x]$ which together with the left premise $C^0 \vee s^0 \simeq t^0$ of the ground Ordered Paramodulation inference implies

the conclusion of this inference. Since both of these clauses are *smaller* then the maximal clause $D^0[h^0[s^0]] \vee L^0[h^0[s^0]]$ of the inference (because $s^0 \succ t^0$), this inference is *redundant* according to the standard redundancy criterion. Hence the lifting diagram in Figure 10 holds (recall, that correspondence must be established only for non-redundant inferences).

Theorem 4.10 (Completeness of $\mathcal{OP}^{\succ}_{\mathbf{Sel}}$ with Redundancy). $(\mathcal{OP}^{\succ}_{Sel}, \mathsf{R}^{\mathcal{S}\succ})$ *is complete.*

Similar usage of redundancy can also justify a very useful refinement of the Ordered Paramodulation rule with *simultaneous paramodulation*: see Figure 12. In

Figure 12 The simultaneous ordered paramodulation rule

(Simulteneous) Ordered Paramodulation

$$\mathtt{OP}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D[\underline{s'}] \vee \boldsymbol{L}[\underline{\boldsymbol{s'}}]}{C\sigma \vee D[t]\sigma \vee L[t]\sigma}$$

where (i) $\sigma = mgu(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $L[s']$ is eligible (strictly maximal if positive) w.r.t. D and σ; (iv) $(s\sigma \simeq t\sigma) \not\geq L[s']\sigma$; (v) $t\sigma \not\geq s\sigma$ and (vi) s' is not a variable.

this rule, paramodulation is simultaneously performed into *several* occurrences of the term s' in the right premise. It can be shown that the usual paramodulation inference becomes redundant as long as the conclusion of the simultaneous paramodulation inference has been drawn:

$$\begin{array}{ccc} C \vee \boldsymbol{s} \simeq \boldsymbol{t}^{\star},\ D[s] \vee \boldsymbol{L}[\boldsymbol{s}] & \overset{\mathtt{OP}}{\Longrightarrow} & C \vee D[s] \vee L[t] \\ \mathtt{OP}\ (\text{Simult.}) \Big\Downarrow & \Upsilon & \Big\| \\ C \vee \boldsymbol{s} \simeq \boldsymbol{t}^{\star},\ C \vee D[t] \vee L[t] & \models & C \vee D[s] \vee L[t] \end{array} \tag{21}$$

The simulteneous paramodulation inference rule can be realized through an additional selection function, that given a clause, its literal and a subterm in this literal, selects some other occurrences of this subterm in this clause. The rule must be applied on all selected subterms in the clause. For the purpose of our decision procedures, we will use the *full selection*, i.e., we will apply simultaneous paramodulation rule on all subterms s' in the right premise.

4.2 The Superposition Calculus

We can notice some similarity between the ordered paramodulation calculus and the *ordered Knuth-Bendix completion* procedure described in subsection 2.3 in

System 2. Both saturation procedures result in an equational model that is represented by a convergent rewrite system. However the Superposition rule is more restricted then the Ordered Paramodulation rule in that the former rewrites only the largest term of an equational atom. The ideas behind the Knuth-Bendix completion and the paramodulation calculus have been joined into the *superposition calculus* that has been first introduced in [Bachmair & Ganzinger, 1990]. The *superposition calculus* $\mathcal{SP}^{\succ}_{Sel}$ is an extension of the ordered resolution calculus $\mathcal{OR}^{\succ}_{Sel}$ with inference rules given in System 10. Instead of a single Ordered Paramodula-

Ordered Paramodulation

$$\texttt{OP}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{L}[\underline{\boldsymbol{s}'}]}{C\sigma \vee D\sigma \vee L[t]\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $L[s']$ is eligible (strictly maximal if positive) w.r.t. D and σ; (iv) $L[s']$ is a non-equational literal; (v) $t\sigma \not\succeq s\sigma$ and (vi) s' is not a variable.

Positive Superposition

$$\texttt{PS}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{r}[\underline{\boldsymbol{s}'}] \simeq \boldsymbol{h}^{\star}}{C\sigma \vee D\sigma \vee r[t]\sigma \simeq h\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $r \simeq h$ is eligible strictly maximal w.r.t. D and σ; (iv) $(s\sigma \simeq t\sigma) \not\succeq (r\sigma \simeq h\sigma)$; (v) $t\sigma \not\succeq s\sigma$; (vi) $h\sigma \not\succeq r\sigma$ and (vii) s' is not a variable.

Negative Superposition

$$\texttt{NS}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{r}[\underline{\boldsymbol{s}'}] \not\simeq \boldsymbol{h}^{\star}}{C\sigma \vee D\sigma \vee r[t]\sigma \not\simeq h\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $r \not\simeq h$ is eligible strictly maximal w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$; (v) $h\sigma \not\succeq r\sigma$ and (vi) s' is not a variable.

Reflexivity Resolution

$$\texttt{RR}: \frac{C \vee \underline{\boldsymbol{s}} \not\simeq \underline{\boldsymbol{s}'}}{C\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$ and (ii) $s \not\simeq s'$ is eligible w.r.t. C and σ.

Equality Factoring

$$\texttt{EF}: \frac{C \vee \underline{s'} \simeq h \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star}}{C\sigma \vee t\sigma \not\simeq h\sigma \vee s'\sigma \simeq h\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. $C \vee s' \simeq h$ and σ, and (iii) $t\sigma \not\succeq s\sigma$.

System 10: The superposition calculus $\mathcal{SP}^{\succ}_{Sel}$

tion rule now we have three inference rules: the Ordered Paramodulation rule into non-equational literals, the Positive Superposition rule for paramodulation into the maximal term of positive equations and the Negative Superposition rule for paramodulation into the maximal term of negative equations.[10] Additionally we have a new inference rule Equality Factoring whose rôle will be revealed in a moment.

[10] Ordered Paramodulation into non-equational literals is often described as an instance of Positive Superposition or Negative Superposition when atoms are viewed as equations over two sorts of expressions

Note that a restriction of the superposition calculus to positive unit equational atoms is our well-known ordered Knuth-Bendix completion procedure.

In order to prove refutational completeness for the superposition calculus $\mathcal{SP}^{\succ}_{Sel}$, we modify the model construction given for the paramodulation calculus $\mathcal{OP}^{\succ}_{Sel}$. It seems like we only need to relax the condition (*iii*) from the definition of productive clause (see Definition 4.5), such that the *maximal non-equational expression* of the maximal atom A in a clause C is irreducible w.r.t. a current rewrite system. If we accept this definition, we obtain a rewrite system which has the following property:

Definition 4.11 (Canonical Rewrite System). A rewrite system I is *canonical (for superposition)* if for every rule $A = (E_1 \Rightarrow E_2) \in I$, the expression E_1 is irreducible w.r.t. I_A. ◈

An analog of Lemma 4.4 can be proven for rewrite systems that are canonical for superposition. For this we need to introduce additional notation. Given an expression E, let I^E denote the set of atoms from I whose expressions are not greater than E: $I^E := \{(E_1 \simeq E_2) \in I \mid E_1 \preceq E, E_2 \preceq E\}$. Then the following property holds:

Lemma 4.12 (Canonical Proofs). *Let I be a canonical rewrite system. Then (**i**) I is confluent and (**ii**) for every atom $A := (E_1 \simeq E_2)$ with $E_1 \succeq E_2$, we have $I\Downarrow \models A$ iff $I^{E_1}\Downarrow \models A$.*

Proof. The proof for case (*i*) of this lemma is identical to the one for Lemma 4.4. Case (*ii*) follows from the fact that every expression involved in a rewrite proof for $A := (E_1 \simeq E_2)$ is not greater then the maximal expression from $\{E_1, E_2\}$. ⧉

Unfortunately an analog of Lemma 4.7 does not hold if candidate models are constructed according to the modified procedure above. This is demonstrated in the following example:

N	*Clauses* C	ΔC
1.	$\mathtt{a} \not\simeq \mathtt{c} \vee \underline{\mathtt{a}} \not\simeq \mathtt{b}$	–
2.	$\mathtt{a} \simeq \mathtt{c} \vee \underline{\mathtt{a}} \simeq \mathtt{b}^{\star}$	$\mathtt{a} \Rightarrow \mathtt{b}$
3.	$\underline{\mathtt{b}} \simeq \mathtt{c}^{\star}$	$\mathtt{b} \Rightarrow \mathtt{c}$

Example 4.13. Let $\succ$ be an ordering such that $\mathtt{a} \succ \mathtt{b} \succ \mathtt{c}$. Consider a clause set N shown in the table to the right. Clauses 1 – 3 from N are arranged in decreasing order w.r.t. $\succ$. Clause 3 produces a rewrite rule $\mathtt{b} \Rightarrow \mathtt{c}$. Although the maximal atom $\mathtt{a} \simeq \mathtt{b}$ of clause 2 is reducible w.r.t. to this rewrite rule, its maximal term $\mathtt{a}$ is not. Hence, clause 2 produces a rewrite rule $\mathtt{a} \Rightarrow \mathtt{b}$. But now the remaining literal $\mathtt{a} \simeq \mathtt{c}$ of this clause becomes `true` in I^2 and consequently in I_N. Hence, the important property (*ii*) of candidate models (see Lemma 4.7) is violated (if this property does not hold, then an inference between a productive clause and a counterexample might produce a clause that is `true` in a candidate model).

This example also demonstrates why the Equality Factoring rule is needed. It is easy to see that the clause set N is unsatisfiable. Indeed, the empty clause can be derived from N in the paramodulation calculus: clauses 2, 3 produce $\mathsf{a} \simeq \mathsf{c}$ and clauses 1, 3 produce $\mathsf{a} \not\simeq \mathsf{c} \vee \mathsf{a} \not\simeq \mathsf{c}$ by applying Ordered Paramodulation inferences into b and the Ordered Factoring rule. The rest inferences are straightforward. However the only possible superposition inferences from N are Negative Superposition between clause 2 and 1 and Equality Factoring from 2. The first inference produces a tautology. Hence, an empty clause cannot be derived from N without the Equality Factoring rule if deletion of tautoloties is allowed.[11] ⋘

In order to retain the properties of productive clauses, we modify the notion of productive clause such that the situation described in Example 4.13 becomes not possible:

Definition 4.14 (Productive Clause). A clause C is *productive clause* w.r.t. set of atoms I, if (***i***) $I \Downarrow \not\models C$; (***ii***) $C = C' \vee A$, where $A = (E_1 \simeq E_2)$ is an eligible strictly greatest atom w.r.t. C' with $E_1 \succ E_2$; (***iii***) E_1 is irreducible w.r.t. I and (***iv***) $(I \cup \{E_1 \Rightarrow E_2\}) \Downarrow \not\models C'$. In this case C *produces* a rewrite rule $\Delta^I C := (E_1 \Rightarrow E_2)$ w.r.t. I. If C is not productive, then we assign $\Delta^I C := \{\}$. ⋘

For proving remaining technical lemmas, we need two additional requirements for orderings which may be used in the superposition calculus.

Definition 4.15 (Admissible Order). An ordering $\succ$ is *admissible for superposition* if $\succ$ is admissible for paramodulation (see Definition 4.1) and addittionally:

(**E3**) $s \succ t \succ h$ implies $(s \not\simeq h) \succ (s \simeq t)$, and
(**E4**) $s \succ t \succ h$ implies $(s \simeq t) \succ (t \not\simeq h)$. ⋘

Condition (E3) for equational atoms plays a similar role as condition (R1) for non-equational atoms. Condition (E4) is needed to ensure monotonicity of the Equality Factoring rule:

Lemma 4.16 (Monotonicity for $\mathcal{SP}^{\succ}_{\mathbf{Sel}}$). *$\mathcal{SP}^{\succ}_{Sel}$ is monotone for ground clauses w.r.t. every admissible ordering $\succ$.*

Proof. Ordered Paramodulation, Positive Superposition and Negative Superposition are restrictions of the Ordered Paramodulation rule from the paramodulation calculus, which is proved to be monotone. The Reflexivity Resolution rule is left unchanged. It remains to show monotonicity of the Equality Factoring rule.

[11] However, this example does not imply that the Equality Factoring rule is strictly necessary when tautologies are *not* deleted. In fact, Bachmair & Ganzinger [1997] proved that a variant of superpostion calculus without Equality Factoring called the *strict superposition calculus*, remains complete. To prove this, they formulated a waker notion of redundancy and used *basic strategies* in order to avoid superposition into variables

(**1**) $\texttt{EF} : C \vee \underline{s} \simeq h \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^\star \vdash C \vee t \not\simeq h \vee s \simeq h$ (Equality Factoring):
From conditions (*ii*) and (*iii*) of this rule, we have $(s \simeq t) \succ (s \simeq h)$, and $s \succ t$, which implies (by totality and monotonicity of $\succ$) that $t \succ h$. Using condition (E4) of admissible orderings (see Definition 4.15), we obtain: $(s \simeq t) \succ (t \not\simeq h)$. Hence, $(C \vee s \simeq h \vee s \simeq t) \succ (C \vee t \not\simeq h \vee s \simeq h)$ what was required to show. ⧉

Lemma 4.17 (Properties of Candidate Models). *Let N be a set of ground clauses. Then for every $C \in N$, (**i**) $I^C\Downarrow \vDash C$ implies that $I_N\Downarrow \vDash C$ and (**ii**) If $C = (C' \vee \boldsymbol{A}^\star)$ is productive w.r.t. I_C then $I_N\Downarrow \nvDash C'$.*

N	*Clauses* C	ΔC
C	$C' \vee \neg B$	?

Proof. (*i*) If $I^C\Downarrow \vDash A$ for some positive atom A in C, then $I_N\Downarrow \vDash A \vDash C$. Otherwise, C should contain negatively some atom $B = (E \simeq E')$ such that $I^C\Downarrow \nvDash B$, but $I_N\Downarrow \vDash B$. Since $E \neq E'$, w.l.o.g. we can assume that $E \succ E'$. Then by Lemma 4.12, $I^E\Downarrow \vDash B$. But for every atom $(E_1 \Rightarrow E_2) \in I^E$, we have $E \succeq E_1 \succ E_2$, and so $C \succeq \neg B = (E \not\simeq E') \succ (E_1 \simeq E_2)$ by conditions (E1) and (E3) of admissible orderings. Hence $I^E\Downarrow \vDash I_C\Downarrow \nvDash B$, which contradicts to $I^E\Downarrow \vDash B$.

(*ii*) Assume that for some productive clause $C = (C' \vee \boldsymbol{A}^\star)$, we have $I_N\Downarrow \vDash C'$. Since $I_C\Downarrow \nvDash C'$, there must be some positive atom B in C' that is `true` in $I_N\Downarrow$. Since $I_N\Downarrow$ is canonical for superposition, then by Lemma 4.12 we have $I^E \vDash B$, where E is the maximal non-equational expression in B. If B is a non-equational atom, then this situation is not possible, since $I^E\Downarrow = I^B\Downarrow \subseteq I^C\Downarrow \nvDash C'$. So $B = (s \simeq h)$ for some ground terms $s \succ h$ ($s \neq h$, since $I_C\Downarrow \nvDash B$).

N	*Clauses* C	ΔC
C	$\overbrace{C'' \vee B}^{C'} \vee \boldsymbol{A}^\star$	A

Since $I^s\Downarrow \vDash B$ but $I_C\Downarrow \nvDash B$, there is a minimal productive clause of the form $D_1 = (D_1' \vee (s \simeq t)) \succeq C$ such that $I_1^D\Downarrow \vDash B$. Since $B = (s \simeq h) \prec A \preceq (s \simeq t)$, then A must be an atom of the form $(s \simeq t')$, with $s \succ t'$. But the term s in $(s \simeq t)$ must be irreducible w.r.t. I_{D_1} by condition (*iii*) of productive clause (see Definition 4.14). This is only possible when $D_1 = C$. So, $I^C\Downarrow = (I_C \cup \{s \Rightarrow t'\})\Downarrow \vDash B = (s \simeq h) \vDash C'$. This situation is not possible by condition (*iv*) of productive clause from Definition 4.14. ⧉

N	*Clauses* C	ΔC
D_1	$D' \vee \boldsymbol{s} \simeq \boldsymbol{t}^\star$	$s \Rightarrow t$
⋎		
C	$C'' \vee \overbrace{s \simeq h}^{B} \vee \overbrace{\boldsymbol{s} \simeq \boldsymbol{t}'}^{\boldsymbol{A}} {}^\star$	$s \Rightarrow t'$

Lemma 4.18 (Counterexample-Reduction Lemma). *Let N be a set of ground clauses which is saturated w.r.t. $\mathcal{SP}^{\succ}_{Sel}$ up to redundancy. Then for every clause $D \in N$ with $I_N\Downarrow \nvDash D \neq \square$, there exists $D_1 \in N$ such that $D_1 \prec D$ and $I_N\Downarrow \nvDash D_1$.*

Proof. Let $I_N\Downarrow \nvDash D$ for some clause $D \in N$. Then $I^D\Downarrow \nvDash D$, since otherwise by Lemma 4.17 (*i*) we would have $I_N\Downarrow \vDash D$. In particular, clause D cannot be productive. So, we have $I_D\Downarrow = I^D\Downarrow \nvDash D$. According to the definition of productive clauses (see Definition 4.14), this situation is possible only in the following cases:

(**1**) $D = (D' \vee \neg \boldsymbol{A})$ for some eligible literal $\neg A$ in D, and $I_D{\Downarrow} \models A = (E_1 \simeq E_2)$. W.l.o.g. assume that $E_1 \succeq E_2$. Then either (**1.1**) $E_1 = E_2 = s$ for some ground term s, or (**1.2**) $A = (P \simeq \mathtt{T})$ is a non-equational atom and P is reducible by some rewrite rule $(E_1' \Rightarrow E_2') \in I_D$, or (**1.3**) $E_1 = r[s]$, $E_2 = h$ and there is a rewrite rule $(s \Rightarrow t) \in I_D$ produced by some clause $C = (C' \vee s \simeq t)$.

In case (1.1), it is possible to reduce counterexample D by applying the Reflexivity Resolution rule, in case (1.2) an Ordered Resolution inference or an Ordered Paramodulation inference is possible similarly as for the paramodulation calculus.

In case (1.3) it is possible to apply the Negative Superposition rule to clauses C and D that produces a clause $\mathtt{NS}[C, D]$: $D_1 = (C' \vee D' \vee r[t] \not\simeq h) \prec D$. We must show that $I_N{\Downarrow} \not\models D_1$. Indeed, $I_N{\Downarrow} \not\models D'$ since $I_N{\Downarrow} \not\models D$ and $I_N{\Downarrow} \not\models C'$ by Lemma 4.17 (ii). If $I_N{\Downarrow} \models (r[t] \simeq h)$, then $I_N{\Downarrow} \models (r[s] \simeq h)$, which is also not possible.

N	*Clauses* C	ΔC
D	$D' \vee \boldsymbol{r[s] \not\simeq h^\star}$	$-$
$\curlyvee$		
D_1	$C' \vee D' \vee r[t] \not\simeq h$	?
C	$C' \vee \boldsymbol{s \simeq t^\star}$	$s \Rightarrow t$

(**2**) $D = (D' \vee \boldsymbol{A})$, where $A = (E_1 \simeq E_2)$ is an eligible greatest literal w.r.t. D', $E_1 \succ E_2$ and $(I_D \cup \{E_1 \Rightarrow E_2\}){\Downarrow} \models D'$. Then there exists an atom $B = (E_1' \simeq E_2')$ from D' whose rewrite proof uses $E_1 \Rightarrow E_2$. W.l.o.g., assume that $E_1' \succeq E_2'$. Then $E_1' = E_1$, since otherwise, by Lemma 4.12 (ii), $I_D{\Downarrow} \models I^{E_1'}{\Downarrow} \models B$. Then either (**2.1**) $A = B$ are non-equational atoms, or (**2.2**) $A = (s \simeq t)$ and $B = (s \simeq h)$ for some ground terms $s \succ t$ and $s \succ h$ and $I_D{\Downarrow} \models (t \simeq h)$.

In case (2.1) the Ordered Factoring rule can be applied to D which reduces this counterexample. In case (2.2) the Equality Factoring rule can be applied to D which produces a clause $\mathtt{EF}[D]$: $D_1 = (D' \vee t \not\simeq h \vee s \simeq h) \prec D$. It remains to show that $I_N{\Downarrow} \not\models D_1$. Indeed, $I_N{\Downarrow} \not\models (t \not\simeq h)$, since $I_N{\Downarrow} \models I_D{\Downarrow} \models (t \simeq h)$, and $I_N{\Downarrow} \not\models (D' \vee s \simeq h)$ since $I_N{\Downarrow} \not\models D$. ⧉

N	*Clauses* C	ΔC
D	$D' \vee s \simeq h \vee \boldsymbol{s \simeq t^\star}$	$-$
$\curlyvee$		
D_1	$D' \vee t \not\simeq h \vee s \simeq h$	?

Remark 4.19. Note that the Equality Factoring rule is applied in case (2.1) only if $I_D{\Downarrow} \models (t \simeq h)$. Since $t \succ h$, then the term t must be reducible in $I_D{\Downarrow}$. In this situation it can be shown that the counterexample can be reduced using an alternative inference rule called Merging Paramodulation [Bachmair & Ganzinger, 1990, 1994] formulated in Figure 13. ⋘

Lifting of the superposition calculus to general clauses is analogous to the paramodulation calculus. Summarising, we have proved completeness of the superposition calculus $\mathcal{SP}^{\succ}_{Sel}$ (both with Equality Factoring or with Merging Paramodulation):

Theorem 4.20 (Completeness of $\boldsymbol{\mathcal{SP}^{\succ}_{Sel}}$ with Redundancy). $(\mathcal{SP}^{\succ}_{Sel}, \mathsf{R}^{\boldsymbol{s}\succ})$ *is complete.*

An extension of superposition rules with simultaneous inferences can be justified similarly as for the paramodulation calculus: see Figure 14.

Figure 13 The merging paramodulation rule

Merging Paramodulation

$$\texttt{MP}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \underline{u} \simeq v \vee \underline{\boldsymbol{r}} \simeq \boldsymbol{h}[\underline{\boldsymbol{s}'}]^{\star}}{C\sigma \vee D\sigma \vee r\sigma \simeq h[t]\sigma \vee u\sigma \simeq v\sigma}$$

where (i) $\sigma = mgu(\{s{=}s', r{=}u\})$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $r \simeq h$ is eligible strictly maximal w.r.t. $D \vee u \simeq v$ and σ; (iv) $t\sigma \not\succeq s\sigma$; (v) $h\sigma \not\succeq r\sigma$ and (vi) s' is not a variable.

Figure 14 The simultaneous superposition rules

(Simulteneous) Positive Superposition

$$\texttt{PS}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D[\underline{s'}] \vee \boldsymbol{r}[\underline{\boldsymbol{s}'}] \simeq \boldsymbol{h}[\underline{\boldsymbol{s}'}]^{\star}}{C\sigma \vee D[t]\sigma \vee r[t]\sigma \simeq h[t]\sigma}$$

where (i) $\sigma = mgu(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $r \simeq h$ is eligible strictly maximal w.r.t. D and σ; (iv) $(s\sigma \simeq t\sigma) \not\succeq (r\sigma \simeq h\sigma)$; (v) $t\sigma \not\succeq s\sigma$; (vi) $h\sigma \not\succeq r\sigma$ and (vii) s' is not a variable.

(Simulteneous) Negative Superposition

$$\texttt{NS}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D[\underline{s'}] \vee \boldsymbol{r}[\underline{\boldsymbol{s}'}] \not\simeq \boldsymbol{h}[\underline{\boldsymbol{s}'}]^{\star}}{C\sigma \vee D[t]\sigma \vee r[t]\sigma \not\simeq h[t]\sigma}$$

where (i) $\sigma = mgu(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $r \not\simeq h$ is eligible strictly maximal w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$; (v) $h\sigma \not\succeq r\sigma$ and (vi) s' is not a variable.

5 Chaining Calculi

The ordered paramodulation and ordered resolution calculi are examples of calculi *modulo theory*, in these cases – the theory of equality. The advantage of calculi with built-in equality over direct treatment of equational axioms, is that the proof search for the first is restricted to a relatively narrow class of proofs, namely the *rewrite proofs*. This basic idea can be further extended to capture some theories that are weaker than equational theories. Below, we revisit specialised *chaining calculi* for transitive and compositional binary relations that have been introduced by Bachmair & Ganzinger [1995, 1998*b*]. Using an extension of the model construction method from the previous sections, we prove completeness of these calculi for a quite general class of orderings, which will play essential rôle in the decision procedures that we are going to present. Among other things, we will discuss necessity for special Transitivity Factoring and Compositional Resolution rules in chaining calculi which can be seen as non-symmetric analogs of the Equality Factoring rule. We also describe a particular hyper-inference strategy, and extend the standard redundancy criterion to suit a larger class of compositional theories.

5.1 Reasoning with Transitive Relations

In calculi with built-in theory, we have two types of signature elements: the *interpreted (or special) symbols* whose interpretation is restricted in a theory, and the remaining *free symbols*. In chaining calculi we will have only binary predicate symbols as special symbols. We have already used one special predicate symbol, namely equality $\simeq$, which can be easily distinguished from others because of its *infix notation* $s \simeq t$, i.e., when the predicate symbol is situated between the arguments. We continue this tradition, and will use the *infix notation* sSt for all special binary predicate symbols S. We say *special atoms* or *special literals* for those atoms or literals that contain special predicate symbols.

Since we will talk about calculi with build-in theories, we will do all logical reasoning modulo theory as well. We say that a clause (set) is $\boldsymbol{T}$*-satisfiable* (here $\boldsymbol{T}$ denotes *a theory*), if it has a $\boldsymbol{T}$*-model*, i.e., a model which admits all axioms of $\boldsymbol{T}$. The *entailment relation modulo theory* is denoted by $\models_{\boldsymbol{T}}$. The notion of standard redundancy based on semantical entailment is also extended to theories. We say that a ground clause C^0 (or a ground inference π^0) is $\boldsymbol{T}$*-redundant* w.r.t. a set of ground clauses N^0, if $N^0{}_{C^0} \models_{\boldsymbol{T}} C^0$ (respectively, $N^0{}_{C_1^0} \models_{\boldsymbol{T}} C^0$, where C_1^0 is the maximal premise of π^0). Redundancy of non-ground clauses and inferences is defined as usual, by considering their eligible ground instances. It will be usually the case that a theory is known from the context, so we will often say "*redundant*" instead of "$\boldsymbol{T}$*-redundant*".

In this section we consider a *theory of transitivity* $\mathcal{T}$ in which some special predicates T are restricted to admit the transitivity property:

$$xTy \wedge yTz \rightarrow xTz \qquad \textit{(Transitivity)} \tag{22}$$

Equality is a transitive symmetric reflexive relation that admits monotonicity property. A transitive relation should behave similar to equality, except that it is neither symmetric, nor reflexive or monotone. Using this observation, we try to obtain a specialised calculus for transitive relation from the superposition calculus, by removing symmetry, reflexivity and monotonicity of the equality predicate $\simeq$ from this calculus.

By modifying the Positive Superposition and Negative Superposition inference rules we arrive to the inference rules given in Figure 15. The Ordered Chaining rule is obtained from the Positive Superposition rule, by removing symmetry and monotonicity of equality, so that a sound inference rule for transitive relations remains. The Negative Chaining rule is similarly obtained from the Negative Superposition rule, however, now we have two variants of rules since the transitive relation might not be symmetric and we must distinguish chaining into the left argument from the chaining into the right argument.

Figure 15 Ordered chaining rules for transitive relations

Ordered Chaining

$$\mathtt{OC}: \frac{C \vee \boldsymbol{tT\underline{s}}^{\star} \quad D \vee \underline{\boldsymbol{s'Tt'}}^{\star}}{C\sigma \vee D\sigma \vee t\sigma Tt'\sigma}$$

where (i) $\sigma = mgu(s, s')$; (ii) tTs is eligible strictly maximal w.r.t. C and σ; (iii) $s'Tt'$ is eligible strictly maximal w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$ and (v) $t'\sigma \not\succeq s'\sigma$.

Negative Chaining

$$\mathtt{NC}: \frac{C \vee \underline{\boldsymbol{sTt}}^{\star} \quad \neg(\underline{\boldsymbol{s'Th}}) \vee D}{C\sigma \vee \neg(t\sigma Th\sigma) \vee D\sigma} \qquad \frac{C \vee \boldsymbol{tT\underline{s}}^{\star} \quad \neg(\boldsymbol{hT\underline{s'}}) \vee D}{C\sigma \vee \neg(h\sigma Tt\sigma) \vee D\sigma}$$

where (i) $\sigma = mgu(s, s')$; (ii) sTt is eligible strictly maximal w.r.t. C and σ; (iii) $\neg(s'Th)$ is eligible w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$ and (v) $h\sigma \not\succeq s'\sigma$.

where (i) $\sigma = mgu(s, s')$; (ii) tTs is eligible strictly maximal w.r.t. C and σ; (iii) $\neg(hTs')$ is eligible w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$ and (v) $h\sigma \not\succ s'\sigma$.

The Reflexivity Resolution rule does not have any counterpart for transitive relation, since they are not necessary reflexive. The Equality Factoring rule can be easily modified into a Transitivity Factoring rule for transitive relations: see Figure 16. The chaining calculus in this form has been considered, for instance

Figure 16 The chaining calculus with the Transitivity Factoring rule is incomplete

Transitivity Factoring

$$\mathtt{TF}: \frac{C \vee \underline{s}Tt \vee \underline{\boldsymbol{s'Th}}}{C\sigma \vee \neg(h\sigma Tt\sigma) \vee s\sigma Tt\sigma} \qquad \frac{C \vee tT\underline{s} \vee \boldsymbol{hT\underline{s'}}}{C\sigma \vee \neg(t\sigma Th\sigma) \vee t\sigma Ts\sigma}$$

where (i) $\sigma = mgu(s, s')$; (ii) sTt is eligible strictly maximal w.r.t. $C \vee \neg(s'Th)$ and σ, and (iii) $t\sigma \not\succeq s\sigma$.

where (i) $\sigma = mgu(s, s')$; (ii) tTs is eligible strictly maximal w.r.t. $C \vee \neg(hTs')$ and σ, and (iii) $t\sigma \not\succeq s\sigma$.

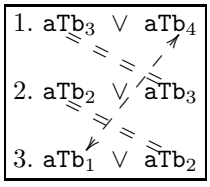
in [Struth, 2001], where this calculus has been derived syntactically by analysing resolution proofs involving transitivity axioms. Unfortunately this calculus is not compatible with tautology deletion[12] (it seems to be that this was not known before):

Example 5.1. Consider a set N consisting of clauses 1 – 8, where $\mathtt{T}$ is a transitive predicate symbol and $\mathtt{a} \succ \mathtt{b}_4 \succ \mathtt{b}_3 \succ \mathtt{b}_2 \succ \mathtt{b}_1$.

N:
1. $\mathtt{aTb}_3 \vee \underline{\mathbf{a}}\mathbf{Tb_4}^{\star}$
2. $\mathtt{aTb}_2 \vee \underline{\mathbf{a}}\mathbf{Tb_3}^{\star}$
3. $\mathtt{aTb}_1 \vee \underline{\mathbf{a}}\mathbf{Tb_2}^{\star}$
4. $\neg(\mathtt{aTb}_3) \vee \neg(\underline{\mathbf{a}}\mathbf{Tb_4})^{\star}$
5. $\neg(\mathtt{aTb}_2) \vee \neg(\underline{\mathbf{a}}\mathbf{Tb_3})^{\star}$
6. $\neg(\mathtt{aTb}_1) \vee \neg(\underline{\mathbf{a}}\mathbf{Tb_2})^{\star}$
7. $\mathbf{b_1}\mathbf{T}\underline{\mathbf{b_4}}^{\star}$
8. $\underline{\mathbf{b_4}}\mathbf{Tb_1}^{\star}$

[12] Contrary to what has been claimed in [Ganzinger et al., 2001], possibly by a mistake

First note that this clause set is unsatisfiable. Indeed, suppose there exists a $\mathcal{T}$-model $\mathcal{M}$ for N (i.e., where $\mathtt{T}$ is interpreted by a transitive relation). Then at least one literal from each clause 1 – 3 must be $\mathtt{true}$ in $\mathcal{M}$. Since all atoms 7 and 8 are also $\mathtt{true}$ in $\mathcal{M}$, then we have $\mathcal{M} \models_{\mathcal{T}} \mathtt{aTb}_1$ *iff* $\mathcal{M} \models_{\mathcal{T}} \mathtt{aTb}_4$. By simple combinatorial analysis, one can observe that for some clause from 1 – 3, *both* literals must be $\mathtt{true}$ in $\mathcal{M}$ (the figure to the right shows which literals should be simultaneously $\mathtt{true}$ or $\mathtt{false}$ in $\mathcal{M}$). However this is not possible because of clauses 4 – 6.



Although the clause set N is unsatisfiable, a contradiction cannot be derived in the chaining calculus with the Transitivity Factoring rule. To demonstrate it in a simple way, we add clauses 9 – 18 to N that express that the remaining pairs of constants $\mathtt{b}_1 - \mathtt{b}_4$ are *not* related to each other by the transitive relation (see the table to the right). Obviously, after we add these clauses, the clause set N remains unsatisfiable.

9. $\neg(\underline{\mathtt{b_4}}\mathtt{Tb_3})^\star$	14. $\neg(\mathtt{b_3}\mathtt{T}\underline{\mathtt{b_4}})^\star$
10. $\neg(\underline{\mathtt{b_4}}\mathtt{Tb_2})^\star$	15. $\neg(\mathtt{b_2}\mathtt{T}\underline{\mathtt{b_4}})^\star$
11. $\neg(\underline{\mathtt{b_3}}\mathtt{Tb_2})^\star$	16. $\neg(\mathtt{b_2}\mathtt{T}\underline{\mathtt{b_3}})^\star$
12. $\neg(\underline{\mathtt{b_3}}\mathtt{Tb_1})^\star$	17. $\neg(\mathtt{b_1}\mathtt{T}\underline{\mathtt{b_3}})^\star$
13. $\neg(\underline{\mathtt{b_2}}\mathtt{Tb_1})^\star$	18. $\neg(\mathtt{b_1}\mathtt{T}\underline{\mathtt{b_2}})^\star$

The Ordered Chaining rule can be applied only to clauses 7 and 8 which produces a new clause 19. The Negative Chaining rule is possible between clauses 1 – 3 and clauses 4 – 6, between clause 8 and clauses 9, 10, and between clause 7 and clauses 14, 15. It is easy to see that the conclusion of every such inference is either a tautology (for inferences $\mathtt{NC}[1,4]$, $\mathtt{NC}[2,5]$ and $\mathtt{NC}[3,6]$), or is subsumed by (and redundant w.r.t.) some clause from 9 – 18. The only possible Ordered Resolution inferences are $\mathtt{OR}[1,4]$, $\mathtt{OR}[2,5]$ and $\mathtt{OR}[3,6]$ that produce tautologies as well. The Ordered Factoring rule cannot be applied to N. It is possible to apply the Transitivity Factoring rule to clauses 1 – 3, but its conclusions will be also subsumed by clauses 9 – 18. Hence, the empty clause cannot be derived from N using the inference rules listed above, when deletion of tautologies and subsumed clauses is allowed.

$\mathtt{OC}[7,8]$: 19. $\mathtt{b_1}\mathtt{Tb_1}^\star$

We stress, that the reason of incompleteness is caused here only by tautology deletion as an instance of redundancy elimination techniques[13], since it was shown in [Bachmair & Ganzinger, 1997] that the chaining calculus with a weaker notion of redundancy is complete even without the Transitivity Factoring rule. ⋘

Note that Example 5.1 does not work, if the relation $\mathtt{T}$ is additionally symmetric (like equality). In this case, additional chaining inferences between clauses 1 – 3 are possible, which allow one to derive the empty clause. In fact, chaining with the Transitivity Factoring rule is complete with the standard redundancy criterion for *partial equivalences*, i.e., for transitive and symmetric relations.

[13]It is possible to show that a slightly longer saturation of clauses 1 – 8 does not contain the empty clause when only tautologies are deleted. So subsumption deletion is not the reason of incompleteness for the chaining calculus with Transitivity Factoring

Refutational completeness for the chaining calculus with the standard redundancy criterion can be restored using the Transitivity Resolution rule proposed in [Bachmair & Ganzinger, 1995, 1998*b*], which is given in Figure 17. Transitivity

Figure 17 The Transitivity Resolution rule

Transitivity Resolution

$$\mathtt{TR}: \frac{C \vee \underline{s}Th \vee \underline{\boldsymbol{s''Tt'}}^{\star} \quad D \vee \underline{\boldsymbol{s'Tt}}^{\star}}{D\sigma \vee \neg(t\sigma Th\sigma) \vee s\sigma Th\sigma} \qquad \frac{C \vee hT\underline{s} \vee \underline{\boldsymbol{t'Ts''}}^{\star} \quad D \vee \underline{\boldsymbol{tTs'}}^{\star}}{D\sigma \vee \neg(h\sigma Tt\sigma) \vee h\sigma Ts\sigma}$$

where (i) $\sigma = mgu(\{s{=}s', s{=}s''\})$; (ii) $s''Tt'$ is eligible strictly maximal w.r.t. $C \vee sTh$ and σ; (iii) $s'Tt$ is eligible strictly maximal w.r.t. D and σ; (iv) $s''\sigma Tt'\sigma \not\succ s'\sigma Tt\sigma$; (v) $t\sigma \not\succeq s'\sigma$ and (vi) $h\sigma \not\succeq s\sigma$.

where (i) $\sigma = mgu(\{s{=}s', s{=}s''\})$; (ii) $t'Ts''$ is eligible strictly maximal w.r.t. $C \vee hTs$ and σ; (iii) tTs' is eligible strictly maximal w.r.t. D and σ; (iv) $t'\sigma Ts''\sigma \not\succ t\sigma Ts'\sigma$; (v) $t\sigma \not\succeq s'\sigma$ and (vi) $h\sigma \not\succeq s\sigma$.

Resolution rule is a sound inference rule that represents a controlled application of resolution between the second premise of the rule $D \vee \underline{\boldsymbol{s'Tt}}^{\star}$ and a transitivity clause $\neg(\underline{\boldsymbol{sTt}}) \vee \neg(tTh) \vee sTh$ (the right variant of this rule is obtained by swapping the arguments of T). Note that the conclusion of the rule is a $\mathcal{T}$-consequence of the right premise of this rule only. Note also, that an application of the Transitivity Factoring rule to clause $C = (C \vee \underline{s}Tt \vee \underline{\boldsymbol{s'Th}})$ can be simulated a self-application of the Transitivity Resolution rule (when both premises are C) followed by deletion of the *duplicate literal* $s\sigma Th\sigma$.

Continuing Example 5.1, the empty clause from clauses 1 – 8 can be now derived using the Transitivity Resolution rule as follows:

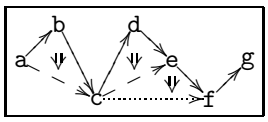
$\mathtt{TR}[3,1]$: 9. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \mathtt{aTb}_1 \vee \mathbf{\mathtt{aTb}_3}^{\star}$	$\mathtt{OR}[14,6]$: 15. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \neg(\mathtt{b}_1\mathtt{Tb}_4) \vee \mathbf{\neg\mathtt{aTb}_1}$
$\mathtt{OR}[9,5]$: 10. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \mathtt{aTb}_1 \vee \mathbf{\neg\mathtt{aTb}_2}$	$\mathtt{OR}[12,15]$: 16. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \mathbf{\neg\mathtt{b}_1\mathtt{Tb}_4}$
$\mathtt{OR}[3,10]$: 11. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \mathtt{aTb}_1 \vee \mathbf{\mathtt{aTb}_1}$	$\mathtt{OR}[7,16]$: 17. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \mathbf{\neg\mathtt{b}_4\mathtt{Tb}_1}$
$\mathtt{OF}[11]$: 12. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \underline{\mathtt{a}}\mathbf{\mathtt{Tb}_1}^{\star}$	$\mathtt{OR}[8,17]$: 18. $\mathbf{\neg\mathtt{b}_4\mathtt{Tb}_1}$
$\mathtt{NC}[12,4]$: 13. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \neg(\mathtt{b}_1\mathtt{Tb}_4) \vee \mathbf{\neg\mathtt{aTb}_3}$	$\mathtt{OR}[8,18]$: 19. $\square$
$\mathtt{OR}[2,13]$: 14. $\neg(\mathtt{b}_4\mathtt{Tb}_1) \vee \neg(\mathtt{b}_1\mathtt{Tb}_4) \vee \mathbf{\mathtt{aTb}_2}^{\star}$	

For proving completeness of chaining calculus, we adapt the model construction given for superposition calculus. Given a set of ground clauses N that is saturated under chaining calculus, we construct a model, which is represented by a set I_N of ground atoms, including those with transitive predicates. Similar to equality, we will read atoms involving transitive predicates as *rewrite rules*, however now they will be (***i***) *labelled* (or *sorted*), since we might possibly have several transitive predicates, and (***ii***) *oriented*, since now transitive predicates are not necessarily symmetric. Following [Bachmair & Ganzinger, 1995, 1998*b*], we say that each transitive atom sTt induces either a *left-to-right rewrite rule* $s \overset{T}{\Rightarrow} t$ when $s \succ t$, or a *right-to-left rewrite rule* $s \overset{T}{\Leftarrow} t$ when $s \prec t$, or a *two-way rewrite rule* $s \overset{T}{\Leftrightarrow} t$, when

$s = t$. Note that $(s \overset{T}{\Leftarrow} t) \neq (t \overset{T}{\Rightarrow} s)$. This correspondence is one-to-one between transitive atoms and rewrite proofs. From now on, we do not distinguish special atoms from the rewrite rules that they induce.

Given a set of atoms I, we say that an atom sTt has a *rewrite proof* in I (in symbols: $s \Downarrow_I^T t$) if either (***i***) $s \overset{T}{\Leftrightarrow} t \in I$, or (***ii***) there exists a non-empty sequence of rewrite steps based on I of the form $s = s_0 \overset{T}{\Rightarrow} s_1 \overset{T}{\Rightarrow} \cdots \overset{T}{\Rightarrow} s_k \overset{T}{\Leftarrow} \cdots \overset{T}{\Leftarrow} s_n = t$, where $0 \leq k \leq n$. A *rewrite model* $I\Downarrow$ induced by a set of atoms I consists of all atoms from I and all atoms that have a rewrite proof in I. The rewrite model $I\Downarrow$ does not necessarily satisfy transitivity axioms. Indeed, consider $I := \{tTs, sTh\}$ with $s \succ t$ and $s \succ h$, i.e., $I = \{t \overset{T}{\Leftarrow} s, s \overset{T}{\Rightarrow} h\}$. Then $I\Downarrow \vDash tTs$, $I\Downarrow \vDash sTh$ but $I\Downarrow \nvDash tTh$. A minimal $\mathcal{T}$-model in which all atoms from I are `true` must contain all atoms sTt that have a *chain proof* in I, i.e., a non-empty sequence of atoms $s_0Ts_1, s_1Ts_2, \ldots, s_{n-1}Ts_n$ from I, where $s = s_0$ and $s = s_n$. A rewrite proof is a chain proof that either consists of one *two-way* rewrite step, or is composed from *left-to-right* rewrite steps followed by *right-to-left* rewrite steps.

Obviously, not every chain proof is a rewrite proof. In the example above, the atom tTh has a chain proof $t \overset{T}{\Leftarrow} s \overset{T}{\Rightarrow} h$ in I, which is not a rewrite proof. Note that for this set I, it is possible to apply the Ordered Chaining rule to atoms $tT\underline{s}$ and $\underline{s}Th$ from I. If we augment I with the conclusion tTh of this inference, we obtain a rewrite proof for tTh.



A fragment of a chain proof of the form $t \overset{T}{\Leftarrow} s \overset{T}{\Rightarrow} h$ is called a *peak*. The peaks in chain proofs can be *eliminated* by replacing them with the conclusion of the Ordered Chaining rule. If all peaks in a chain proof are eliminated then we obtain a rewrite proof. Hence a saturation with the Ordered Chaining rule can be seen as an analog of Knuth-Bendix Completion procedure for transitive relations. It is easy to see that a variant of the Critical Pair Lemma holds for this *ordered chaining completion*, namely, if I is a set of atoms that is closed under the Ordered Chaining rule then every atom that has a chain proof in I, has a rewrite proof in I. In other words, if I is a set of atoms that is saturated in ordered chaining, then $I\Downarrow$ is a minimal $\mathcal{T}$-model in which all atoms from I are `true`.

5.2 Reasoning with Compositional Binary Relations

Before giving a formal proof for refutational completeness of the chaining calculus, we discuss a possibility of its extension to more general theories than the theory of transitivity. Transitivity axiom (22) can be seen as a *compositional law* for binary relations of form (23).

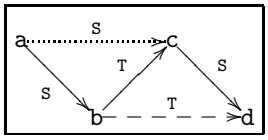
$$S \circ T \subseteq H \quad = \quad xSy \wedge yTz \rightarrow xHz \tag{23}$$

If $\boldsymbol{\mathcal{C}}$ is a set of compositional laws, one can consider a *compositional theory* (which we denote with the same latter), that satisfies all compositional axioms from $\boldsymbol{\mathcal{C}}$. We write $(S \circ T \subseteq H) \in \boldsymbol{\mathcal{C}}$ (or simply $S \circ T \subseteq H$ when $\boldsymbol{\mathcal{C}}$ is clear from the context) for compositional laws that belong to $\boldsymbol{\mathcal{C}}$. As usual, predicate symbols that are not free in the theory (i.e., they are involved in compositional axioms) are called *special predicate symbols*. Below we are concerned with the question: *to which extend the rewrite techniques can be adopted for such compositional theories?*

Example 5.2. Let $\boldsymbol{\mathcal{C}}$ be a theory induced by a set of compositional axioms given in the right table. Consider a set of ground atoms $I = \{\mathtt{aSb}, \mathtt{bTc}, \mathtt{cSd}\}$. By applying compositional axioms, it is easy to see that $I \models_{\boldsymbol{\mathcal{C}}} \mathtt{aSc}$, $I \models_{\boldsymbol{\mathcal{C}}} \mathtt{bTd}$, and so $I \models_{\boldsymbol{\mathcal{C}}} \mathtt{aSd}$ and $I \models_{\boldsymbol{\mathcal{C}}} \mathtt{aTd}$ (where $\models_{\boldsymbol{\mathcal{C}}}$ denotes entailment in $\boldsymbol{\mathcal{C}}$). Hence the sequence $\mathtt{aSb}, \mathtt{bTc}, \mathtt{cSd}$ can be seen as a chain proof for $\mathtt{aSd}$ and $\mathtt{aTd}$.

$\mathtt{S} \circ \mathtt{T} \subseteq \mathtt{S}$
$\mathtt{T} \circ \mathtt{S} \subseteq \mathtt{T}$
$\mathtt{S} \circ \mathtt{S} \subseteq \mathtt{T}$
$\mathtt{T} \circ \mathtt{T} \subseteq \mathtt{S}$

Now suppose that we have an ordering $\succ$ in which $\mathtt{a} \succ \mathtt{b}$ and $\mathtt{c} \succ \mathtt{b} \succ \mathtt{d}$. Then the chain proof given above has a peak $\mathtt{b} \overset{T}{\Leftarrow} \mathtt{c} \overset{S}{\Rightarrow} \mathtt{d}$: see the figure to the right. When we eliminate this peak by applying the correspondent compositional axiom, we obtain a rewrite proof $\mathtt{a} \overset{S}{\Rightarrow} \mathtt{b} \overset{T}{\Rightarrow} \mathtt{d}$. However, the last rewrite rule entails only atom $\mathtt{aSd}$. Hence after eliminating of peaks from the chain proof, we have lost the proof for atom $\mathtt{aTd}$. ◈



The reason for the failure of the completion procedure demonstrated in Example 5.2, is that the chain proof given in this example does *not* admit the *associativity property*: the result of composition of atoms in this proof depends on the order in which these atoms are composed. The sequence of relations $\mathtt{S} \circ \mathtt{T} \circ \mathtt{S}$ can be composed in two different ways: if we start from the first two relations we obtain $(\mathtt{S} \circ \mathtt{T}) \circ \mathtt{S} = \mathtt{S} \circ \mathtt{S} = \mathtt{T}$, if we start from the last two relations, we obtain $\mathtt{S} \circ (\mathtt{T} \circ \mathtt{S}) = \mathtt{S} \circ \mathtt{T} = \mathtt{S}$. For obtaining a correct result after eliminating peaks from chain proofs we require additional property to hold for compositional axioms for $\boldsymbol{\mathcal{C}}$:

$$(S \circ T) \circ H = S \circ (T \circ H) \qquad \textit{(Associativity Of Composition)} \qquad (24)$$

which formally means that *whenever* $(S \circ T \subseteq U) \in \boldsymbol{\mathcal{C}}$ *and* $(U \circ H \subseteq W) \in \boldsymbol{\mathcal{C}}$ *then there exists a special predicate symbol* V *such that* $(T \circ H \subseteq V) \in \boldsymbol{\mathcal{C}}$ *and* $(S \circ V \subseteq W) \in \boldsymbol{\mathcal{C}}$.

Most compositional theories are associative. The theory of transitivity is an associative compositional theory. Bachmair & Ganzinger [1995, 1998*b*] give an example of useful associative compositional axioms which go beyond transitivity:

$$\begin{array}{lll} \prec \circ \prec \subseteq \prec; & \precsim \circ \precsim \subseteq \precsim; & \sim \circ \sim \subseteq \sim; \\ \prec \circ \precsim \subseteq \prec; & \prec \circ \sim \subseteq \prec; & \precsim \circ \sim \subseteq \precsim; \\ \precsim \circ \prec \subseteq \prec; & \sim \circ \prec \subseteq \prec; & \sim \circ \precsim \subseteq \precsim. \end{array} \qquad (25)$$

For the particular axioms above it is especially easy to check the associativity property. Let $\gg$ be a precedence on special predicate symbols such that $\succ \gg \succsim \gg \sim$. Then the compositional axioms above can be characterised as follows:

Definition 5.3. Given a precedence $\gg$ on special predicate symbols, we say that a compositional theory $\boldsymbol{\mathcal{C}}$ is *induced by* $\gg$ if $\boldsymbol{\mathcal{C}} = \{S \circ T \subseteq H \mid H = max_{\gg}(S,T)\}$. ⋘

All compositional axioms in (25) are induced by the precedence $\gg$ given above. Every compositional theory that is induced by a precedence $\gg$, is evidently associative, and composition of several special relations can be computed as follows:

$$S_1 \circ \cdots \circ S_n \subseteq S \quad \textit{iff} \quad S = max_{\gg}(S_1,..,S_n).$$

Bachmair & Ganzinger [1998*b*] have restricted the class of allowed compositional axioms to those induced by a total precedence on special predicates. The model construction and redundancy elimination will work if one imposes a weaker restrictions on compositional axioms:

Definition 5.4. We say that a set $\boldsymbol{\mathcal{C}}$ of compositional axioms is *compatible with a precedence* $\gg$ on special predicates if $(S \circ T \subseteq H) \in \boldsymbol{\mathcal{C}}$ implies that $S \gtrdot H$ or $T \gtrdot H$. ⋘

However, in this report *we do not impose any of these restrictions*, since there are many interesting compositional theories that do not enjoy this property, like the one given in the example below. Unfortunately, this will require certain modification of standard redundancy for the chaining calculus.

Example 5.5. Consider a theory consisting of special binary predicate symbols $\mathtt{P}_i$ for every natural i with $0 \leq i < p$ which admit the following compositional axioms:

$$\mathtt{P}_n \circ \mathtt{P}_m \subseteq \mathtt{P}_{(n+m \bmod p)} \tag{26}$$

for every n and m with $0 \leq n < p$ and $0 \leq m < p$. It is easy to check that this compositional theory is associative. Indeed, $\mathtt{P}_{n_1} \circ \cdots \circ \mathtt{P}_{n_k} \subseteq \mathtt{P}_{(n_1+\cdots+n_k \bmod p)}$ does not depend on the order in which these compositional axioms are applied.

However, if $p > 1$, then there is no total precedence $\gg$ on special predicate symbols that is compatible with all compositional axioms. Indeed, assume that $\gg$ is such a precedence and $\mathtt{P}_m$ is the maximal special predicate symbol w.r.t. to this precedence. Then $\mathtt{P}_{(m-1 \bmod p)} \circ \mathtt{P}_{(m+1 \bmod p)} \subseteq \mathtt{P}_m$, hence it is not possible that $\mathtt{P}_{(m-1 \bmod p)} \gtrdot \mathtt{P}_m$ or $\mathtt{P}_{(m+1 \bmod p)} \gtrdot \mathtt{P}_m$ unless either $(m-1 = m \bmod p)$, or $(m+1 = m \bmod p)$, which is only possible when $p = 1$. ⋘

The ordered chaining calculus for compositional binary relations $\mathcal{OC}^{\succ}_{Sel}$ is formulated in System 11. In all inference rules we assume that S, T and H are special

Ordered Chaining

$$\mathtt{OC}: \frac{C \vee \boldsymbol{tS\underline{s}}^{\star} \quad D \vee \boldsymbol{\underline{s}'Tt'}^{\star}}{C\sigma \vee D\sigma \vee t\sigma Ht'\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) tSs is eligible strictly maximal w.r.t. C and σ; (iii) $s'Tt'$ is eligible strictly maximal w.r.t. D and σ; (iv) $t\sigma \not\succ s\sigma$; (iv)$'$ $t\sigma \neq s\sigma$ if $H = T$; (v) $t'\sigma \not\succ s'\sigma$ and (v)$'$ $t'\sigma \neq s'\sigma$ if $H = S$.

Negative Chaining

$$\mathtt{NC}: \frac{C \vee \boldsymbol{\underline{s}St}^{\star} \quad \boldsymbol{\neg(\underline{s}'Hh)} \vee D}{C\sigma \vee \neg(t\sigma Th\sigma) \vee D\sigma} \qquad \frac{C \vee \boldsymbol{tT\underline{s}}^{\star} \quad \boldsymbol{\neg(hH\underline{s}')} \vee D}{C\sigma \vee \neg(h\sigma St\sigma) \vee D\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) sSt is eligible strictly maximal w.r.t. C and σ; (iii) $\neg(s'Hh)$ is eligible w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$ and (v) $h\sigma \not\succeq s'\sigma$.

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) tTs is eligible strictly maximal w.r.t. C and σ; (iii) $\neg(hHs')$ is eligible w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$ and (v) $h\sigma \not\succ s'\sigma$.

Compositional Resolution

$$\mathtt{CR}: \frac{C \vee \underline{s}Hh \vee \boldsymbol{\underline{s}''S't'}^{\star} \quad D \vee \boldsymbol{\underline{s}'St}^{\star}}{D\sigma \vee \neg(t\sigma Th\sigma) \vee s\sigma Hh\sigma} \qquad \frac{C \vee hH\underline{s} \vee \boldsymbol{t'S'\underline{s}''}^{\star} \quad D \vee \boldsymbol{tT\underline{s}'}^{\star}}{D\sigma \vee \neg(h\sigma St\sigma) \vee h\sigma Hs\sigma}$$

where (i) $\sigma = \mathrm{mgu}(\{s{=}s', s{=}s''\})$; (ii) $s''S't'$ is eligible strictly maximal w.r.t. $C \vee sHh$ and σ; (iii) $s'St$ is eligible strictly maximal w.r.t. D and σ; (iv) $s''\sigma S't'\sigma \not\succ s'\sigma St\sigma$; (v) $t\sigma \not\succeq s'\sigma$ and (vi) $h\sigma \not\succeq s\sigma$.

where (i) $\sigma = \mathrm{mgu}(\{s{=}s', s{=}s''\})$; (ii) $t'S's''$ is eligible strictly maximal w.r.t. $C \vee hHs$ and σ; (iii) tTs' is eligible strictly maximal w.r.t. D and σ; (iv) $s''\sigma S't'\sigma \not\succ t\sigma Ts'\sigma$; (v) $t\sigma \not\succeq s'\sigma$ and (vi) $h\sigma \not\succeq s\sigma$.

System 11: The ordered chaining calculus for compositional binary relations $\mathcal{OC}^{\succ}_{Sel}$

predicate symbols such that $(S \circ T \subseteq H) \in \boldsymbol{\mathcal{C}}$, where $\boldsymbol{\mathcal{C}}$ is an *associative* compositional theory. The calculus is parametrised, as usual, with an admissible ordering $\succ$ (specified below) and a selection function *Sel* for negative literals. Note that in the Ordered Chaining rule we can "chain" compositional relations even if they have equal arguments, but only if the resulting compositional relation differs from those that have been used in the premises (otherwise the conclusion is subsumed by the corresponding premise): see conditions $(iv)'$ and $(v)'$ of this rule. This, however, never happens if compositional axioms are induced by some total precedence. Note also that a conclusion of every inference rule from System 11 might not be unique, since it is neither excluded that both $(S \circ T \subseteq H_1) \in \boldsymbol{\mathcal{C}}$ and $(S \circ T \subseteq H_2) \in \boldsymbol{\mathcal{C}}$ for some $H_1 \neq H_2$, nor that both $(S \circ T_1 \subseteq H) \in \boldsymbol{\mathcal{C}}$ and $(S \circ T_2 \subseteq H) \in \boldsymbol{\mathcal{C}}$ for some $T_1 \neq T_2$ or both $(S_1 \circ T \subseteq H) \in \boldsymbol{\mathcal{C}}$ and $(S_2 \circ T \subseteq H) \in \boldsymbol{\mathcal{C}}$ for some $S_1 \neq S_2$. In other terms, all these inference rules are *multi-conclusion*.

Definition 5.6 (Admissible Order). The ordering $\succ$ on expressions is called

admissible for chaining if $\succ$ is *admissible for resolution* (see Definition 3.34) and additional:

(**T**) $\succ$ is total on ground terms,

and for every compositional axiom $S \circ T \subseteq H$ and terms $s \succ t$, $s \succ h$, the following hold:

(**C1**) $\neg(sHh) \succ (sSt)$; $\neg(hHs) \succ (tTs)$; $\neg(sHs) \succ (tTs)$;
(**C2**) $\neg(sHh) \succ \neg(tTh)$; $\neg(hHs) \succ \neg(hSt)$; $\neg(sHs) \succ \neg(sSt)$;
(**C3**) $(sSt) \succ \neg(tTh)$; $(tTs) \succ \neg(hSt)$; $(sHs) \succ (sSt)$ ⟡

Conditions (C1) and (C2) are needed to ensure monotonicity of the Negative Chaining rule and condition (C3) guarantees monotonicity of the Compositional Resolution rule:

Lemma 5.7 (Monotonicity for $\mathcal{OC}^{\succ}_{\mathbf{Sel}}$). *All rules except the Ordered Chaining rule from $\mathcal{OC}^{\succ}_{Sel}$ are monotone for ground clauses w.r.t. every admissible ordering.*

Proof. (**1**) Negative Chaining:
We show that the conclusion of this rule is always smaller than its right premise:

(**1.*a***) $\mathtt{NC} : C \vee \underline{\boldsymbol{s}}\boldsymbol{St}^{\star}, \boldsymbol{\neg(\underline{s}Hh)} \vee D \vdash C \vee \neg(tTh) \vee D$ (Negative Chaining, left):
By conditions (iv) and (v) of this rule we must have $s \succ t$ and $s \succ h$ (we use totality condition (T) for ordering $\succ$ here). Since $S \circ T \subseteq H$, by conditions (C1) and (C2) of admissible orderings (see Definition 5.8), we have $\neg(sHh) \succ (sSt)$ and $\neg(sHh) \succ \neg(tTh)$. By condition (ii) of the rule we have $(sSt) \succ C$. Hence $\neg(sHh) \succ C$. An so, $(\neg(sHh) \vee D) \succ (C \vee \neg(tTh) \vee D)$.

(**1.*b***) $\mathtt{NC} : C \vee \boldsymbol{tT}\underline{\boldsymbol{s}}^{\star}, \boldsymbol{\neg(hH\underline{s})} \vee D \vdash C \vee \neg(hSt) \vee D$ (Negative Chaining, right):
By condition (v) of this rule we must have $s \succeq h$. If $s \succ h$, then the situation is fully symmetric to the left version of this rule. If $s = h$, then we should use the last properties from conditions (C1) and (C2): $\neg(hHs) = \neg(sHs) \succ (tTs)$ and $\neg(hHs) = \neg(sHs) \succ \neg(sSt)$. Again, since $(tTs) \succ C$ by condition (ii) of the rule, we obtain $(\neg(hHs) \vee D) \succ (C \vee \neg(hSt) \vee D)$.

(**2**) Compositional Resolution:
We show that the conclusion of this rule is smaller that its right premise:

(**2.*a***) $\mathtt{CR} : C \vee \underline{s}Hh \vee \boldsymbol{\underline{s}S't'}^{\star}, D \vee \boldsymbol{\underline{s}St}^{\star} \vdash D \vee \neg(tTh) \vee sHh$ (Compositional Resolution, left):
By conditions (iv) and (ii) of this rule, we have $(sSt) \succeq (sS't) \succ (sHh)$. Additionally, by condition (C3) of admissible orderings, we have $(sSt) \succ \neg(tTh)$. Hence $(D \vee sSt) \succ (D \vee \neg(tTh) \vee sHh)$.

(**2.*b***) $\mathtt{CR} : C \vee hH\underline{s} \vee \boldsymbol{t'S'\underline{s}}^{\star}, D \vee \boldsymbol{tT\underline{s}}^{\star} \vdash D \vee \neg(hSt) \vee hHs$ (Compositional Resolution, right):
This case is completely symmetric to case $(2.a)$. ⧉

Unfortunately the Ordered Chaining rule might not be monotone if the underlying compositional axioms are not compatible with any total precedence on special predicate symbols (see Definition 5.4 and Example 5.5). Indeed, for every ordering $\succ$ that is monotone for the Ordered Chaining rule, one can define such a precedence $\gg$ by setting $S \gg T$ *iff* $cSc \succ cTc$, where c is some fixed constant. Then monotonicity of the Ordered Chaining rule would imply that $max_{\succ}(cSc, cTc) \succeq cHc$ for every compositional axiom $S \circ T \subseteq H$, hence $S \gtrdot H$ or $T \gtrdot H$ and so all compositional axiom would have been admissible with $\gg$.

5.3 The Subterm Chaining Calculus

It is possible to combine the chaining calculus and the superposition calculus to handle compositional theories with equality. One could simply use the Ordered Paramodulation rule to perform paramodulation into non-equational literals (including all special non-equational literals) as before. However it is possible to use the advantage of compositional theory and perform paramodulation inferences only *into the largest argument* of special literals. Essentially, we treat the equality predicate $\simeq$ as a part of a compositional theory. We assume that every compositional theory $\boldsymbol{\mathcal{C}}$ contains all axioms $\simeq \circ S \subseteq S$ and $S \circ \simeq \subseteq S$ for every special predicate symbol S. *However we do not allow the equational predicate to be the result of composition of non-equational predicates, i.e., $S \circ T \subseteq \simeq$ implies $S = \simeq$ and $T = \simeq$.* This is done to avoid problems with monotonicity.

Applying these modifications, we obtain a so-called *subterm chaining calculus* that is an extension of the ordered chaining calculus from System 11 with inference rules given in System 12, where now equality can be used as a compositional predicate symbol. Note that the Positive Superposition and the Negative Superposition rules are instances of the Ordered Subterm Chaining and the Negative Subterm Chaining rules respectively, when $S = \simeq$ (in the left variants). The right variants of these rules are not needed for equational literals, since we treat equality symmetrically. The Equality Factoring rule is simulated by a self-application of the Compositional Resolution rule for equational atoms.

Definition 5.8 (Admissible Order). The ordering $\succ$ on expressions is called *admissible (for subterm chaining)* if $\succ$ is *admissible for chaining* (see Definition 5.8) and superposition (see Definition 4.15). ⋘

Lemma 5.9 (Monotonicity for $\mathcal{SC}^{\succ}_{\mathbf{Sel}}$). *All rules except the Ordered Chaining rule from $\mathcal{SC}^{\succ}_{Sel}$ are monotone for ground clauses w.r.t. every admissible ordering.*

Proof. Monotonicity of the Negative Chaining, Compositional Resolution, Reflexivity Resolution and Equality Factoring rules has been already shown (see Lemma 5.7 and Lemma 4.16). The remaining rules Ordered Subterm Chaining and Negative

Ordered Paramodulation

$$\texttt{OP}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{L}[\underline{\boldsymbol{s'}}]}{C\sigma \vee D\sigma \vee L[t]\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $L[s']$ is eligible (strictly maximal if positive) w.r.t. D and σ; (iv) $L[s']$ is a non-special literal; (v) $t\sigma \not\succeq s\sigma$ and (vi) s' is not a variable.

Reflexivity Resolution

$$\texttt{RR}: \frac{C \vee \underline{\boldsymbol{s}} \not\simeq \underline{\boldsymbol{s'}}}{C\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$ and (ii) $s \not\simeq s'$ is eligible w.r.t. C and σ.

Ordered Subterm Chaining

$$\texttt{OSC}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad \boldsymbol{r}[\underline{\boldsymbol{s'}}]\boldsymbol{Sh}^{\star} \vee D}{C\sigma \vee r[t]\sigma Sh\sigma \vee D\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) rSh is eligible strictly maximal w.r.t. D and σ; (iv) $(s\sigma \simeq t\sigma) \not\succeq (r\sigma Sh\sigma)$; (v) $t\sigma \not\succeq s\sigma$; (vi) $h\sigma \not\succeq r\sigma$ and (vii) s' is not a variable.

$$\frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad \boldsymbol{hSr}[\underline{\boldsymbol{s'}}]^{\star} \vee D}{C\sigma \vee h\sigma Sr[t]\sigma \vee D\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) hSr is eligible strictly maximal w.r.t. D and σ; (iv) $(s\sigma \simeq t\sigma) \not\succeq (h\sigma Sr\sigma)$; (v) $t\sigma \not\succeq s\sigma$; (vi) $h\sigma \not\succ r\sigma$; (vi)$'$ $S \neq \simeq$, and (vii) s' is not a variable.

Negative Subterm Chaining

$$\texttt{NSC}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad \neg(\boldsymbol{r}[\underline{\boldsymbol{s'}}]\boldsymbol{Sh}) \vee D}{C\sigma \vee \neg(r[t]\sigma Sh\sigma) \vee D\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $\neg(rSh)$ is eligible w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$; (v) $h\sigma \not\succeq r\sigma$ and (vi) s' is not a variable.

$$\frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad \neg(\boldsymbol{hSr}[\underline{\boldsymbol{s'}}]) \vee D}{C\sigma \vee \neg(h\sigma Sr[t]\sigma) \vee D\sigma}$$

where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. C and σ; (iii) $\neg(hSr)$ is eligible w.r.t. D and σ; (iv) $t\sigma \not\succeq s\sigma$; (v) $h\sigma \not\succ r\sigma$; (v)$'$ $S \neq \simeq$, and (vi) s' is not a variable.

System 12: The subterm chaining calculus $\mathcal{SC}^{\succ}_{Sel}$

Subterm Chaining are monotone since they are restricted versions of the Ordered Paramodulation rule. ◻

Admissible orderings for the chaining calculi can be obtained by taking any total reduction ordering $\succ$ on ground terms and non-special atoms (say, *KBO* or *LPO*) and associating with every ground literal $L = (E_1\ S\ E_2)$ (or $L = \neg(E_1\ S\ E_2)$)[14] a complexity measure:

$$c(L) \ := \ (\ max(L)\,,\ 1{-}pol(L)\,,\ min(L)\,,\ S(L)\,,o(L)\ ) \tag{27}$$

where $max(L)$ and $min(L)$ are respectively the maximal and the minimal expressions from $\{E_1, E_2\}$; $pol(L)$ is the *polarity* of the literal, (recall, that $pol(L) = 1$

[14]We use equational representation for non-special literals as described in subsection 4.1

if L is positive and $pol(L) = 0$ if L is negative, see p. 7); $S(L) = S$ is the special predicate symbol of L (if L is non-special literal then we have $S(L) := \simeq$) and $o(L)$ is the *orientation* of L: $o(L) = 1$ if S is non-symmetric and $E_1 \succ E_2$, and $o(L) = 0$ otherwise. Now, the literals are compared according to this vector representation, i.e., by the lexicographic combination $(\succ, >, \succ, \gg, >)$ of orderings (recall the definition and properties from p. 20), where $\succ$ is an ordering on ground terms and non-special expressions, $>$ is the standard ordering on natural numbers and $\gg$ is a precedence on special predicate symbols. It is relatively straightforward to verify all conditions of admissible orderings for the lifting of the ordering defined by this complexity measure.

Note 5.10. *In [Bachmair & Ganzinger, 1998b] the class of admissible orders $\succ$ has been restricted to those induced by a complexity measure similar to* (27) *(in this paper these orderings are called strongly admissible). In this report we have analysed the completeness proof and formulated more general conditions (C1) – (C3) for admissible orderings. This will allow us to tailor specific orderings for deciding certain fragments of first-order logic over compositional theories, which would not be possible within the class of strongly admissible orderings.* ⋘

5.4 Refutational Completeness

In this section we give a formal proof of refutational completeness for the subterm chaining calculus $\mathcal{SC}^{\succ}_{Sel}$ which is the most general chaining calculus from those we have described. The completeness proof goes as usual, by model construction which associates with each clause set N a *candidate model* $I_N\Downarrow$ induced by a set of atoms I_N, which we view as a labelled rewrite system.

Let I be a set of ground atoms. A *(labelled) rewrite relation* induced by a set of atoms I is defined by $E_1 \overset{S}{\Rightarrow}_I E_2$ ($E_1 \overset{S}{\Leftarrow}_I E_2$, $E_1 \overset{S}{\Leftrightarrow}_I E_2$) if (***i***) $E_1 \overset{S}{\Rightarrow} E_2 \in I$ (resp. $E_1 \overset{S}{\Leftarrow} E_2 \in I$, $E_1 \overset{S}{\Leftrightarrow} E_2 \in I$) or (***ii***) $S = \simeq$, $E_1 = E_1[E'_1]$ and $E_2 = E_2[E'_2]$ for some $(E'_1 \Rightarrow E'_2) \in I$. In other words, a rewrite relation induced by I is a minimal set of labelled rewrite rules that contains I and monotone w.r.t. equality.

Definition 5.11 (Rewrite Proofs). An atom $A := (E\ S\ E')$ has a *rewrite proof* in I (in symbols: $E \Downarrow^S_I E'$), if there exists a sequence of expressions $E = E_0, E_1, .., E_k, E_{k+1}, .., E_n = E'$ for $0 \leq k \leq n$, and a sequence of special predicate symbols $S_1, S_2, .., S_n$ such that: (***i***) $S_1 \circ S_2 \circ \cdots \circ S_n \subseteq S$; (***ii***) $E_{i-1} \overset{S_i}{\Rightarrow}_I E_i$ for $1 \leq i \leq k-1$; (***iii***) $E_{i-1} \overset{S_i}{\Leftarrow}_I E_i$ for $k+1 \leq i \leq n$ and (***iv***)[15] either $E_{k-1} \overset{S_k}{\Rightarrow}_I E_k$ or $E_{k-1} \overset{S_k}{\Leftrightarrow}_I E_k$ (see the left part of Figure 18). ⋘

[15] This condition is not needed if $k = 0$

Figure 18 Rewrite and non-rewrite proofs for ordered chaining

valleys:	*elementary non-valleys:*	
$E_0 \overset{S_1}{\Rightarrow} E_1 \cdots \overset{S_{k-1}}{\Rightarrow} E_{k-1} \overset{S_k}{\Rightarrow} E_k \overset{S_{k+1}}{\Leftarrow} \cdots \overset{S_n}{\Leftarrow} E_n$ (S)	$E_0 \overset{S_1}{\Leftarrow} E_1 \overset{S_2}{\Rightarrow} E_2$	*(peak)*
or	$E_0 \overset{S_1}{\Leftrightarrow} E_1 \overset{S_2}{\Rightarrow} E_2,\ E_0 \overset{S_1}{\Leftarrow} E_1 \overset{S_2}{\Leftrightarrow} E_2$	*(plateau)*
$E_0 \overset{S_1}{\Rightarrow} E_1 \cdots \overset{S_{k-1}}{\Rightarrow} E_{k-1} \overset{S_k}{\Leftrightarrow} E_k \overset{S_{k+1}}{\Leftarrow} \cdots \overset{S_n}{\Leftarrow} E_n$	$E_0 \overset{S_1}{\Leftrightarrow} E_1 \overset{S_2}{\Leftrightarrow} E_2$	*(plane)*

In other words, a rewrite proof is a chain consisting of *let-to-right rewrite steps* followed by at most one *two-way rewrite step* and a sequence of *right-to-left rewrite steps*. These rewrite proofs are called *valleys* in [Bachmair & Ganzinger, 1998*b*]. The elementary non-valley chains of length 2 can be classified as follows: (***i***) $E_0 \overset{S_1}{\Leftarrow} E_1 \overset{S_2}{\Rightarrow} E_2$ is a *peak*; (***ii***) $E_0 \overset{S_1}{\Leftrightarrow} E_1 \overset{S_2}{\Rightarrow} E_2$ or $E_0 \overset{S_1}{\Leftarrow} E_1 \overset{S_2}{\Leftrightarrow} E_2$ is a *plateau* and (***iii***) $E_0 \overset{S_1}{\Leftrightarrow} E_1 \overset{S_2}{\Leftrightarrow} E_2$ is a *plain*. In all these chains it must be that $E_1 \succeq E_0$ and $E_1 \succeq E_2$. A valley can be seen as a chain that do not contain any peaks, plateaus or plains.

A *(rewrite) model* $I\Downarrow$ induced by a set of atoms (= labelled rewrite rules) I is defined by $I\Downarrow \vDash A = (E_1\ S\ E_2)$ *iff* A has a rewrite proof in I. Given an expression E, let I^E denotes the set of atoms from I whose non-special expressions are not greater than E: $I^E := \{(E_0\ S_1\ E_2) \in I \mid E_0 \preceq E, E_1 \preceq E\}$. Then the following property can be shown:

Lemma 5.12 (Properties of Rewrite Proofs). *Let I be a set of atoms and A be an atom whose maximal non-special expression is E. Then $I\Downarrow \vDash A$ iff $I^E\Downarrow \vDash A$ iff $I_{(\neg A)}\Downarrow \vDash A$.*

Proof. Every rewrite proof in I^E or $I_{(\neg A)}$ is also a rewrite proof in I. Hence from $I^E\Downarrow \vDash A$ or $I_{(\neg A)}\Downarrow \vDash A$ it follows that $I\Downarrow \vDash A$. To prove *"only if"* parts of this lemma, suppose the atom $A = (E\ S\ E')$ has a rewrite proof $P := (E = E_0\ S_1\ E_1 \cdots S_n\ E_n = E')$. W.l.o.g., assume that $E \succeq E'$ (the remaining case is considered symmetrically). It follows from the definition of a rewrite proof that every expression E_i with $0 \leq i \leq n$ involved in P, is not greater than E. Hence $I^E\Downarrow \vDash A$.

It remains to show that $I_{(\neg A)} \vDash A$ as well. For this we need to use the conditions of admissible ordering from Definition 5.6. We demonstrate that every atom $E_{i-1}\ S_i\ E_i$ from P, is smaller than $\neg A$. If both E_{i-1} and E_i are smaller then $E = E_0$, then $\neg A = \neg(E_0\ S\ E_n) \succ (E_0\ S_i'\ E_i)$, where $S_1 \circ \ldots \circ S_i \subseteq S_i'$ (by condition (C1) of admissible orderings). And finally, $(E_0\ S_i'\ E_i) \succ \neg(E_{i-1}\ S_i\ E_i) \succ (E_{i-1}\ S_i\ E_i)$ (by conditions (C3) and (R1) of admissible orderings). If either E_{i-1} or E_i is equal to $E = E_0$, then $E_{i-1}\ S_i\ E_i$ is either the first or the last step in P. Hence, the following cases can be distinguished: (***a***) $i = 1$ and $E = E_0 = E_{i-1} \succ E_i \neq E_n$, or (***b***) $i = n$ and $E = E_0 \neq E_{i-1} \prec E_i = E_n$, or (***c***) $i = n = 1$, i.e., $A = (E_{i-1}\ S_{i-1}\ E_i)$. In case (a), we have $\neg A \succ (E_{i-1}\ S_{i-1}\ E_i)$ by condition (C1) of admissible ordering.

Case (b) is symmetric to case (a). In case (c), $\neg A \succ (E_{i-1}\ S_{i-1}\ E_i)$ by condition (R1) of admissible ordering. ⧉

It is not always true that a rewrite model induced by a set of atoms admits compositional and/or equational axioms. This is however true if the set of atoms I is complete in the following sense:

Definition 5.13 (Canonical Rewrite System). We say that a plateau, peak or a plain $E_0\ S_1\ E_1\ S_2\ E_2$ *commutes* in I if $S_1 \circ S_2 \subseteq S$ implies that $E_0\ S\ E_2$ has a rewrite proof in I. A rewrite system I is *canonical (for chaining)* if every plateau, peak or a plain in I commutes. ◈

Below we establish an analog of Critical Pair Lemma for labelled rewrite systems:

Lemma 5.14 (Canonical Proofs). *Let I be a canonical rewrite system for ordered chaining and A be an atom. Then $I \models_{\overline{T}} A$ iff $I\Downarrow \models A$.*

Proof. The *"if"* part of this lemma is trivial, since every rewrite proof in I is a chain in I. In order to show the remaining part of the lemma, let $P := (E_0\ S_1\ E_1\ S_2\ E_2 \cdots S_n\ E_n)$ be a chain for $A = (E_0\ S\ E_n)$ based on I with $S_1 \circ \cdots \circ S_n \subseteq S$. We show by induction on multiset $\{E_0, E_1,.., E_n\}_m$ (w.r.t. the multiset extension of the ordering $\succ$ on expressions) that $I\Downarrow \models A$.

Suppose the chain P is not a rewrite proof. Then there should be a peak, a plateau or a plain $E_{i-1}\ S_i\ E_i\ S_{i+1}\ E_{i+1}$ in P, i.e., with $E_i \succeq E_{i-1}$ and $E_i \succeq E_{i+1}$. Since I is canonical, then $I\Downarrow \models E_{i-1}\ S'_{i+1}\ E_{i+1}$, where $S_i \circ S_{i+1} \subseteq S'_{i+1}$, i.e., there is a valley $E_{i-1}\ S_i^1\ E_i^1\ S_i^2\ E_i^2 \cdots S_i^k\ E_{i+1}$ with $S_i^1 \circ S_i^2 \circ \cdots S_i^k \subseteq S'_{i+1}$ based on I. Note, that in particular $E_i \succ E_i^j$ for all j with $1 \leq j < k$, and so, $\{E_{i-1}, E_i, E_{i+1}\}_m \succ \{E_{i-1}, E_i^1,.., E_i^{k-1}, E_{i+1}\}_m$. Hence, we found a new chain for A with a smaller complexity. By induction hypothesis, we obtain that $I\Downarrow \models A$. ⧉

Definition 5.15 (Candidate Models, Productive Clauses). Let $N \subseteq \mathrm{Cl}^0_\Sigma$ and $C \in \mathrm{Cl}^0_\Sigma$. The *candidate models* $I_N\Downarrow$ for N, $I_C\Downarrow$ for N_C and $I^C\Downarrow$ for N^C are induced respectively by the sets I_N, I_C and I^C that are defined as follows:
$I_N := \bigcup_{C\in N} I^C$; $I_C = \bigcup_{C'\in N_C} I^{C'}$, and $I^C := I_C \cup \Delta C$, where $\Delta C := \{A\}$ if:
($\boldsymbol{i}$) $I_C\Downarrow \not\models C$;
($\boldsymbol{ii}$) $C = (C' \vee A)$ where A is an eligible maximal atom w.r.t. C', and
($\boldsymbol{iii}$) for every $C_1 = (C'_1 \vee A_1) \in N^C$ with $\Delta C_1 = \{A_1\}$, we have $(I_C \cup \{A\})\Downarrow \not\models C'_1$.
If there is no atom A satisfying conditions ($\boldsymbol{i}$) – ($\boldsymbol{iii}$), we assign $\Delta C = \{\}$. If $\Delta C \neq \{\}$, we call clause C *productive w.r.t. N* and say that C *produces* ΔC. ◈

Lemma 5.16 (Properties of Candidate Models). *Let the candidate models and productive clauses for a set N of ground clauses be defined like in Definition 5.15. Then for every $C \in N$, (**i**) $I^C\Downarrow \models C$ implies that $I_N\Downarrow \models C$, and (**ii**) if $C = C' \vee \boldsymbol{A}^\star$ is a productive clause, then $I_N\Downarrow \not\models C'$.*

Proof. (***i***) In order to show the first property, assume that $I^C\Downarrow \models C$, but $I_N\Downarrow \not\models C$. This is only possible if for some atom $B = (E_1\ S\ E_2)$ occurring negatively in C, we have $I^C\Downarrow \not\models B$ but $I_N\Downarrow \models B$. Then by Lemma 5.12, $I_{(\neg B)}\Downarrow \models B$, which is not possible since $\neg B \preceq C$.

Part (***ii***) of this lemma is guaranteed by the case (*iii*) of the definition for productive clauses (see Definition 5.15). ⧉

Lemma 5.17 (Counterexample-Reduction Lemma). *Let N be a set of ground clauses which is saturated w.r.t. $\mathcal{SC}^{\succ}_{\mathit{Sel}}$. Then for every clause $D \in N$ with $I_N\Downarrow \not\models D \neq \square$, there exists a clause $D_1 \in N$ such that $D_1 \prec D$ and $I_N\Downarrow \not\models D_1$.*

Proof. If D is a counterexample for $I_N\Downarrow$, i.e., $I_N\Downarrow \not\models D$, then $I_D\Downarrow = I^D\Downarrow \not\models D$ (here we use Lemma 5.16 (*i*)). By Lemma 5.16 (*ii*), D cannot be a productive clause, so Definition 5.15 leaves us only the following possibilities:

(**1**) $D = (D' \vee \neg\boldsymbol{A})$, where $\neg A$ is an eligible literal in D, and $I_D\Downarrow \models A = (E_1\ H\ E_2)$ for some special predicate symbol H. For definiteness, assume that $E_1 \succ E_2$ (the remaining case $E_2 \succeq E_1$ is considered symmetrically). Then either (**1.1**) $H = \simeq$ and $E_1 = E_2 = s$ for some ground term s, or (**1.2**) $A = (P \simeq \mathtt{T})$ is a non-special atom and P is reducible by some rewrite rule $(E_1' \Rightarrow E_2') \in I_C$,[16] or (**1.3**) $E_1 = s$, $E_2 = h$ and the first step in the rewrite proof of $A = (sHh)$ is $s \overset{S}{\Rightarrow} t$ for some $S \neq \simeq$ and a ground term $t \prec s$, or (**1.4**) $E_1 = r[s]$, $E_2 = h$ and the first step in the rewrite proof of $A = (r[s]Hh)$ is done by $(s \Rightarrow t) \in I_D$. The cases (1.1) and (1.2) are identical to those considered for the paramodulation and superposition calculi. The principally new cases for the chaining calculus are (1.3) and (1.4).

N	*Clauses* C	ΔC
D	$D' \vee \neg(\boldsymbol{sHh})$	$-$
$\curlyvee$		
D_1	$C' \vee D' \vee \neg(tTh)$	?
C	$C' \vee \boldsymbol{sSt}^\star$	$s \overset{S}{\Rightarrow} t$

In case (1.3) the rewrite rule $s \overset{S}{\Rightarrow} t$ must be produced by some productive clause $C = (C' \vee \boldsymbol{sSt}^\star) \prec D$. Moreover, there exists a special predicate symbol T such that $S \circ T \subseteq H$ and $I_N\Downarrow \models (tTh)$. Then a Negative Chaining inference with the conclusion $\mathtt{NC}[C, D]$: $D_1 = (C' \vee D' \vee \neg(tTh)) \in N$ is possible which is a smaller clause than D. D_1 is a counterexample for $I_N\Downarrow$, since $I_N\Downarrow \not\models D'$, $I_N\Downarrow \models (tTh)$ and $I_N\Downarrow \not\models C'$ by Lemma 5.16 (*ii*).

[16] Note that no *labeled* rewrite rule can be used in a rewrite proof for A, since P is not a term

N	Clauses C	ΔC
D	$D' \vee \neg(\boldsymbol{r[s]Hh})$	$-$
$\curlyvee$		
D_1	$C' \vee D' \vee \neg(r[t]Hh)$	?
C	$C' \vee \boldsymbol{s \simeq t^\star}$	$s \Rightarrow t$

In case (1.4) the rewrite rule $s \Rightarrow t$ is produced by some productive clause $C = (C' \vee \boldsymbol{s \simeq t^\star})$. Then the Negative Subterm Chaining rule can be applied to C and D which produces a clause $\mathtt{NSC}[C, D]$: $D_1 = (C' \vee D' \vee \neg(r[t]Hh)) \in N$ which is a smaller counterexample for $I_N\Downarrow$ than D.

N	Clauses C	ΔC
D	$D' \vee \overbrace{\boldsymbol{(E_1\ S\ E_2)}}^{A}$	$-$
$\curlyvee$		
C	$C'' \vee \overbrace{(E_1'\ H\ E_2')}^{B} \vee \boldsymbol{A_1}^\star$	A_1

(**2**) $D = (D' \vee \boldsymbol{A})$, where $A = (E_1\ S\ E_2)$ is an eligible maximal literal in D, but D is not productive because condition (iii) from Definition 5.15 is violated. In this case, there exists a productive clause $C = (C' \vee \boldsymbol{A_1}^\star) \preceq D$, such that $(I_D \cup A)\Downarrow \vDash C'$, i.e., $C' = (C'' \vee B)$ where $(I_D \cup A)\Downarrow \vDash B = (E_1'\ H\ E_2')$. Assume that $E_1 \succeq E_2$ (the case $E_2 \succ E_1$ is considered symmetrically). Then $E_1 \succeq E_1'$ and $E_1 \succeq E_2'$, since otherwise for $E' := \max\{E_1', E_2'\}$, we have $I_N\Downarrow \vDash I^{E'}\Downarrow = (I_D \cup A)^{E'}\Downarrow \vDash B$ (by Lemma 5.12) which is not possible by Lemma 5.16 (ii). Since atom A is used in the rewrite proof of B, then $E_1 = E_1'$ and either (**2.1**) it is a one-step rewrite proof, i.e., $A = B$, or (**2.2**) $A \neq B$ and $E_1 \succ E_2$: see the definition of rewrite proofs (Definition 5.11). In case (2.1) $A \succeq A_1 \succ B = A$ hence $C = D$, $A_1 = B$ and the counterexample can be reduced using the Ordered Factoring rule.

N	Clauses C	ΔC
D	$D' \vee \boldsymbol{sSt^\star}$	$-$
$\curlyvee$		
D_1	$D' \vee \neg(tTh) \vee sHh$	?
C	$C' \vee sHh \vee \boldsymbol{sS't'^\star}$	A

In case (2.2), A and B must be special atoms: $A = (sSt)$ and $B = (sHh)$ for some ground terms $s \succ t$ and $s \succeq h$. The case $s = h$ is not possible because otherwise by the last property from condition (C3) of admissible orderings (see Definition 5.8), we have $B = (sHs) \succ (sSt) = A$. Since $(I_N \cup \{s \overset{S}{\Rightarrow} t\})\Downarrow \vDash sHh$ then $I_N\Downarrow \vDash tTh$ for some special predicate symbol T such that $S \circ T \subseteq H$. Then the conclusion $\mathtt{CR}[C, D]$: $D_1 = (D' \vee \neg(tTh) \vee sHh)$ a Compositional Resolution inference, which is a smaller clause than D_1, must belong to N. Since $I_N\Downarrow \nvDash D'$, $I_N\Downarrow \vDash (tTh)$ and $I_N\Downarrow \nvDash sHh$ (the last is by Lemma 5.16 (ii) applied to C), we have obtained a smaller counterexample than D. ⧉

Remark 5.18. It is possible to dramatically restrict the Compositional Resolution rule when the maximal literal $s''S't'$ in the left premise is equality. We show that in the correspondent case of Counterexample-Reduction Lemma (Lemma 5.17, case (2.2)) a counterexample can be reduced using the Ordered Subterm Chaining or the following extension of the Equality Factoring rule for other special predicate symbols: see Figure 19.

Figure 19 An extension of the Equality Factoring rule for compositional theories

Equality Factoring

$$\texttt{EF}: \frac{C \vee \underline{s}' H h \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^\star}{C\sigma \vee \neg(t\sigma H h\sigma) \vee s'\sigma H h\sigma} \qquad \frac{C \vee h H \underline{s}' \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^\star}{C\sigma \vee \neg(h\sigma H t\sigma) \vee h\sigma H s'\sigma}$$

[*where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. $C \vee s'Hh$ and σ, and (iii) $t\sigma \not\succeq s\sigma$.*] [*where (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) $s \simeq t$ is eligible strictly maximal w.r.t. $C \vee hHs'$ and σ, and (iii) $t\sigma \not\succeq s\sigma$.*]

Indeed, if in case (2.2) of Lemma 5.17 $S' = \simeq$, then there are only two following cases possible:

N	Clauses C	ΔC
D	$D' \vee \boldsymbol{sSt}^\star$	–
$\curlyvee$		
D_2	$C' \vee sHh \vee t'St \vee D'$	?
C	$C' \vee sHh \vee \boldsymbol{s} \simeq \boldsymbol{t'}^\star$	$s \Rightarrow t'$
$\curlyvee$		
D_1	$C' \vee \neg(t'Hh) \vee sHh$	?

Case (***a***) $I_N{\Downarrow} \vDash (t'St)$. Since $I_N{\Downarrow} \vDash tTh$, then $I_N{\Downarrow} \vDash (t'Hh)$ and the conclusion $\texttt{EF}[C]$: $D_1 = (C' \vee \neg(t'Hh) \vee sHh) \in N$ of an Equality Factoring inference from C is a smaller counterexample than D.

Case (***b***) $I_N{\Downarrow} \nvDash (t'St)$. Then the conclusion $\texttt{OSC}[C, D]$: $D_2 = (C' \vee sHh \vee t'St \vee D') \in N$ of an Ordered Subterm Chaining inference from C and D is a smaller counterexample than D. ⟡

Note 5.19. *In the original completeness proofs for ordered chaining calculi, Bachmair & Ganzinger [1995, 1998*b*] used a slightly different model construction. Given a saturated set of clauses N they have showed by induction over a well ordering $\succ$ that $I^C{\Downarrow} \vDash C$. So the counterexamples in their sense are those clauses C that are* `false` *in $I^C{\Downarrow}$ (not in $I_N{\Downarrow}$ like here). The advantage of the model construction that we have presented here is twofold. First, the structure and philosophy of our proof goes along with the completeness proofs for the ordered resolution and the superposition calculi. Second, we clearly separate properties of candidate models for arbitrary clause sets (in Lemma 5.16) from those for saturated sets of clauses (in Lemma 5.17).*

*Going slightly deeper into details, condition (iii) of productive clauses (see Definition 5.15) is missing in [Bachmair & Ganzinger, 1995, 1998*b*] which clearly gives problems with the property (ii) from Lemma 5.16. Without this condition, the properties of productive clauses and counterexample reduction have to be demonstrated together in one inductive proof, which can be carried out only for saturated clause sets.* ⟡

5.5 Hyper-Inferences

It is possible to define hyper-inferences for the chaining calculi in similar fashion as it has been done for the resolution calculus (see Figure 8 and Figure 9 on p.58). A

counterexample having several selected negative literals can be reduced by applying appropriate inferences to *all* of them *simultaneously*. However it is difficult to describe such a hyper-combination of counterexample-reducing inferences, since in the chaining calculus there are many of those: Ordered Resolution, Ordered Paramodulation, Reflexivity Resolution and Negative Subterm Chaining. Every combination of these inferences might be applied to different negative literals simultaneously.

In decision procedures it is often important to control the application of the Negative Chaining rule into variables (i.e., when the unified term in the negative literal is a variable). To reduce possible negative effect of the Negative Chaining rule (such as increase in the number of variables in the conclusion on the inference), we introduce a particular hyper-inference strategy which can be implemented using an *a-posteriori selection function* (see Figure 9). Given a clause with several selected literals, we allow either (***i***) a simultaneous application of the negative chaining rule on *all* of these literals, or (***ii***) any other inference with *any* of selected literals. In Figure 20 (the upper part) we have formulated a special Negative Hyper-Chaining inference rule that formalises the first part of this strategy. This rule is applied to a

Figure 20 A hyper- extension of the Negative Chaining rule

Negative Hyper-Chaining

$$\mathtt{HC}: \frac{\left\{C_i \vee \underline{\boldsymbol{s_i}}\boldsymbol{S_i}\boldsymbol{t_i}^{\star},\ C_j \vee \boldsymbol{t_j}\boldsymbol{T_j}\underline{\boldsymbol{s_j}}^{\star} \mid i \in I, j \in J\right\} \quad \bigvee_{i\in I} \neg(\underline{\boldsymbol{s'_i}}\boldsymbol{H_i}\boldsymbol{h_i})^{\sharp} \vee \bigvee_{j\in J} \neg(\boldsymbol{h_j}\boldsymbol{H_j}\underline{\boldsymbol{s'_j}})^{\sharp} \vee D}{\bigvee_{i\in I}[C_i\sigma \vee \neg(t_i\sigma T_i h_i\sigma)] \vee D\sigma}$$

where (i) $S_k \circ T_k \subseteq H_k$ for all $k \in I \cup J$; (ii) $\sigma = mgu(\{s_k{=}s'_k \mid k \in I \cup J\})$; and there is a ground substitution $\sigma^0 = \sigma\tau^0$ such that for all $i \in I$, $j \in J$: (iii) $s_i S_i t_i$ ($t_j S_j s_j$) are eligible strictly maximal w.r.t. C_i (C_j) and σ^0; (iv) $\{\neg(s'_i H_1 h_i),\ \neg(h_j \tilde{H}_n s'_j) \mid i \in I, j \in J\}_m$ is eligible w.r.t. D and σ^0; (v) $t_i\sigma^0 \not\succeq s_i\sigma^0$, $t_j\sigma^0 \not\succeq s_j\sigma^0$ and (vi) $h_i\sigma^0 \not\succeq s'_i\sigma^0$, $h_j\sigma^0 \not\succ s'_j\sigma^0$.

Negative Hyper-Chaining

$$\mathtt{HC}: \frac{C_i \vee \underline{\boldsymbol{s_1}}\tilde{\boldsymbol{S}}_1\boldsymbol{t_1}^{\star} \ \ldots\ C_j \vee \underline{\boldsymbol{s_n}}\tilde{\boldsymbol{S}}_n\boldsymbol{t_n}^{\star} \quad \neg(\underline{\boldsymbol{s'_1}}\tilde{\boldsymbol{H}}_1\boldsymbol{h_1})^{\sharp} \vee \cdots \vee \neg(\underline{\boldsymbol{s'_n}}\tilde{\boldsymbol{H}}_n\boldsymbol{h_n})^{\sharp} \vee D}{C_i\sigma \vee \neg(t_1\sigma\tilde{T}_1 h_1\sigma) \vee D\sigma}$$

where (i) $\tilde{S}_i \circ \tilde{T}_i \subseteq \tilde{H}_i$ for all $1 \le i \le n$; (ii) $\sigma = mgu(\{s_i{=}s'_i \mid 1 \le i \le n\})$; and there is a ground substitution $\sigma^0 = \sigma\tau^0$ such for all i with $1 \le i \le n$: (iii) $s_i S_i t_i$ is eligible strictly maximal w.r.t. C_i and σ^0; (iv) $\{\neg(s'_1 H_1 h_1),..,\neg(s'_n \tilde{H}_n h_n)\}_m$ is eligible w.r.t. D and σ^0; (v) $t_i\sigma^0 \not\succeq s_i\sigma^0$ and (vi) $h_i\sigma^0 \not\succeq s'_i\sigma^0$.

clause whose all selected literals are special and for each of them a Negative Chaining inference with the same clause is possible. Then all these inferences are applied at once. Note that the ordering restriction for all negative chaining inferences must be *simultaneously* satisfied for some *ground* instance of the unifier σ. In this form

the rule is more restrictive than if the ordering restrictions are verified separately for σ, as is usually done (recall the discussion from Example 3.35). This helps to filter-out many unnecessary chaining inferences:

Example 5.20. Consider a clause $D := (\neg\mathtt{xTy} \vee \neg\mathtt{yTx})$, where $\mathtt{T}$ is a transitive predicate symbol. Both negative literals in this clause are maximal, since they differ only by permutation of variables. Hence it is possible to apply the Ordered Chaining rule to both of them, e.g., with a clause $C := \mathtt{f(x)Tx}$. However the Negative Hyper-Chaining rule with (two copies of) C cannot be applied to *both* literals of D *simultaneously*, since condition (vi) of this rule is violated. Indeed, there is no ground substitution σ^0 such that $\mathtt{y}\sigma^0 \not\succeq \mathtt{x}\sigma^0$ and $\mathtt{x}\sigma^0 \not\succeq \mathtt{y}\sigma^0$. Note, that this condition can be possibly satisfied for a non-ground substitution σ. ⋘

To simplify the exposition and usage of the Negative Hyper-Chaining rule (in particular the annoying distinction between "left" and "right" arguments of special predicate symbols), we introduce additional notation. For every binary predicate symbol S, we introduce a shortcut $S^\smile$ that denotes the *inverse* of S, i.e., $(xS^\smile t) := (tSx)$. According to this notation, every compositional axiom $S \circ T \subseteq H$ induces a compositional axiom $T^\smile \circ S^\smile \subseteq H^\smile$ on inverse relations. Finally, let $\tilde{S}$ stand for S or $S^\smile$. Using this notation, the Negative Hyper-Chaining can be formulated more compactly (although with condition (vi) relaxed) in the lower part of Figure 20.

5.6 Redundancy

The standard redundancy criterion based on semantical entailment cannot be directly adapted for the chaining calculus. The problem is caused by the Ordered Chaining rule which might be not monotone. Note that monotonicity of this rule has not been used in our model construction, in particular in the proof of Counterexample-Reduction Lemma (Lemma 5.17), since this rule may not reduce counterexamples. The sole purpose of this rule is to guarantee that the candidate model $I_N\Downarrow$ constructed for a saturated clause set N is a $\boldsymbol{\mathcal{C}}$-model.

To show why the standard redundancy does not work, suppose, that we have a compositional axiom $(S \circ T \subseteq H) \in \boldsymbol{\mathcal{C}}$ with $H \neq S$, $H \neq T$ and $tHt \succeq tSt$, $tHt \succeq tTt$ for some ground term t. That is, we have a situation when the Ordered Chaining inference from clauses $tS\underline{t}$ and $\underline{t}Ht$ is possible but not monotone. Using the standard redundancy criterion based on semantical entailment (w.r.t. a theory $\boldsymbol{\mathcal{C}}$), it is easy to see that clause $C := tHt$ is redundant w.r.t. an *unsatisfiable* clause set $N := \{tSt, tTt, tHt, \neg(tHt)\}$, since $N_C := \{tSt, tTt\} \models\!\!\!\!\!\!\sim_{\boldsymbol{\mathcal{C}}} tHt$. However, the empty clause could not be derived from N if we delete clause C *permanently*. If we analyse the model construction given in Counterexample Reduction Lemma (Lemma 5.17), we notice that although $N_C \models\!\!\!\!\!\!\sim_{\boldsymbol{\mathcal{C}}} C$, this does not suffices to establish that $I_C\Downarrow \models C$, since $I_C\Downarrow$ is *not necessarily a* $\boldsymbol{\mathcal{C}}$*-model*. In the particular example

above, we cannot delete C, because it is a productive clause that should reduce the minimal counterexample in N.

Let us review the arguments used to justify the standard redundancy criterion. A ground clause $C \in N$ might not be considered in the model construction, if we know *a-priory* that either $I_N\Downarrow \not\models N_C$ (which is by Lemma 5.16 (i) equivalent to $I_C\Downarrow \not\models N_C$), or $I_C\Downarrow \models C$ (hence C is not a productive clause and not a counterexample). In both cases C may neither be a minimal counterexample nor a productive clause that reduces the minimal counterexample. To put it simple, a clause C is redundant if $I_C\Downarrow \models N_C$ *implies* $I_C\Downarrow \models C$. However, contrary to paramodulation or superposition calculi, the set N_C is *not* necessary *saturated* under the chaining inference rules even if N *is* saturated. Hence, I_C might not be canonical, and so $I_C\Downarrow$ is might not be a $\boldsymbol{\mathcal{C}}$-model, which means that $N_C \not\models_{\mathbf{c}} C$ is not sufficient to establish redundancy.

It is possible to characterise redundancy by means of rewrite proofs, similarly as it is done for instance in [Bachmair & Ganzinger, 1997], by noticing that every set I_C must be closed under *monotone* Ordered Chaining inferences. However, demonstrating entailment via rewrite proofs is less convenient than by semantical arguments, therefore we formulate redundancy using a sufficient semantical approximation. Note that in order to establish redundancy for C, it suffices to show that $I^{C_1}\Downarrow \models N^{C_1}$ *implies* $I^{C_1}\Downarrow \models C$ for *some* $C_1 \prec C$. If the last property holds then either there exists a counterexample in N^{C_1}, which is smaller than C, or the clause C is not productive, hence C might not be considered. Although I_C is not closed under the Ordered Chaining rule, the set I^{C_1} might be closed. This idea is formalised using the following notion:

Definition 5.21 (Redundancy Ordering). A *redundancy ordering* for an admissible ordering $\succ$ (for chaining calculi) is a partial ordering $\succ\!\!\succ$ on ground literals such that the following conditions hold:

(**CRO1**) $\succ\!\!\succ \subseteq \succ$, $\succ\!\!\succ \circ \succ\!\!\succ \subseteq \succ\!\!\succ$, $\succ\!\!\succ \circ \succ \subseteq \succ\!\!\succ$;

(**CRO2**) $S \circ T \subseteq H$, $L \succ\!\!\succ tSs$, $L \succ\!\!\succ sTh$ and $s \succeq t$, $s \succeq h$ imply that $L \succ\!\!\succ tHh$ ⋘

Using a redundancy ordering we can express a property for inference systems which is weaker than monotonicity. An inference π (an inference system $\boldsymbol{\mathcal{S}}$) is *weakly monotone* w.r.t. $\succ\!\!\succ$, if for every ground clause C^0 the set of the clauses that are $\prec$-smaller than C^0, is closed under π (respectively $\boldsymbol{\mathcal{S}}$). Note that if π is monotone w.r.t. $\succ$ for ground clauses, then it is weakly monotone w.r.t. $\succ\!\!\succ$.

Condition (CRO1) from Definition 5.21 expresses that $\succ\!\!\succ$ is a partial subordering of $\succ$ which is closed under composition with $\succ$. This is needed to insure that every inference that is monotone w.r.t. $\succ$ remains weakly monotone w.r.t. $\succ\!\!\succ$. Note that if $\succ$ is well-founded, then $\succ\!\!\succ$ is also well-founded. Condition (CRO2) is needed to ensure that the Ordered Chaining rule is weakly monotone w.r.t. $\succ\!\!\succ$.

Lemma 5.22 (Weak Monotonicity for $\mathcal{SC}^{\succ}_{\mathbf{Sel}}$). *The subterm chaining calculus $\mathcal{SC}^{\succ}_{Sel}$ is weakly monotone w.r.t. every redundancy ordering $\succ$ for any admissible ordering $\succ$.*

For example, given an admissible ordering $\succ$ defined by (27), a redundancy ordering $\succ$ for $\succ$ can be induced by the following complexity measure on literals:

$$\dot{c}(L) \;:=\; (\, max(L)\,,\; 1{-}pol(L)\,,\; min(L)\,) \tag{28}$$

It is easy to see that this ordering fulfils both conditions from Definition 5.21 w.r.t. $\succ$ from (27).

Now, the standard redundancy criterion is defined as usual, but w.r.t. a redundancy ordering $\succ$ instead of an admissible ordering $\succ$:

Definition 5.23 (Standard Redundancy). Given a set of ground clauses N^0, and a ground clause C^0, let $N^0{}_{\prec C^0} := \{C^0{}_1 \in N^0 \mid C^0{}_1 \prec C^0\}$ be the set of ground clauses from N^0 that are $\prec$-smaller than C^0.

A *ground clause C^0 is redundant w.r.t.* a set of ground clauses N^0, if C^0 follows in $\boldsymbol{\mathcal{C}}$ from the set $N^0{}_{\prec C^0}$. A *ground inference π^0 is redundant w.r.t. N^0 in $\mathcal{SC}^{\succ}_{Sel'}$* if either $\pi^0 \notin \mathcal{SC}^{\succ}_{Sel'}$ or, otherwise, the conclusion of π^0 follows in $\boldsymbol{\mathcal{C}}$ from $N^0{}_{\prec C^0_i}$ for some premise C^0_i of π^0.

A (general) *clause C is redundant* w.r.t. a clause set N if every ground instance $C^0 \in \{C\}^{\mathrm{gr}}$ of C follows in $\boldsymbol{\mathcal{C}}$ from N^{gr}. An *inference π is redundant w.r.t. N in $\mathcal{SC}^{\succ}_{Sel}$*, if for *every* projection Sel' of Sel from the premises of π, the ground instance π^0 of π is redundant w.r.t. N^{gr} in $\mathcal{SC}^{\succ}_{Sel'}$.

We denote this redundancy criterion by $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ} = (\mathsf{R}^{\succ}_{\mathrm{Cl}}(\cdot), \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}_{\mathrm{Inf}}(\cdot))$. ⋘

It is easy to show that this notion of redundancy is a redundancy criterion indeed according to Definition 3.20: the proof of this fact is just a repetition of the proof for Lemma 3.21 (for this we only need to use that $\succ$ is well founded). Finally, we establish completeness for the subterm chaining calculus with redundancy:

Theorem 5.24 (Completeness of $\boldsymbol{\mathcal{SC}^{\succ}_{Sel}}$ with Redundancy).
$(\mathcal{SC}^{\succ}_{Sel}, \mathsf{R}^{\boldsymbol{\mathcal{S}}\succ})$ is complete for every selection function Sel, admissible ordering $\succ$ and every redundancy ordering $\succ$ for $\succ$.

Proof. Let N be saturated up to redundancy and N^{gr} be the set of ground instances of N. We need to consider a situation, when an inference π reducing a counterexample D in N^{gr} is redundant w.r.t. N^{gr}, i.e., the conclusion D_1 of π from N^{gr} follows from clauses $N_{\prec D}$ in theory $\boldsymbol{\mathcal{C}}$. Since the subterm chaining calculus is weakly monotone w.r.t. $\succ$, then $N_{\prec D}$ is saturated under $\mathcal{SC}^{\succ}_{Sel}$. By induction hypothesis, $I_{\prec D}{\Downarrow}$ is a $\boldsymbol{\mathcal{C}}$-model for $N_{\prec D}$ and consequently for D_1. Thus, $I_{\prec D}{\Downarrow} \models D_1$ which means that D_1 is *not* a counterexample. So redundant inferences might not be used for reduction of counterexamples. ⧉

6 Clause Normal Form Transformation

Saturation-based theorem provers operate not with first-order formulas but with sets of clauses. To use provers for full first-order logic, it is required to transform formulas into a *clause normal form* (**CNF**). It is well-known that the quality of the **CNF**- *transformation* has a great impact on efficiency and success of the overall theorem proving attempt. There are examples, where (naïve) textbook procedures produce exponentially large **CNF**'s in the size of the input first-order formula, which makes them not very useful in practice.

For saturation-based decision procedures the **CNF**-transformation plays even more important rôle, since the transformation has to map a decidable first-order fragment into a decidable clause class. Thus, essential properties, that contribute to decidability of a particular first-order fragment should be inherited into their clause representations.

CNF-transformation has been thoroughly studied in literature [for the overview see Nonnengart & Weidenbach, 2001; Baaz, Egly & Leitsch, 2001]. The traditional way of producing **CNF**'s for decidable first-order fragments consists of three main steps. First, a formula is translated into a negation normal form by pushing negation inwards as far as possible. Second, a so-called structural transformation is applied to a formula, that splits the formula into a conjunction of simple formulas. In the last step, skolemization is employed that introduces Skolem functions for existentially quantified variables. After this step the universal quantifiers are dropped and the result is written in a clause form.

Below we present all these transformations in their general form and later we give their variants for considered first-order fragments. In our presentation we make a special emphasis on complexity issues of transformations. We estimate a worst time complexity of transformations and the size of their results. Some material presented in this section belongs to a *logical folklore* (especially transformation procedures and complexity calculations), therefore it is hard to give any reference or historical remarks about them. For the same reason, many proofs may seem too sketchy. This section does not aim in giving comprehensive account of techniques and results about **CNF** transformation. For those, the reader is forwarded to the overview articles [Nonnengart & Weidenbach, 2001; Baaz et al., 2001] where many aspects of doing **CNF**-transformation in efficient way are presented and further links to literature are provided.

6.1 Negation Normal Form

A first-order formula (involving conjunction, disjunction and negation as the only boolean connectives) is in *negation normal form* (or shortly **NNF**) if negation symbol appears only in front of atoms in this formula. A transformation that puts

a first-order formula into **NNF** is probably the simplest normalization step. Given a first-order formula, one has to distribute negation over other boolean connectives and quantifiers using the usual *de-Morgan*'s lows:

$$\begin{array}{ll} \neg(A \wedge B) \Rightarrow \neg A \vee \neg B & \qquad \neg\neg A \;\Rightarrow\; A \qquad \neg\forall x.A \Rightarrow \exists x.\neg A \\ \neg(A \vee B) \Rightarrow \neg A \wedge \neg B & \qquad\qquad\qquad\quad \neg\exists x.A \Rightarrow \forall x.\neg A \end{array} \tag{29}$$

Figure 21 Negation normal form transformation for first-order formulas

$$\mathcal{FO} ::= A \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \neg F_1 \mid \forall y.F_1 \mid \exists y.F_1.$$

$$\begin{array}{rcl|rcl} [F]^{nnf} := [A]^{nnf} & = & A \mid & [F]^{nnf}_{\neg} := [A]^{nnf} & = & \neg A \mid \\ [F_1 \wedge F_2]^{nnf} & = & [F_1]^{nnf} \wedge [F_2]^{nnf} \mid & [F_1 \wedge F_2]^{nnf}_{\neg} & = & [F_1]^{nnf}_{\neg} \vee [F_2]^{nnf}_{\neg} \mid \\ [F_1 \vee F_2]^{nnf} & = & [F_1]^{nnf} \vee [F_2]^{nnf} \mid & [F_1 \vee F_2]^{nnf}_{\neg} & = & [F_1]^{nnf}_{\neg} \wedge [F_2]^{nnf}_{\neg} \mid \\ [\neg F_1]^{nnf} & = & [F_1]^{nnf}_{\neg} \mid & [\neg F_1]^{nnf}_{\neg} & = & [F_1]^{nnf} \mid \\ [\forall y.F_1]^{nnf} & = & \forall y.[F_1]^{nnf} \mid & [\forall y.F_1]^{nnf}_{\neg} & = & \exists y.[F_1]^{nnf}_{\neg} \mid \\ [\exists y.F_1]^{nnf} & = & \exists y.[F_1]^{nnf}. & [\exists y.F_1]^{nnf}_{\neg} & = & \forall y.[F_1]^{nnf}_{\neg}. \end{array}$$

Formally, we define a **NNF**-transformation for the set of first-order formulas assuming that they are constructed according to the grammar $\mathcal{FO}$ given in Figure 21, where A is an atom $A = a(\overline{x})$, and F_1, $F_2 \in \mathcal{FO}$. As usual, we think of formulas involving other connectives: $A \rightarrow B$ and $A \leftrightarrow B$ as abbreviations standing for $\neg A \vee B$ and $(\neg A \vee B) \wedge (\neg B \vee A)$ respectively. Negation normal form $[F]^{nnf}$ of a formula F is obtained by applying the function $[\cdot]^{nnf}$ that is defined recursively over the definition of $\mathcal{FO}$ and using additional auxiliary function $[\cdot]^{nnf}_{\neg}$ (see Figure 21). We have used an additional function $[\cdot]^{nnf}_{\neg}$ for computing a negation normal form of a negated formula. Defining **NNF**-transformation in this way makes it more obvious that recursion terminates [compare, for instance, our transformation with the one defined in Baaz et al., 2001]. Moreover the functions are defined by a proper recursion over the grammar definition of $\mathcal{FO}$. We will see later, that normal forms and clause sets for different fragments can be found much more easily given recursive definitions for the fragments.

Proposition 6.1. *For every formula $F \in \mathcal{FO}$ the result $G = [F]^{nnf}$ of* **NNF***-transformation can be computed in polynomial time in $|F|$ and produces a formula G in negation normal form such that (i) G is equivalent to F and (ii) $|G| \leq 2{\cdot}|F|$.*

Proof. Given a formula $F \in \mathcal{FO}$, the result of **NNF**-transformation $[F]^{nnf}$ is computed according to the definition in Figure 21 using at most $|F|$ recursive calls (ones for every subformula). Every recursion call takes a polynomial time

(depends on a data structure used to manipulate with formulas), so the result can be computed in polynomial time in the size of the input formula.

By induction over the construction of $F \in \mathcal{FO}$ it is easy to show that $[F]^{nnf} \equiv F$, $[F]^{nnf}_{\neg} \equiv \neg F$, $|[F]^{nnf}| \leq 2{\cdot}|F|$ and $|[F]^{nnf}_{\neg}| \leq 2{\cdot}|F|$. ⧉

The goal of **NNF**-transformation is to produce an equivalent formula having only positive occurances of non-atomic subformulas. This is needed for the subsequent structural transformation step. Note that the set of first-order formulas in negation normal form can be defined by the grammar:

$$[\mathcal{FO}]^{nnf} ::= A \mid \neg A \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \forall y.F_1 \mid \exists y.F_1. \tag{30}$$

where A is an atom and F_1, $F_2 \in [\mathcal{FO}]^{nnf}$.

Example 6.2. Let us compute negation normal form for the formula:

$$F := \ (\forall y.(\mathtt{a}(x) \wedge \mathtt{b}(y)) \rightarrow \exists z.\mathtt{c}(x,z)) \ \equiv^{def} \ (\neg\forall y.(\mathtt{a}(x) \wedge \mathtt{b}(y)) \vee \exists z.\mathtt{c}(x,z)).$$

Applying the function $[\cdot]^{nnf}$ according to the definition in Figure 21, we obtain:

$$\begin{aligned}
[F]^{nnf} = \ & [\neg\forall y.(\mathtt{a}(x) \wedge \mathtt{b}(y)) \vee \exists z.\mathtt{c}(x,z)]^{nnf} \\
= \ & ([\neg\forall y.(\mathtt{a}(x) \wedge \mathtt{b}(y))]^{nnf} \vee [\exists z.\mathtt{c}(x,z)]^{nnf}) \\
= \ & ([\forall y.(\mathtt{a}(x) \wedge \mathtt{b}(y))]^{nnf}_{\neg} \vee \exists z.[\mathtt{c}(x,z)]^{nnf}) \\
= \ & (\exists y.[\mathtt{a}(x) \wedge \mathtt{b}(y)]^{nnf}_{\neg} \vee \exists z.\mathtt{c}(x,z)) \\
= \ & (\exists y.([\mathtt{a}(x)]^{nnf}_{\neg} \vee [\mathtt{b}(y)]^{nnf}_{\neg}) \vee \exists z.\mathtt{c}(x,z)) \\
= \ & \underline{(\exists y.(\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y)) \vee \exists z.\mathtt{c}(x,z))} \in [\mathcal{FO}]^{nnf}.
\end{aligned}$$

⧈

6.2 The Structural Transformation

The structural transformation plays important rôle not only for obtaining resolution decision procedures, but in automated reasoning in general, where it is more known as *formula renaming* [see Nonnengart & Weidenbach, 2001]. The basic principle of the structural transformation can be formulated as follows. Let $F[G]$ be a first-order formula with positive occurances of a subformula G. Then $F[G]$ can be replaced with the formula $F[G/\mathtt{P}_G] \wedge \forall\overline{x}.(\mathtt{P}_G \rightarrow G)$, where $\mathtt{P}_G = \mathtt{p}_G(\overline{x})$ is a fresh predicate (not occurring in $F[G]$) with $\mathit{free}[G] \subseteq \overline{x}$, which is a satisfiability preserving transformation. In other words, if one replaces a positive subformula by a predicate containing all variables of this formula and adds a universally closed conjunct expressing that the predicate implies the replaced formula, then one obtains an *equisatisfiable* formula. The conjunct that is added is also called a *definition* for the subformula G, and the predicate $\mathtt{p}_G(\overline{x})$ is called a *definitional predicate*

(sometimes called a *label*) for G. One also says that the subformula G is being *renamed* with the predicate $\mathtt{P}_G$.

Proposition 6.3. *Let $F[G]$ be a first-order formula with positive occurances of a subformula G. Let $\mathtt{P}_G = \mathtt{p}_G(\overline{x})$ be a (fresh) definitional predicate for G such that free$[G] \subseteq \overline{x}$. Then $F[G/\mathtt{P}_G] \wedge \forall \overline{x}.(\mathtt{P}_G \rightarrow G)$ is conservative over $F[G]$.*

Proof. For proving this proposition, we need to show that (i) $F[G]$ is a logical consequence of $F[G/\mathtt{P}_G] \wedge \forall \overline{x}.(\mathtt{P}_G \rightarrow G)$ and (ii) any model of $F[G]$ can be *expanded* to a model of $F[G/\mathtt{P}_G] \wedge \forall \overline{x}.(\mathtt{P}_G \rightarrow G)$ (see Definition 2.2). The point (i) follows from Replacement Lemma 2.4. To prove the point (ii), let $\mathcal{M}$ be a model for $F[G]$. We expand the model $\mathcal{M}$ to a model $\mathcal{M}'$ by interpreting the new predicate symbol $\mathtt{p}_G$ such that $\mathtt{p}_G^{\mathcal{M}'}(\overline{x}) :\equiv G^{\mathcal{M}}$. This is always possible, since free$[G] \subseteq \overline{x}$. Obviously, this expanded interpretation $\mathcal{M}'$ is a model of $F[G/\mathtt{P}_G] \wedge \forall \overline{x}.(\mathtt{P}_G \rightarrow G)$. ⧉

The structural transformation has been initially used for propositional **CNF** transformation. Using additional (*introduced*) propositional symbols it was possible to avoid exponential blow-up for the situations, where there is no polynomial **CNF**'s over the initial signature. The simple example of such situation is the formula:

$$\mathtt{P}_n \;=\; (\mathtt{A}_1 \wedge \mathtt{A}'_1) \vee (\mathtt{A}_2 \wedge \mathtt{A}'_2) \vee \cdots \vee (\mathtt{A}_n \wedge \mathtt{A}'_n), \qquad n > 0 \tag{31}$$

which has a **CNF**, consisting of 2^n clauses. If one introduces a new propositional symbol $\mathtt{B}_i$, for every conjunct $\mathtt{A}_i \wedge \mathtt{A}'_i$, $1 \leq i \leq n$ the formula $\mathtt{P}_n$ can be represented by $2n + 1$ clauses:

$$\begin{array}{cccc} \mathtt{B}_1 \vee \mathtt{B}_2 \vee \cdots \vee \mathtt{B}_n; & & & \\ \neg\mathtt{B}_1 \vee \mathtt{A}_1; & \neg\mathtt{B}_2 \vee \mathtt{A}_2; & \cdots & \neg\mathtt{B}_n \vee \mathtt{A}_n; \\ \neg\mathtt{B}_1 \vee \mathtt{A}'_1; & \neg\mathtt{B}_2 \vee \mathtt{A}'_2; & \cdots & \neg\mathtt{B}_n \vee \mathtt{A}'_n; \end{array}$$

Note that $\mathtt{P}_n$ is not equivalent to the conjunction of these clauses: in an interpretation where all $\mathtt{B}_i$ are false the conjunction is false, but $\mathtt{P}_n$ can be either false or true independently from the values of $\mathtt{B}_i$, $1 \leq i \leq n$. However, the sets of the clauses above is conservative over $\mathtt{P}_n$: (i) the formula $\mathtt{P}_n$ is a logical consequence of these clauses and (ii) every model of $\mathtt{P}_n$ can be expanded to a model for the clauses by interpreting $\mathtt{B}_i$ as $\mathtt{A}_i \wedge \mathtt{A}'_i$, $1 \leq i \leq n$.

The structural transformation is even more useful for first-order formulas. The repeated application of renaming for quantified subformulas of a first-order formula can lower the quantifier alternation degree of the formula, which results in smaller and simpler clauses. Formula renaming techniques have other advantages for proof search [see Nonnengart & Weidenbach, 2001], however, the most important advantage of the structural transformation is that it preserves in some sense the *structure* of the input first-order formula [for a related discussion see Baaz et

Figure 22 The structural transformation for a recursively defined set of formulas

$$\mathcal{F} ::= B_1 \mid B_2 \mid \cdots \mid B_n \mid R_1[F_1,..,F_{k_1}] \mid R_2[F_1,..,F_{k_2}] \mid \cdots \mid R_m[F_1,..,F_{k_m}]$$

$$\begin{array}{lll} [F]^{str} := \mathtt{P}_F \wedge [F]^{def}; & & \\ [F]^{def} := [B_i]^{def} = \forall \overline{x}.(\mathtt{P}_F \rightarrow B_i), & \mid & 1 \leq i \leq n, \\ [R_j[F_1,..,F_{k_j}]]^{def} = \forall \overline{x}.(\mathtt{P}_F \rightarrow R_j[\mathtt{P}_{F_1},..,\mathtt{P}_{F_{k_j}}]) \wedge [F_1]^{def} \wedge \cdots \wedge [F_{k_j}]^{def} & \mid & 1 \leq j \leq m. \end{array}$$

al., 2001]. The structural transformation makes it possible to inherit the properties of formulas from decidable fragments to their clause normal forms.

The structural transformation can be defined recursively over a sets of formulas $\mathcal{F}$ represented by a general grammar in Figure 22, where B_i, $1 \leq i \leq n$ are first-order formulas for base cases of the definition, and $R_j[F_1,..,F_{k_j}]$, $1 \leq j \leq m$ are recursive constructors of new formulas from old ones. Hereby we assume that every F_i with $1 \leq i \leq k_j$ occurs positively in $R_j[F_1,..,F_{k_j}]$.[17] The definition (30) for first-order formulas in negation normal form is an example of such recursive construction. The result of structural transformation for a formula $F \in \mathcal{F}$ is defined by $[F]^{str} := \mathtt{P}_F \wedge [F]^{def}$, where $\mathtt{P}_F$ is a definitional predicate for F and $[F]^{def}$ is a conjunction of formulas expressing definitions for the introduced predicates. $[F]^{def}$ is defined recursively over $\mathcal{F}$ (see Figure 22). According to this definition, a formula F is matched to one of the cases of the recursive definition for $\mathcal{F}$ and the result is computed accordingly. Here $\mathtt{P}_F = \mathtt{p}_F(\overline{x})$ is a (fresh) definitional predicate for F and $\overline{x} = \mathit{free}[F]$. For example, the function $[\cdot]^{def}$ for the set of first-order formulas in negation normal form defined by (30) has the following form:

$$\begin{array}{rcll} [F]^{def} := [A]^{def} & = & \forall \overline{x}.(\mathtt{P}_F \rightarrow A) & \mid \\ [\neg A]^{def} & = & \forall \overline{x}.(\mathtt{P}_F \rightarrow \neg A) & \mid \\ [F_1 \vee F_2]^{def} & = & \forall \overline{x}.(\mathtt{P}_F \rightarrow \mathtt{P}_{F_1} \vee \mathtt{P}_{F_2}) \wedge [F_1]^{def} \wedge [F_2]^{def} & \mid \\ [F_1 \wedge F_2]^{def} & = & \forall \overline{x}.(\mathtt{P}_F \rightarrow \mathtt{P}_{F_1} \wedge \mathtt{P}_{F_2}) \wedge [F_1]^{def} \wedge [F_2]^{def} & \mid \\ [\forall y.F_1]^{def} & = & \forall \overline{x}.(\mathtt{P}_F \rightarrow \forall y.\mathtt{P}_{F_1}) \wedge [F_1]^{def} & \mid \\ [\exists y.F_1]^{def} & = & \forall \overline{x}.(\mathtt{P}_F \rightarrow \exists y.\mathtt{P}_{F_1}) \wedge [F_1]^{def} \ . & \end{array} \tag{32}$$

Remark 6.4. Please note that the structural transformation is defined not for a formula, but for its *recursive definition*. A formula might be represented by many different recursive definitions, and consequently might have several different results of the structural transformation. Note also that the structural transformation (32) applied to a formula in **NNF** produces a formula in **NNF** (according to our convention $A \rightarrow B$ is a shortcut for $\neg A \vee B$). ⋘

[17] The structural transformation can be also defined for negative occurances of subformulas, however we do not need this since we put formulas into negation normal form first

Note that the result $[G]^{str}$ of the structural transformation for a formula $G \in \mathcal{F}$ computed according to the definitions in Figure 22, can be written as follows:

$$[G]^{str} \;=\; \mathtt{P}_G \wedge \bigwedge_{F=B_i} \forall \overline{x}.(\mathtt{P}_F \rightarrow B_i) \wedge \bigwedge_{F \;=\; R_j[F_1,..,F_{k_j}]} \forall \overline{x}.(\mathtt{P}_F \rightarrow R_j[\mathtt{P}_{F_1},..,\mathtt{P}_{F_{k_j}}]), \qquad (33)$$

where the conjunctions are taken over subformulas F of G. In particular, the total number of conjuncts is at most $|G|$. Now we give some other properties for the structural transformation.

Lemma 6.5. *Let a formula set $\mathcal{F}$ and a function $[\cdot]^{def}$ be defined like in Figure 22. Let $F[G]$ be a first-order formula with positive occurrences of a subformula $G \in \mathcal{F}$. Then $F[G/\mathtt{P}_G] \wedge [G]^{def}$ is conservative over $F[G]$.*

Proof. We prove Lemma by induction over the construction of $G \in \mathcal{F}$. For a base case $G = B_i$, $1 \leq i \leq n$ by Proposition 6.3, $F[\mathtt{P}_G] \wedge \forall x.(\mathtt{P}_G \rightarrow G) = F[\mathtt{P}_G] \wedge [G]^{def}$ is conservative over $F[G]$.

To prove Lemma for the induction step corresponding to a constructor R_j with $1 \leq j \leq m$, i.e. for $G = R_j[G_1,..,G_{k_j}]$ (see Figure 22), assume by induction hypothesis that Lemma holds for the formulas $G_1, \ldots, G_{k_j}$. That is, for any $F[G_i]$ with positive occurrences of G_i, $F[\mathtt{P}_{G_i}] \wedge [G_i]^{def}$ is conservative over $F[G_i]$, $1 \leq i \leq k_j$. Then of the following sequence of formulas, each formula is conservative over the previous one:

$$\begin{aligned}
&F[G] = F[R_j[G_1,..,G_{k_j}]] && (\Rightarrow \text{by Proposition 6.3})\\
&F[\mathtt{P}_G] \wedge \forall \overline{x}.(\mathtt{P}_G \rightarrow R_j[G_1,..,G_{k_j}]) && (\Rightarrow \text{by induction hypothesis})\\
&F[\mathtt{P}_G] \wedge \forall \overline{x}.(\mathtt{P}_G \rightarrow R_j[\mathtt{P}_{G_1},..,\mathtt{P}_{G_{k_j}}]) \wedge [G_1]^{def} \wedge \cdots \wedge [G_{k_j}]^{def} \;=\; F[\mathtt{P}_G] \wedge [G]^{def}.
\end{aligned}$$

□

Proposition 6.6. *Let a formula set $\mathcal{F}$ and a function $[\cdot]^{str}$ be defined like in Figure 22. Then the result $H := [G]^{str}$ of structural transformation for $G \in \mathcal{F}$ can be computed in polynomial time in $|G|$ and produces a formula H such that (i) H is conservative over G and (ii) $|H| = O(w \cdot |G|)$, where $w := width(G)$.*

Proof. Given a formula $G \in \mathcal{F}$, the function $[G]^{str}$ is computed with at most $|G|$ recursive calls. By induction on G it is easy to show that every recursion step can be done in polynomial time (the polynomial depends on the data structure used to represent formulas).

The property (i) follows from Lemma 6.5, when we take $F[G] := G$. To show the property (ii), note that every conjunct in (33) has the size at most $O(w)$, where $w := width(G)$, since formulas E_i with $1 \leq i \leq n$ and operators $\mathtt{R_j}$ with $1 \leq j \leq m$ are fixed for the formula set $\mathcal{F}$ and the size of every definitional

predicate $|\mathtt{P}_F| \leq w+1$ (which is the number of variables in the subformula F of G plus the size of the predicate symbol $\mathtt{p}_F$). Since the number of such conjuncts is at most $|G|$, we conclude that $|[G]^{str}| = O(w{\cdot}|G|)$. ⧉

Example 6.7. Let us now perform the structural transformation for the result of **NNF**- transformation $G := [F]^{nnf} = (\exists y.(\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y)) \vee \exists z.\mathtt{c}(x,z)) \in [\mathcal{FO}]^{nnf}$ computed in Example 6.2. The result of structural transformation for G is the formula $[G]^{str} = \mathtt{P}_G \wedge [G]^{def}$, where $\mathtt{P}_G = \underline{\mathtt{p}_0(x)}$ ($\mathtt{p}_0$ is a fresh predicate symbol introduced for G), and $[G]^{def}$ is computed according to the definition (32) as follows:

$$\begin{aligned}
[G]^{def} &= [\exists y.(\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y)) \vee \exists z.\mathtt{c}(x,z)]^{def} \\
&= \underline{\forall x.(\mathtt{p}_0(x) \rightarrow (\mathtt{p}_1(x) \vee \mathtt{p}_2(x)))} \wedge [\exists y.(\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y))]^{def} \wedge [\exists z.\mathtt{c}(x,z)]^{def};
\end{aligned}$$

($\mathtt{p}_1(x)$ is introduced for $\exists y.(\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y))$; $\mathtt{p}_2(x)$ is introduced for $\exists z.\mathtt{c}(x,z)$)

$$[\exists y.(\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y))]^{def} = \underline{\forall x.(\mathtt{p}_1(x) \rightarrow \exists y.\mathtt{p}_3(x,y))} \wedge [\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y)]^{def};$$

($\mathtt{p}_3(x,y)$ is introduced for $\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y)$)

$$\begin{aligned}
[\neg\mathtt{a}(x) \vee \neg\mathtt{b}(y)]^{def} &= \forall xy.(\mathtt{p}_3(x,y) \rightarrow \mathtt{p}_4(x) \vee \mathtt{p}_5(y)) \wedge [\neg\mathtt{a}(x)]^{def} \wedge [\neg\mathtt{b}(y)]^{def} \\
&= \underline{\forall xy.(\mathtt{p}_3(x,y) \rightarrow \mathtt{p}_4(x) \vee \mathtt{p}_5(y))} \wedge \underline{\forall x.(\mathtt{p}_4(x) \rightarrow \neg\mathtt{a}(x))} \\
&\qquad \wedge \underline{\forall y.(\mathtt{p}_5(y) \rightarrow \neg\mathtt{b}(y))}
\end{aligned}$$

($\mathtt{p}_4(x)$ is introduced for $\neg\mathtt{a}(x)$; $\mathtt{p}_5(y)$ is introduced for $\neg\mathtt{b}(y)$)

$$\begin{aligned}
[\exists z.\mathtt{c}(x,z)]^{def} &= \forall x.(\mathtt{p}_2(x) \rightarrow \exists z.\mathtt{p}_6(x,z)) \wedge [\mathtt{c}(x,z)]^{def} \\
&= \underline{\forall x.(\mathtt{p}_2(x) \rightarrow \exists z.\mathtt{p}_6(x,z))} \wedge \underline{\forall xz.(\mathtt{p}_6(x,z) \rightarrow \mathtt{c}(x,z))}
\end{aligned}$$

($\mathtt{p}_6(x,z)$ is introduced for $\mathtt{c}(x,z)$).

The result of the structural transformation for G is the conjunction of the underlined formulas:

$$\begin{aligned}
[G]^{str} = \mathtt{p}_0(x) \wedge \forall x.[\mathtt{p}_0(x) \rightarrow (\mathtt{p}_1(x) \vee \mathtt{p}_2(x))] \wedge \forall x.[\mathtt{p}_1(x) \rightarrow \exists y.\mathtt{p}_3(x,y)] \wedge \\
\forall xy.[\mathtt{p}_3(x,y) \rightarrow \mathtt{p}_4(x) \vee \mathtt{p}_5(y)] \wedge \forall x.[\mathtt{p}_4(x) \rightarrow \neg\mathtt{a}(x)] \wedge \forall y.[\mathtt{p}_5(y) \rightarrow \neg\mathtt{b}(y)] \wedge \\
\forall x.[\mathtt{p}_2(x) \rightarrow \exists z.\mathtt{p}_6(x,z)] \wedge \forall xz.[\mathtt{p}_6(x,z) \rightarrow \mathtt{c}(x,z)]. \quad (34)
\end{aligned}$$

which is of the form (33).

Although the result looks more complicated to a human eye than the input formula, it is more easy to process by a theorem prover since every conjunct now has a very simple form. ⧈

6.3 Skolemization

In the next step of **CNF**-transformation the existentially quantified variables of a formula are skolemized. For the purpose of obtaining saturation-based decision procedures one usually uses the standard outermost Skolemization. Given a formula F in negation normal (30), the result $[F]_i^{sk}/[F]_o^{sk}$ of applying the *innermost/outermost Skolemization* to F is recursively defined in subsection 6.3, where

Figure 23 Skolemization for first-order formulas in **NNF**

$$
\begin{array}{rcl|l|rcl|l}
[F]_*^{sk} := [A]_*^{sk} & = & A & & [\forall y.F_1]_*^{sk} & = & \forall y.[F_1]_*^{sk} & * \in \{i,o\} \\
[\neg A]_*^{sk} & = & \neg A & & [\exists y.F_1]_i^{sk} & = & [F_1]_i^{sk}\cdot\{y/\mathtt{f}_F(\overline{x})\} & \text{(innermost Skolemization)} \\
[F_1 \vee F_2]_*^{sk} & = & [F_1]_*^{sk} \vee [F_2]_*^{sk} & & & & \text{or} & \\
[F_1 \wedge F_2]_*^{sk} & = & [F_1]_*^{sk} \wedge [F_2]_*^{sk} & & [\exists y.F_1]_o^{sk} & = & [F_1\cdot\{y/\mathtt{f}_F(\overline{x})\}]_o^{sk} & \text{(outermost Skolemization)}
\end{array}
$$

$$
\begin{array}{rcl|rcl}
[F]_m^{sk} := [A]_m^{sk} & = & A & [\forall y.F_1]_m^{sk} & = & \forall y.[F_1]_m^{sk} \\
[\neg A]_m^{sk} & = & \neg A & [\exists y.F_1]_m^{sk} & = & [F_1]_m^{sk}\cdot\{y/\mathtt{f}_F(\overline{x})\} \mid [F_1\cdot\{y/\mathtt{f}_F(\overline{x})\}]_m^{sk} \\
[F_1 \vee F_2]_m^{sk} & = & [F_1]_m^{sk} \vee [F_2]_m^{sk} & & & \text{(mixed Skolemization)} \\
[F_1 \wedge F_2]_m^{sk} & = & [F_1]_m^{sk} \wedge [F_2]_m^{sk} & &
\end{array}
$$

A is an atom, F_1, $F_2 \in [\mathcal{FO}]^{nnf}$ and $\mathtt{f}_F(\overline{x})$ is a *Skolem function* introduced for the existentially quantified formula $F = \exists y.F_1$ over its free variables $\overline{x} = \mathit{free}[F]$. According to this definition, Skolemization replaces every occurrence of an existentially quantified variable y in F_1 by a Skolem function $\mathtt{f}_F(\overline{x})$. The difference between innermost and outermost Skolemizations, is that the first method performs Skolemization "from inside out": skolemizes a subformula and then performs a substitution, whereas the second method applies a substitution immediately and then proceed skolemizing subformulas. In the second case one usually obtains smaller Skolem functions, since the number of free variables in a subformula decreases. This is, however, not always the case as can be demonstrated in the following example:

Example 6.8. Let us skolemize the formula $F = \exists z.(\mathtt{a}(x,y,z) \wedge \exists u.\mathtt{b}(z,u))$ in the innermost and outermost way:

$$
\begin{array}{ll}
[F]_i^{sk} = & [F]_o^{sk} = \\
= [\exists z.(\mathtt{a}(x,y,z)\wedge\exists u.\mathtt{b}(z,u))]_i^{sk} & = [\exists z.(\mathtt{a}(x,y,z)\wedge\exists u.\mathtt{b}(z,u))]_o^{sk} \\
= [\mathtt{a}(x,y,z)\wedge\exists u.\mathtt{b}(z,u)]_i^{sk}\cdot\{z/\mathtt{f}_1(x,y)\} & = [(\mathtt{a}(x,y,z)\wedge\exists u.\mathtt{b}(z,u))\cdot\{z/\mathtt{f}_1(x,y)\}]_o^{sk} \\
= \mathtt{a}(x,y,z)\wedge[\exists u.\mathtt{b}(z,u)]_i^{sk}\cdot\{z/\mathtt{f}_1(x,y)\} & = [\mathtt{a}(x,y,\mathtt{f}_1(x,y)\wedge\exists u.\mathtt{b}(\mathtt{f}_1(x,y),u)]_o^{sk} \\
= \mathtt{a}(x,y,z)\wedge[\mathtt{b}(z,u)]_i^{sk}\cdot\{u/\mathtt{f}_2(z)\}\cdot\{z/\mathtt{f}_1(x,y)\} & = \mathtt{a}(x,y,\mathtt{f}_1(x,y))\wedge[\exists u.\mathtt{b}(\mathtt{f}_1(x,y),u)]_o^{sk} \\
= \mathtt{a}(x,y,z)\wedge\mathtt{b}(z,u)\cdot\{u/\mathtt{f}_2(z)\}\cdot\{z/\mathtt{f}_1(x,y)\} & = \mathtt{a}(x,y,\mathtt{f}_1(x,y))\wedge[\mathtt{b}(\mathtt{f}_1(x,y),u)]_o^{sk}\cdot\{u/\mathtt{f}_3(x,y)\} \\
= \mathtt{a}(x,y,z)\wedge\mathtt{b}(z,\mathtt{f}_2(z))\cdot\{z/\mathtt{f}_1(x,y)\} & = \mathtt{a}(x,y,\mathtt{f}_1(x,y))\wedge[\mathtt{b}(\mathtt{f}_1(x,y),\mathtt{f}_3(x,y))]_o^{sk} \\
= \mathtt{a}(x,y,\mathtt{f}_1(x,y))\wedge\mathtt{b}(\mathtt{f}_1(x,y),\underline{\mathtt{f}_2(\mathtt{f}_1(x,y))}) & = \mathtt{a}(x,y,\mathtt{f}_1(x,y))\wedge\mathtt{b}(\mathtt{f}_1(x,y),\underline{\mathtt{f}_3(x,y)})
\end{array}
$$

where $\mathtt{f}_1(x,y)$ is a Skolem function introduced for the formula F; $\mathtt{f}_2(z)$ is a Skolem function introduced for the subformula $\exists u.\mathtt{b}(z,u)$ and $\mathtt{f}_3(x,y)$ is a Skolem function introduced for the formula $\exists u.\mathtt{b}(\mathtt{f}_1(x,y),u)$. Note that the results of innermost and outermost Skolemizations differ in underlined terms $\mathtt{f}_2(\mathtt{f}_1(x,y))$ and $\mathtt{f}_3(x,y)$, which semantically denote the same function. ⋘

Innermost Skolemization, can lead to exponentially large results in worst case which is demonstrated in a simple example below. Therefore, without usage of special data-structures, that allow for shearing of terms, innermost Skolemization is not very useful in practice, except for situations where it allows one to reuse Skolem functions (see Remark 6.12 below).

Example 6.9. Let us find the results of innermost and outermost Skolemizations for the formula $A = \exists x_1.\exists x_2 ... \exists x_n.\mathtt{a}(x_1,x_2,..,x_n)$. The outermost Skolemization yields the formula

$$\mathtt{a}(\mathtt{c}_1,\mathtt{c}_2,..,\mathtt{c}_n),$$

where for each i with $1 \leq i \leq n$, the Skolem constant $\mathtt{c}_i$ is introduced for the formula

$$\exists x_i ... \exists x_n.\mathtt{a}(\mathtt{c}_1,..,\mathtt{c}_{i-1},x_i,..,x_n).$$

The innermost Skolemization is computed as follows:

$$\begin{aligned}
[A]_i^{sk} &= [\exists x_2 ... \exists x_n.\mathtt{a}(x_1,x_2,..,x_n)]_i^{sk}\cdot\{x_1/\mathtt{c}_1\}\\
&= [\exists x_3 ... \exists x_n.\mathtt{a}(x_1,x_2,..,x_n)]_i^{sk}\cdot\{x_2/\mathtt{f}_2(x_1)\}\cdot\{x_1/\mathtt{c}_1\}\\
&= \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&= [\exists x_{i+1} ... \exists x_n.\mathtt{a}(x_1,x_2,..,x_n)]_i^{sk}\cdot\{x_i/\mathtt{f}_i(x_1,..,x_{i-1})\}\cdot\cdots\cdot\{x_2/\mathtt{f}_2(x_1)\}\cdot\{x_1/\mathtt{c}_1\}\\
&= \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&= \mathtt{a}(x_1,x_2,..,x_n)\cdot\{x_n/\mathtt{f}_n(x_1,..,x_{n-1})\}\cdot\cdots\cdot\{x_i/\mathtt{f}_i(x_1,..,x_{i-1})\}\cdot\cdots\cdot\{x_2/\mathtt{f}_2(x_1)\}\cdot\{x_1/\mathtt{c}_1\}
\end{aligned}$$

where for each i with $1 < i \leq n$, the Skolem function $\mathtt{f}_i(x_{i-1},..,x_1)$ is introduced for the formula $\exists x_i ... \exists x_n.\mathtt{a}(x_1,x_2,..,x_n)$. After computing the last sequence of substitutions we obtain a formula of exponential size in n. Indeed, by induction on i with $1 \leq i \leq n$, it can be shown that after performing the i-th substitution (for the variable x_{n-i+1}), the number of occurrences for each variable x_j with $1 \leq j \leq (n-i)$ becomes 2^i (after applying the substitution for x_{n-i+1} the number of occurrences for the remaining variables gets doubled). In particular the constant $\mathtt{c}_1$ occurs in the result of the Skolemization 2^n times. So, the innermost Skolemization produces an exponentially large formula. ⋘

Because of this dramatical difference in complexity between innermost and outermost Skolemizations (that will be made precise in Proposition 6.10), we will refrain from using innermost Skolemization. Unless stated otherwise, every Skolemization that we use $[F]^{sk}$ is outermost.

Innermost and outermost Skolemizations can be mixed (some quantifiers can be skolemized in innermost others in outermost way). This hybrid Skolemization is denoted by $[F]^{sk}_m$ (see the definition in subsection 6.3). The result of Skolemization is not logically equivalent to the input formula, but preserves its (un)satisfiability.

Proposition 6.10. *For any formula $F \in [\mathcal{FO}]^{nnf}$ the result of mixed Skolemization $[F]^{sk}_m$ (outermost Skolemization $[F]^{sk}_o$) can be computed in exponential (polynomial) time in $|F|$ such that the following holds: (i) $[F]^{sk}_m$ is conservative over F and (ii) $|[F]^{sk}_m| \leq (w+1)^e \cdot |F|$; $|[F]^{sk}_o| \leq (w+1)\cdot|F|$, where $w = width(F)$ and e is the number of existential quantifiers in F.*

Proof. In every computation of the function $[F]^{sk}_m$ there are at most $|F|$ recursive call (exactly one for every logical connective in F, since a substitution does not change any of those). Therefore the recursive function $[\cdot]^{sk}_m$ always terminates. Since there are at most linear number of recursion calls, the computation time for Skolemization is mainly influenced by the (intermediate) sizes of formulas, which we estimate below.

To estimate the size of $[F]^{sk}_m$, note that during its computation exactly e substitutions have been applied to a formula (one for every existential quantifier). Every substitution increases the size of the formula in at most $w+1$ times (every Skolem function contains at most w variables). Therefore, the size of the result $|[F]^{sk}_m|$ is bounded by $(w+1)^e \cdot |F|$. For the outermost Skolemization, we can obtain a much lower bound, since one could notice that Skolem functions cannot be nested in the result $[F]^{sk}_o$. Indeed, according to the definition of $[\cdot]^{sk}_o$ given in Figure 23, Skolem functions that are substituted for variables do not contain variables that will be instantiated afterwards. So, the size of the result $|[F]^{sk}_o|$ cannot exceed the value $(w+1)\cdot|F|$, which is polynomial in $|F|$. Hereby, we have proven the point (ii) of Proposition.

The bounds on the sizes of skolemized formulas imply that the procedure computing mixed (outermost) Skolemization can be implemented in exponential (respectively polynomial) time in the size of the input. It remains to show the point (i) of Proposition which we do by induction over $[\cdot]^{sk}_m$.

First, it can be shown that F is a logical consequence of $[F]^{sk}_m$. The base case and induction steps for conjunction and disjunction is trivial. For the case with a universal quantifier, $\forall y.F_1$ is a logical consequence of $[\forall x.F_1]^{sk}_m = \forall x.[F_1]^{sk}_m$ by Replacement Lemma (see Lemma 2.4), since F_1 is a logical consequence of $[F_1]^{sk}_m$ by induction hypothesis. For the case with an existential quantifier, $\exists y.F_1$ is a logical consequence of the formula $F_1 \cdot \{y/\mathtt{f}_F(\overline{x})\}$, which by induction hypothesis is a logical consequence of both $[F_1]^{sk}_m \cdot \{y/\mathtt{f}_F(\overline{x})\}$ and $[F_1 \cdot \{y/\mathtt{f}_F(\overline{x})\}]^{sk}_m$.

Now we prove that any model $\mathcal{M}$ of F can be *expanded* to a model $\mathcal{M}'$ of $[F]^{sk}_m$. We expand any interpretation $\mathcal{M}$ by defining the interpretation of *new*

Skolem functions as follows:

$$\mathtt{f}^{\mathcal{M}'}_{\exists y.F_1}(d_1, d_2,.., d_n) := \begin{cases} \text{some } d & \text{s.t. } \mathcal{M} \models F_1 \cdot \{y/d, x_1/d_1,.., x_n/d_n\} \\ d_1 & \text{if } \mathcal{M} \models F_1 \cdot \{y/d, x_1/d_1,.., x_n/d_n\} \text{ for no } d \in D \end{cases}$$

where $x_1, x_2,.., x_n$ are all free variables of F_1. Note, that this definition relied on *Axiom of Choice* [for a related discussion concerning the rôle of Axiom of Choice and Skolemization in saturation-based theorem proving see de Nivelle, 2003]. Using the above definition, by induction over $[\cdot]^{sk}_m$, it is straightforward to show that for any first-order interpretation $\mathcal{M}$ and a valuation of variables η, there exists an expansion $\mathcal{M}'$ of $\mathcal{M}$ such that $\mathcal{M}, \eta \models F$ implies $\mathcal{M}', \eta \models [F]^{sk}_m$. ⧉

Before applying Skolemization we make one more step, which is the *existential closure* of a formula. We replace a formula $G \in [\mathcal{FO}]^{nnf}$ with $\exists \overline{x}.G$, where $\overline{x} = \mathit{free}[G]$. For example, the formula F in Example 6.8 should have been existentially closed to $\exists xy.F$. This step also preserves (un)satisfiability of a formula. From now on we assume that the formula to be skolemized is a sentence (that is, it does not contain free variables).

Example 6.11. Let us perform existential closure and Skolemization for the result $H := [G]^{str}$ (34) computed in Example 6.7. The existential closure yields the formula $\exists x.H$, since $\{x\} = \mathit{free}[H]$, skolemizing which we obtain the formula:

$$[H]^{sk} = \mathtt{p}_0(\underline{\mathtt{c}_1}) \wedge \forall x.[\mathtt{p}_0(x) \rightarrow (\mathtt{p}_1(x) \vee \mathtt{p}_2(x))] \wedge \forall x.[\mathtt{p}_1(x) \rightarrow \mathtt{p}_3(x, \underline{\mathtt{f}_1(x)})] \wedge$$
$$\forall xy.[\mathtt{p}_3(x, y) \rightarrow \mathtt{p}_4(x) \vee \mathtt{p}_5(y)] \wedge \forall x.[\mathtt{p}_4(x) \rightarrow \neg\mathtt{a}(x)] \wedge \forall y.[\mathtt{p}_5(y) \rightarrow \neg\mathtt{b}(y)] \wedge$$
$$\forall x.[\mathtt{p}_2(x) \rightarrow \mathtt{p}_6(x, \underline{\mathtt{f}_2(x)})] \wedge \forall xz.[\mathtt{p}_6(x, z) \rightarrow \mathtt{c}(x, z)]. \quad (35)$$

where the Skolem constant $\mathtt{c}_1$ is introduced for the formula $\exists x.H$, the Skolem function $\mathtt{f}_1(x)$ is introduced for the subformula $\exists y.\mathtt{p}_3(x, y)$ of H and the Skolem function $\mathtt{f}_2(x)$ is introduced for the subformula $\exists z.\mathtt{p}_6(x, z)$ of H. ⋘

Remark 6.12. Both in structural transformation and in Skolemization the introduced predicate symbols, respectively Skolem functions can be *reused*, if the formulas for which they are introduce are equivalent (modulo variable renaming). For example, given the formula:
$F = \forall x.\exists y.\mathtt{a}(x, y) \wedge \exists z.\exists y.\mathtt{a}(z, y)$, one can introduce a definitional predicate $\mathtt{p}_1(x)$ for the subformula $\exists y.\mathtt{a}(x, y)$ and reuse it for the subformula $\exists y.\mathtt{a}(z, y)$ as follows:

$$[F]^{str} = \forall x.\mathtt{p}_1(x) \wedge \exists z.\mathtt{p}_1(z) \wedge \forall x.[\mathtt{p}_1(x) \rightarrow \exists y.\mathtt{a}(x, y)].$$

Alternatively one can introduce a Skolem function $\mathtt{f}_1(x)$ for the subformula $\exists y.\mathtt{a}(x, y)$ and reuse it for the subformula $\exists y.\mathtt{a}(z, y)$ as follows (we have used the innermost Skolemization):

$$[F]^{sk} = \forall x.\mathtt{a}(x, \mathtt{f}_1(x)) \wedge \mathtt{a}(\mathtt{c}_1, \mathtt{f}_1(\mathtt{c}_1))$$

where the Skolem constant c_1 is introduced for the subformula $\exists z.\exists y.\mathtt{a}(z,y)$. This re-usage technique can further refine the structural transformation and Skolemization which yield smaller **CNF**'s in many situations [see Nonnengart & Weidenbach, 2001]. ⟐

6.4 Clausification

Clausification is probably the most expensive transformation in computation of a clause normal form. Like Skolemization it can produce exponentially large results, if not used carefully.

After Skolemization step we may assume that all existentially quantified variables are eliminated and only universally quantified variables left (remember that our formula is closed). We drop the universal quantifiers (first, renaming all bounded variables apart), since they do not carry any information anymore. Thus, we assume that all variables of the resulted expression are implicitly universally quantified. The obtained formula has the following simple form:

$$\mathcal{P} ::= L \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \ . \tag{36}$$

where $L ::= a(t_1, t_2,.., t_n) \mid \neg a(t_1, t_2,.., t_n)$ is a *literal*, that is, an atom or its negation, whose arguments are first-order terms. Now an expression of the form (36) is translated to a set of clauses using a *clausification* transformation. A quantifier-free first order formula F in negation normal form is in *conjunction normal form* (also denoted **CNF** since there is not much difference with clause normal form) if any subformula of the form $F_1 \vee F_2$ of F does not contain conjunctions. During the clausification step, the conjunctions are repeatedly distributed over disjunctions using the following rewrite rules applied to subformulas of a formula:

$$A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C) \quad \text{and} \quad (A \wedge B) \vee C \Rightarrow (A \vee C) \wedge (B \vee C)$$

There are different ways to show that this transformation terminates. The transformation rules above can be oriented using an *LPO*-ordering [see Bachmair & Ganzinger, 2001], however this does not give one any complexity bounds for the transformation. In order to estimate the time spent on clausification and the size of the result, we compute the result $[F]^{cnf}$ of **CNF** transformation using the function $[\cdot \mid \cdot]^{cnf}$ defined in Figure 24. The function $[\cdot \mid \cdot]^{cnf}$ is defined recursively over the first argument $F \in \mathcal{P}$ of the form (36). The second argument is used for bookkeeping a disjunction of literals that are cut from the formula F while decomposing it.

The main difficulty in proving termination of this function (and estimating its computational cost) arises in the last recursion step, which normalizes a formula modulo associativity of disjunction. For this purpose we introduce a new weight

Figure 24 Clausification for quantifier-free first-order formulas

$$
\begin{array}{rrcl}
[F]^{cnf} := [F \,|\, \bot]^{cnf}; & [F \,|\, C]^{cnf} := [L \,|\, C]^{cnf} & = & C \vee L \;\; | \\
& [F_1 \vee L \,|\, C]^{cnf} & = & [C \vee L]^{cnf} F_1 \;\; | \\
& [F_1 \wedge F_2 \,|\, C]^{cnf} & = & [F_1 \,|\, C]^{cnf} \wedge [F_2 \,|\, C]^{cnf} \;\; | \\
& [F_1 \vee (F_2 \wedge F_3) \,|\, C]^{cnf} & = & [F_1 \vee F_2 \,|\, C]^{cnf} \wedge [F_1 \vee F_3 \,|\, C]^{cnf} \;\; | \\
& [F_1 \vee (F_2 \vee F_3) \,|\, C]^{cnf} & = & [(F_1 \vee F_2) \vee F_3 \,|\, C]^{cnf} \; .
\end{array}
$$

$$
\begin{array}{rcl|rcl}
|F|_{cnf} := |L|_{cnf} & = & 1 \;\; | & |F|^1_{cnf} := |L|^1_{cnf} & = & 1 \;\; | \\
|F_1 \vee F_2|_{cnf} & = & |F_1|_{cnf} + |F_2|^{\mathbf{1}}_{cnf} + 1 \;\; | & |F_1 \vee F_2|^1_{cnf} & = & |F_1|^1_{cnf} + |F_2|^1_{cnf} + \mathbf{2} \;\; | \\
|F_1 \wedge F_2|_{cnf} & = & |F_1|_{cnf} + |F_2|_{cnf} + 1 \; . & |F_1 \wedge F_2|^1_{cnf} & = & |F_1|^1_{cnf} + |F_2|^1_{cnf} + 1 \; .
\end{array}
$$

function $|\cdot|_{cnf}$ that is defined in Figure 24. The weight function $|F|_{cnf}$ sums the weights literals, conjunction and disjunction symbols in F. All these symbols have a weight 1 unless it is a disjunction symbol that appears inside the right disjunct F_2 of some subformula $F_1 \vee F_2$ of F. In this case the weight of the disjunction symbol is 2 (additional penalty is added for the occurrence in a right disjunct). For example, $|(L_1 \vee L_2) \vee L_3|_{cnf} = 5$, but $|L_1 \vee (L_2 \vee L_3)|_{cnf} = 6$, which makes it possible to orient the last recursion step (associativity of disjunction). Using the weight function $|\cdot|_{cnf}$ we show that clausification terminates in exponential time and produces at most exponentially large **CNF**.

Proposition 6.13. *For any formula $F \in \mathcal{P}$ defined according to* (36)*, the result of clausification $S := [F]^{cnf}$ can be computed in polynomial time in $2^{|F|}$ and produces a formula S in* **CNF** *such that (i) $S \equiv F$ and (ii) $|S| \leq 2^{|F|}$.*

Proof. First we show that for any formula $F \in \mathcal{P}$ there are at most $2^{2 \cdot |F|}$ recursive calls of the function $[\cdot \,|\, \cdot]^{cnf}$ possible during the computation of $[F]^{cnf}$. Observe that in every recursion call the weight of the first parameter strictly decreases:

$$
\begin{array}{rl}
|F_1 \vee L|_{cnf} = & |F_1|_{cnf} + |L|^1_{cnf} + 1 > |F_1|_{cnf}; \\
|F_1 \wedge F_2|_{cnf} = & |F_1|_{cnf} + |F_2|_{cnf} + 1 > \max(|F_1|_{cnf}, |F_2|_{cnf}); \\
|F_1 \vee (F_2 \wedge F_3)|_{cnf} = & |F_1|_{cnf} + (|F_2|^1_{cnf} + |F_3|^1_{cnf} + 1) + 1 > \\
& > |F_1|_{cnf} + \max(|F_2|^1_{cnf}, |F_3|^1_{cnf}) + 1 = \\
& = \max(|F_1 \vee F_2|_{cnf}, |F_1 \vee F_3|_{cnf}); \\
|F_1 \vee (F_2 \vee F_3)|_{cnf} = & \; |F_1|_{cnf} + (|F_2|^1_{cnf} + |F_3|^1_{cnf} + 2) + 1 > \\
& > (|F_1|_{cnf} + |F_2|^1_{cnf} + 1) + |F_3|^1_{cnf} + 1 = |(F_1 \vee F_2) \vee F_3|_{cnf}.
\end{array}
$$

Note also that $|F|_{cnf} < 2 \cdot |F|$ for every $F \in \mathcal{P}$.

The computation tree for $[F \,|\, \cdot]^{cnf}$ is a binary tree: every recursion step fires at most two recursive calls of the function $[\cdot \,|\, \cdot]^{cnf}$. On every branch of this tree there are at most $|F|_{cnf} < 2 \cdot |F|$ recursion calls (the weight function cannot decrease

more than $|F|_{cnf}$ times). Therefore, the size of the recursion tree and the total number of recursion calls used in computation of $[F]^{cnf}$ is bounded by $2^{2\cdot|F|}$. Below we will show that the function produces the output of the size at most $2^{|F|}$. This will imply that clausification can be computed in polynomial time in $2^{|F|}$ (the polynomial overhead is caused by manipulations with intermediate results, that depend on a data structure implementation).

Since we have proved that the function $[\cdot \mid \cdot]^{cnf}$ terminates, we may use induction over it. By such an induction now it is easy to show that the result of clausification is equivalent to the input formula and has at most exponential size in the input formula. More precisely, (i) $[F \mid C]^{cnf} \equiv C \vee F$ and (ii) $|[F \mid C]^{cnf}| < (|C|+2)\cdot 2^{|F|}$. This respectively imply the parts (i) and (ii) of Proposition. ⧉

Clausification of a formula $F \in \mathcal{P}$ produces a conjunction of disjunctions. We drop the conjunctions between disjunctions and view the result as sets of clauses. All clauses have implicitly disjoint variables (which are implicitly universally quantified), so we can reuse variable names.

Example 6.14. By applying clausification for the formula computed in Example 6.11, we obtain the following set of clauses:

$$\begin{array}{ll} \mathtt{p}_0(\mathtt{c}_1); & \neg\mathtt{p}_4(x) \vee \neg\mathtt{a}(x); \\ \neg\mathtt{p}_0(x) \vee \mathtt{p}_1(x) \vee \mathtt{p}_2(x); & \neg\mathtt{p}_5(y) \vee \neg\mathtt{b}(y); \\ \neg\mathtt{p}_1(x) \vee \mathtt{p}_3(x, \mathtt{f}_1(x)); & \neg\mathtt{p}_2(x) \vee \mathtt{p}_6(x, \mathtt{f}_2(x)); \\ \neg\mathtt{p}_3(x,y) \vee \mathtt{p}_4(x) \vee \mathtt{p}_5(y); & \neg\mathtt{p}_6(x,z) \vee \mathtt{c}(x,z). \end{array}$$

⧫

Remark 6.15. Although clausification is exponential in worst case (see example (31)), most translations that we give for fragments are polynomial. This is mainly because of the structural transformation step which is applied before clausification. Since the structural transformation produces a formula of the form (33), we need to perform clausification only for conjuncts of these formula, which have a fixed form for a fragment. Therefore, every conjunct is translated into **CNF** with a linear overhead in the size and clausification can be computed in polynomial time. ⧫

6.5 Summary for CNF-Transformations

Complexity estimations for the steps of **CNF**-transformation are summarized in Table 1. As seen from the table, the most expensive steps of the transformation are Skolemization and clausification, which are exponential in worst case. Note that this does not really imply that **CNF**-transformation is doubly exponential in worst case, since Skolemization produces possibly exponentially large literals, but

Table 1 Summary for complexity of **CNF** transformations

\Transformation / Complexity\	**NNF** $\dashrightarrow[\ ]^{nnf}\dashrightarrow$	**Structural** $\dashrightarrow[\ ]^{str}\dashrightarrow$	**Ex. Closure** $F\dashrightarrow\exists\overline{x}.F$	**Skolemization** $\dashrightarrow[\ ]^{sk}\dashrightarrow$	**Clausification** $\dashrightarrow[\ ]^{cnf}\dashrightarrow$
Size	$2{\cdot}\lvert F\rvert$	$O(w{\cdot}\lvert F\rvert)$	$w+\lvert F\rvert$	$(w+1)^e{\cdot}\lvert F\rvert$ $(w+1)\cdot\lvert F\rvert^*$	$2^{\lvert F\rvert}$ $O(\lvert F\rvert)^\dagger$
Time	$p(\lvert F\rvert)$	$p(\lvert F\rvert)$	$p(\lvert F\rvert)$	$p((w+1)^e{\cdot}\lvert F\rvert)$ $p(\lvert F\rvert)^*$	$p(2^{\lvert F\rvert})$ $p(\lvert F\rvert)^\dagger$

where $w = \mathit{width}(F)$; *e = the number of existential quantifiers in F;* $p(\cdot)$ *is a polynomial function.*

*for outermost Skolemization

†in combination with structural transformation

leaves the number of boolean connectives polynomial, whereas clausification exponentially increases the number of boolean connectives leaving literals unchanged. Thus, the result of **CNF**-transformation is at most exponential in the size of the input formula.

In decision procedures one typically employs polynomial **CNF**-transformation. This is achieved by (i) using the outermost skolemization and (ii) applying the structural transformation before clausification step (see Remark 6.15). In fact, in most procedures, **CNF**-transformation produces a result of the size $O(w{\cdot}|F|)$, where w is the width of the input formula F.

7 The Theorem Proving Process

In this section we revisit the saturation process considered in subsection 3.4. So far, saturation process has been described from an abstract viewpoint (see Figure 5), now we address it from pragmatic side. We discuss how a saturation-based strategy can be *implemented*. Here we mainly address the following two questions: (**1**) how redundancy can be practically used in theorem provers and (**2**) how deduction of clauses can be organised in a fair way? In order to answer these and other questions, we consider a simple model of a saturation-based theorem prover. This model will be used for deriving complexity bounds for decision procedures that we present afterwards.

7.1 Simplification Rules

The calculi introduced in previous sections form a core inference system in saturation-based theorem provers. The inference rules of these calculi are usually called the *deduction rules*. Besides deduction rules, most calculi employ additional *simplification rules* during a saturation process. Simplification rules implement deletion of redundant clauses and *clause rewriting* (i.e., replacing clauses with simpler ones). Their purpose is to keep the search space of a theorem prover manageable, i.e., to keep as small number of clauses as possible. In fact, modern theorem provers such as VAMPIRE [Riazanov & Voronkov, 2002] or SPASS [Weidenbach et al., 2002], spend most of their computation time on simplification and not on deduction.

The general form of simplification rules that we consider in this report is specified by

A Simplification Rule

$$\mathtt{SR}: \frac{S \cup [\![S']\!]}{S_1 \parallel \cdots \parallel S_n} \tag{37}$$

$$\big[\textit{where the conditions of the rule hold}\big]$$

where S, S', $S_1, .., S_n$ are clause sets. Some premises of a simplification rule might be enclosed in brackets $[\![\cdots]\!]$, which means that they should be deleted after this inference is applied. Simplification rules might be *multi-conclusion*, i.e., possibly several clauses can be produced in an inference. We also admit *nondeterministic* inference rules, i.e., when there are several choices between conclusions of an inference rules called the *possible conclusions* of the rule (here we separate these choices with $\parallel$).

Definition 7.1 (Admissible Simplification Rules).
We say that a simplification rule $R: S \cup [\![S']\!] \vdash S_1 \parallel \cdots \parallel S_n$ is *sound* if every

model for $S \cup S'$ is a model for some S_i with $1 \leq i \leq n$. R is *compatible with a redundancy criterion* $\mathbf{R} = (\mathsf{R}_{\mathrm{Cl}}(\cdot), \mathsf{R}_{\mathrm{Inf}}(\cdot))$ if for every i with $1 \leq i \leq n$, we have $S' \subseteq \mathsf{R}_{\mathrm{Cl}}(S \cup S_i)$. Finally, we say that a simplification rule R is *admissible* for a calculus $\boldsymbol{\mathcal{C}}$ based on a redundancy criterion $\mathbf{R}$ if R is sound and compatible with $\mathbf{R}$. ⋘

Soundness of simplification rules is required to ensure that a rule can be always applied such that it preserves satisfiability of a clause set. Additionally, deletion of premises must be performed according to a redundancy criterion in order to retain refutational completeness. In other words, application of simplification rules should not prevent from deriving the empty clause from an unsatisfiable clause set. The type of nondeterminism in simplification rules is a "*don't know*" nondeterminism, meaning that only a certain choice of possible conclusions (which is *a-priori* not known) preserves satisfiability of a clause set. This makes a saturation process also nondeterministic (i.e., with backtracking).

Figure 25 Some simplification rules used in saturation-based theorem provers

Tautology Deletion

$$\mathtt{TD}: \frac{[\![C \vee A \vee \neg A]\!]}{}$$

[—]

Subsumption Deletion

$$\mathtt{SD}: \frac{C \quad [\![D]\!]}{}$$

[*where (i) C strictly subsumes D*]

Elimination of Duplicate Literals

$$\mathtt{ED}: \frac{[\![C \vee L \vee L]\!]}{C \vee L}$$

[—]

Splitting

$$\mathtt{SP}: \frac{[\![C \vee D]\!]}{C \parallel D}$$

[*where (i) $C \neq \square$; (ii) $D \neq \square$ and (iii) $vars[C] \cap vars[D] = \{\}$.*]

In figure 25 we list some simplification rules which are often used in saturation-based theorem provers.[18] We have already discussed several times the impact of tautology deletion on refutational completeness of some calculi. The Tautology Deletion rule implements deletion of simple syntactical tautologies. This rule is applied to a clause that has complementary literals and does nothing except that it deletes this clause. Another similar simplification rule is Elimination of Duplicate Literals, which is applied to a clause that has multiple occurrences of some literal and replaces it with a clause in which duplicate occurrences of this literal are removed. The Subsumption Deletion rule is applied to two clauses, one of which strictly subsumes the other. This rule does not produce any conclusion, but deletes

[18] See [Weidenbach, 2001] for a variety of other simplification rules that are commonly used in saturation-based theorem provers

the subsumed clause. Finally, the Splitting rule is applied to a clause $C \vee D$ that consists of two non-empty variable disjoint parts C and D. If such a clause is `true` in a model, then either C must be `true` or D must be `true` in this model. This rule implements this nondeterministic choice and deletes the premise of this rule. It is easy to check that all simplification rules from Figure 25, are admissible according to Definition 7.1 w.r.t. the standard redundancy criterion with subsumption (see Appendix A.2).

7.1.1 Simplification rules extending a signature

For some saturation-based decision procedures we need to employ simplification rules that might extend a signature. An example of such inference rules is the following rule:

Splitting through New Predicate Symbol

$$\mathsf{SPP}: \frac{C \vee D}{\begin{array}{l} C \vee \quad u_{_C}(\overline{x}) \\ D \vee \neg u_{_C}(\overline{x}) \end{array}} \qquad (38)$$

$$\left[\begin{array}{l} \textit{where (i) } C \neq \square\textit{; (ii) } D \neq \square\textit{; (iii) } \mathit{vars}[C] \cap \mathit{vars}[D] = \overline{x} \\ \textit{and (iv) } u_{_C} \textit{ is an extended predicate symbol introduced} \\ \textit{for } C. \end{array}\right]$$

This simplification rule splits a clause consisting of two subclauses C and D by introducing a new predicate symbol $u_{_C}(\overline{x})$ over common variables of these subclauses. In case when $u_{_C}(\overline{x})$ is sufficiently small in the ordering, the premise of this rule becomes redundant w.r.t. to the conclusions of this rule (obviously, the premise logically follows from the conclusions of this rule). An instance of Splitting through New Predicate Symbol, when C and D do not have variables in common, has been considered in [Riazanov & Voronkov, 2001; de Nivelle, 2001] for simulating the Splitting rule and to avoid backtracking in saturation-based theorem provers.

In order to integrate such inference rules in our framework, we assume that for every *extended predicate or functional symbol* u that might be introduced by a simplification rule, there is a function that given a first-order interpretation $\mathcal{I}$ over the initial signature, expands this interpretation to new signature elements. In other words, interpretation of base elements of a signature, uniquely determines the interpretation for every extended element. We will denote by $u^{\mathcal{I}}$ the interpretation of the symbol u under this expansion of $\mathcal{I}$.

Extension of interpretations for introduced predicate symbols can be done by supplying *first-order definitions* for them. For example, formula $u_{_C}(\overline{x})$ above can

be interpreted as a first-order formula $\exists\overline{y}(\neg C)$, where $\overline{y} = vars[C] \setminus \overline{x}$. It is easy to check that every model for the premise of Splitting through New Predicate Symbol is also a model for the conclusions of this rule, when u_C is interpreted in this way. In other words, this simplification rule is *sound*.

In order to indicate *which* elements from premises determine the interpretation of new predicate and functional symbols, we supply them with indexes. For example, the new predicate symbol u_C introduced in Splitting through New Predicate Symbol is indexed by C, which means that its interpretation is determined only by interpretation of C. In this way, this predicate symbol can be *reused*, for instance in an application of this rule for a clause $C \vee D'$ with $D' \neq D$ and $vars[C] \cap vars[D] = vars[C] \cap vars[D']$.

7.2 A Model of a Saturation Process

In order to estimate complexity of different saturation-based strategies, we describe a simple model of computation for a refutational-based theorem prover. Similar, but more detailed and more practically-oriented models of saturation can be found for instance in [Weidenbach, 2001; Bachmair & Ganzinger, 2001].

Figure 26 A simple model of computation for a saturation-based theorem prover

```
Prover (N)
  S := ({} | {} | N);
  while (U ≠ {} and □ ∉ U) do
     S := Deduce (S);
  end;
  if (U = {}) then return ("Satisfiable");
  if (□ ∈ U) then return ("Unsatisfiable");
end.
```

state worked-**o**ut

$S = (D \mid O \mid U)$

deleted **u**sable

A *state* of our saturation-based theorem prover is a set of clauses S which is partitioned on three pairwise disjoint subsets D, O and U of *deleted clauses*, *worked-out clauses* and *usable clauses* respectively (we will shortly write $S = (D \mid O \mid U)$). Given an input set of clauses N to be processed, a state of the prover is initialised to $S := (\{\} \mid \{\} \mid N)$. After that, the *main loop* of a prover is executed, where this clause set is processed according to deduction and simplification rules. This process stops if either all clauses are processed ($U = \{\}$), or the empty clause $\square$ is derived ($\square \in U$): see Figure 26.

Now suppose $\boldsymbol{\mathcal{S}}$ is an *inference system* consisting of deduction and (nondeterministic) simplification rules. Given an inference $\pi \in \boldsymbol{\mathcal{S}}$, we write $S \cup [\![S']\!] \vdash_\pi N$ if a clause set N is a possible conclusion of the inference π from $S \cup S'$, where all clauses from S' are deleted. This relation is extended to the full inference system

by writing $S \cup [\![S']\!] \vdash_{\mathcal{S}} N$ when N is the set of all clauses that are obtained in one step from clauses $S \cup S'$ and S' is the set of all clauses that were deleted in these inferences, i.e., formally:

$$S \cup [\![S']\!] \vdash_{\mathcal{S}} N \quad \textit{iff} \quad N = \bigcup_{\pi \in \mathcal{S}} N_\pi;\ S' = \bigcup_{\pi \in \mathcal{S}} S'_\pi;\ \textit{where}\ S \cup S' \setminus S'_\pi \cup [\![S'_\pi]\!] \vdash_\pi N_\pi. \quad (39)$$

In order to explain how deduction takes place, we need to introduce additional operation with theorem proving states, namely *insertion of clauses*. Given a theorem proving state $S = (D \mid O \mid U)$, the result of *insertion* of a clause set N *into* S is a theorem proving state defined by: $S \cup N = (D \mid O \mid U) \cup N := D \mid O \mid (U \cup N \setminus (D \cup O))$. It is obvious that insertion preserves disjointness for the sets of deleted, worked-out and usable clauses. Now the transformation $\mathsf{Deduce}(S)$ performing deduction of new clauses in the *main loop* of a prover is defined in Figure 27 According to this definition, the next theorem proving state is

Figure 27 Deduction of new clauses

Deduce

$$D \mid O \sqcup O' \mid \{C\} \sqcup U \ \Rightarrow\ (D \cup O' \mid O \cup \{C\} \mid U) \cup N \quad \text{where} \quad O \cup \{C\} \cup [\![O']\!] \vdash_{\mathcal{S}} N$$

$$D \mid O \sqcup O' \mid \{C\} \sqcup U \ \Rightarrow\ (D \cup O' \cup \{C\} \mid O \mid U) \cup N \quad \text{where} \quad O \cup [\![\{C\} \cup O']\!] \vdash_{\mathcal{S}} N$$

obtained from the previous state by (**1**) selecting a usable clause, (**2**) inserting conclusions of all inferences between this clause and worked-out clauses, (**3**) moving deleted clauses into D and (**4**) moving the selected clause into the set of worked-out clauses O if it has not been deleted. Note that the Deduce transformation might be nondeterministic, since the relation $\vdash_{\mathcal{S}}$ is not necessary functional because of nondeterministic simplification rules. However, if all simplification rules are deterministic, then the saturation procedure is also deterministic.

In our model of saturation, deleted clauses are not removed from a clause set, but moved into a special set D of *deleted clauses* (speaking in deferent terms, these clauses are *marked as deleted*). This is done to implement a so-called *permanent deletion*. Using permanent deletion, clauses that were once deleted, are memorised in a theorem proving state which prevents it from deriving them again. This helps avoiding (infinite) repetitions of deriving/deletion of the same clauses and solves certain problems with *fairness* of a saturation process. However, permanent deletion requires more memory.[19] For certain inference rules, like Tautology Deletion or Elimination of Duplicate Literals it is possible to completely remove deleted premises, when these inference rules are applied *eagerly*. *Eager application of* a simplification rule means that the rule is applied to a clause set as soon as it has

[19]This is a reason why most of theorem provers do not use permanent deletion and solve problems with fairness by different means

become applicable. In this case, all clauses are kept *normalised* w.r.t. these rules. For example, eager application of Elimination of Duplicate Literals, amounts to deletion of all duplicate literals in clauses that are produced. The only requirement for eager application, is that this normalisation process must always terminate. It is possible to show that, in fact, all simplification rules from Figure 25 can be used *eagerly*, i.e., saturation of every clause set w.r.t. these inference rules always terminates (indeed, conclusions of these inference rules have a strictly smaller literals than their premises).

In Figure 28, we modified the computational model from Figure 26 to make use of eager simplification rules. For this, we have defined an additional transformation Simplify(S) for states of a prover. This transformation performs all one-step

Figure 28 A model of computation with eager simplification rules

```
Prover (N)
  S:=Normalise ({} | {} | N);
  while (U ≠ {}) and (□ ∉ U) do
     S:=Normalise (Deduce (S));
  end;
  if (U = {}) then return ("Satisfiable");
  if (□ ∈ U) then return ("Unsatisfiable");
end.
```

```
Normalise (S)
  repeat
     S' := S;
     S := Simplify (S');
  until (S = S');
  return(S);
end.
```

Simplify

$$D \mid O \sqcup O' \mid U \sqcup U' \;\Rightarrow\; (D \cup O' \mid O \mid U) \cup N \quad \text{where} \quad O \cup U \cup [\![O' \cup U']\!] \vdash_{\mathcal{S}_e} N$$

inferences from the set of worked-out and usable clauses w.r.t. the set $\mathcal{S}_e$ of *eager simplification rules* of an inference system. If a *worked-out* clause is deleted during any of these inferences, than it is moved into the set D of deleted clauses. On the other hand, all deleted *usable* clauses are simply removed from a clause set. This transformation is iteratively repeated until nothing new is derived or deleted. The described procedure is implemented using function Normalise(S), which is applied in a state of a prover after each deduction step.

7.2.1 Correctness of the theorem-proving procedures

Before using the procedures given in Figure 26 and Figure 28, we have to demonstrate their correctness. That is, we need to show their (**1**) *soundness*: for every satisfiable input set N there is a branch where a prover does *not* return "Unsatisfiable", and (**2**) *completeness*: for every unsatisfiable input set N a prover must return "Unsatisfiable".

Soundness for a theorem prover follows from soundness of inference rules because the Deduce and the Simplify transformations can be always applied in a

satisfiability-preserving way. For showing completeness, we need to make additional assumptions about the data structure used to operate with sets of clauses (to ensure, in particular, that the results of different operations can be always computed). We assume that there is an efficient mechanism that provides for:

1. **Insertion of clauses:** *Given a theorem proving state S, the result of insertion of a clause set N into S can be effectively computed.*
2. **Enumeration of clauses:** *For every state $S = (D \mid O \mid U)$, the sets O and U of worked-out and usable clauses can be effectively enumerated.*
3. **Fair selection of usable clauses:** *There is a function Choose$(\cdot)$ that given a theorem proving state $S = (D \mid O \mid U)$ selects a usable clause Choose$(S) \in U$ such that for every sequence of theorem proving states $S_i = (D_i \mid O_i \mid U_i)$, $i = 1, 2, .., n, ..$, with Choose$(S_i) \notin U_j$ when $1 \leq i < j$, we have $\bigcup_{i \geq 1} \bigcap_{j \geq i} U_j = \{\}$. In other words, if selected clauses are permanently removed from the set of usable clauses, then every usable clause will be eventually removed.*

The first property can be achieved by keeping the set of all clauses sorted, and checking whether a clause is already in this set during insertion. In this way, insertion of each clause C in a state S can be done in time $O(|C|)$. For the second property, one can use links to the "next" and "previous" elements in O and U. The third property can be implemented in various ways, for instance, by assigning to every new clause a positive weight (say, the number of symbols in a clause) and selecting a minimal new clause according to this weight [see Weidenbach, 2001].

Properties 1 and 2 above insure that one can always compute the results of transformations Deduce and Simplify. It is easy to see that every state S computed in saturation process contains all inferences from the set O of worked-out clauses. Since deletion of clauses is done according to a redundancy criterion, all clauses from D must be *redundant* w.r.t. S.

Note that rules Deduce and Simplify are instances of *theorem proving derivations* given in Figure 5. Hence, the *limit* of every sequence $S_1 \setminus D_1,\ S_2 \setminus D_2,\ \cdots$ of non-deleted clauses from theorem proving states, is saturated up to redundancy provided that all derivations were performed fairly (see Definition 3.24 on p. 47). In this case, completeness of our theorem prover follows from Completeness Criterion (see Corollary 3.28).

Fairness of derivations is guaranteed by assumption 3 above. Indeed, if some inference π can be applied to all $S_i \setminus D_i$ starting from some $j \geq 1$, then by condition 3, there exists $k \geq j$ such that O_k contains all premises of π. Hence the conclusion of π must be in S_k, and so π is redundant w.r.t. $S_k \setminus D_k$.

7.2.2 Complexity of saturation procedures

In many saturation-based decision procedures it is possible to estimate the maximal number of different clauses that might be produced by a saturation procedure. This gives a bound on space complexity of decision procedures. We will use the model of theorem prover given in Figure 28 to estimate the *time* required to compute a saturation of bounded size.

We assume that the conclusion of every inference can be computed in polynomial time in the size of its premises. This assumption holds for all calculi introduced in this chapter. Moreover, we assume that *normalisation* of a clause set N w.r.t. eager simplification rules *can be done in polynomial time* in the size of N. For many simplification rules and clause classes this is indeed the case. It is possible to show that normalisation w.r.t. Tautology Deletion, Elimination of Duplicate Literals and Splitting can be done in polynomial time. Checking subsumption is in general an NP-complete problem [see Garey & Johnson, 1979], however for clauses with bounded number of literals containing all variables of a clause, Subsumption Deletion can be done in polynomial time [Gottlob & Leitsch, 1985; Ganzinger & de Nivelle, 1999]. In other cases, this rule can be always replaced with a suitable polynomial approximation [see Weidenbach, 2001].

We give estimation for time complexity of saturation procedures in terms of:

- $|N|$ - *the size of the initial clause set;*
- $\boldsymbol{c}$ - *the maximal number of normalised clauses;*
- $\boldsymbol{s}$ - *the maximal number of clauses in a normalised clause set;*
- $\boldsymbol{m}$ - *the maximal size of a normalised clause;*
- $\boldsymbol{k}$ - *the maximal number of premises in all deduction and simplification rules.*

Note that $\boldsymbol{s}$ bounds the number of worked-out and usable clauses after each normalisation steps, whereas $\boldsymbol{c}$ bounds the total number of clauses (including deleted ones) after normalisation. The value $\boldsymbol{s}$ might be strictly smaller than $\boldsymbol{c}$, if there are eager simplification rules with at least two premises. For example, if we use Subsumption Deletion eagerly, then $\boldsymbol{s}$ bounds the maximal number of clauses in a set where no clause strictly subsumes other clauses.

Now we estimate the time required to compute a saturation according to the procedure from Figure 28. Normalisation of the input clause set can be done it time $p(|N|)$ (where $p(\cdot)$ is a polynomial function). In order to estimate the time required for the *main loop*, note that after each iteration of this loop, the set $D \cup O$ is incremented exactly on one clause, namely the usable clause which was selected in the Deduce transformation. Since the total number of clauses in S after normalisation step is bounded by $\boldsymbol{c}$, we may have at most $\boldsymbol{c}$ iterations of the main loop. In each iteration, according to Figure 27, one should compute all possible

inferences between worked-out clauses and the selected clause. The number of such inferences is bounded by $O(\boldsymbol{s}^{(\boldsymbol{k}-1)})$ (recall that the set of worked-out clauses is normalised before this step, hence their total number is bounded by $\boldsymbol{s}$). Since every inference can be done in polynomial time in $\boldsymbol{m}$, the computational cost for the Deduce step is $O(\boldsymbol{s}^{(\boldsymbol{k}-1)} \cdot p(\boldsymbol{m}))$, which is also a bound on the size on new clauses that are derived in this step. The subsequent normalisation step can be done in polynomial time in this value. Hence the total running time for the saturation procedure is bounded by:

$$\boxed{\boldsymbol{t} \;=\; p(|N|) + \boldsymbol{c} \cdot p(\boldsymbol{m} \cdot \boldsymbol{s}^{(\boldsymbol{k}-1)})} \tag{40}$$

The estimation (40) for the time required to compute a saturation, can possibly be refined further, by considering a more detailed model of a theorem prover like say in [Weidenbach, 2001]. However, the main message of this section is that the *time complexity for saturation procedures is polynomial in its space complexity* (under the given assumptions about complexity of inference rules). One should remember, however, that a saturation procedure can be nondeterministic, hence although non-deterministic space complexity classes coincide with correspondent non-deterministic classes (for PSPACE or higher), this is not true for time complexity classes.

A Technical Appendixes

A.1 Lifting and Redundancy with Selection Functions

In this appendix we describe a completeness proof for the ordered resolution calculus with selection functions $\mathcal{OR}^{\succ}_{Sel}$ defined in subsection 3.6 in System 7. We also give formal definitions of redundancy criteria for non-ground calculi.

A.1.1 Abstract calculi and approximations

Before introducing a technique for proving completeness of $\mathcal{OR}^{\succ}_{Sel}$, we consider the lifting procedure described for $\mathcal{OR}^{\succ}$ in section 3 from a more abstract viewpoint.

Definition A.1 (Calculus).
An *(abstract) calculus* is a quadruple $\boldsymbol{\mathcal{C}} = (\boldsymbol{\mathcal{L}}, \boldsymbol{P}, \boldsymbol{\mathcal{S}}_p, \mathbf{R}_p)$, where:

1. $\boldsymbol{\mathcal{L}} = (\boldsymbol{D}, \Box_{\boldsymbol{\mathcal{L}}}, \boldsymbol{T}_{\boldsymbol{\mathcal{L}}})$ is a *logic*, where $\boldsymbol{D}$ is a nonempty set called the *domain*; $\Box_{\boldsymbol{\mathcal{L}}} \in \boldsymbol{D}$ is a *basic contradiction*, and $\boldsymbol{T}_{\boldsymbol{\mathcal{L}}} \subseteq 2^{\boldsymbol{D}}$ is a collection of *satisfiable sets* that enjoys the following properties: ($\boldsymbol{i}$) $D_1 \subseteq D_2 \in \boldsymbol{T}_{\boldsymbol{\mathcal{L}}}$ implies $D_1 \in \boldsymbol{T}_{\boldsymbol{\mathcal{L}}}$ and ($\boldsymbol{ii}$) $\{\Box_{\boldsymbol{\mathcal{L}}}\} \notin \boldsymbol{T}_{\boldsymbol{\mathcal{L}}}$;

2. $\boldsymbol{P}$ is a non-empty set of *admissible parameters* of the calculus and for each $p \in \boldsymbol{P}$:

3. $\boldsymbol{\mathcal{S}}_p \subseteq \mathrm{Inf}_{\boldsymbol{D}} := \{(d_1,..,d_k \vdash d) \mid d_i, d \in \boldsymbol{D}, 1 \leq i \leq k \geq 0\}$ is an *inference system* and

4. $\mathbf{R}_p = (\mathsf{R}^p_{\mathrm{Cl}}(\cdot), \mathsf{R}^p_{\mathrm{Inf}}(\cdot))$ is a *redundancy criterion*, where for every $D \subseteq \boldsymbol{D}$ we have $\mathsf{R}^p_{\mathrm{Cl}}(D) \subseteq \boldsymbol{D}$ is the set of *redundant clauses w.r.t.* D and $\mathsf{R}^p_{\mathrm{Inf}}(D) \subseteq \mathrm{Inf}_{\boldsymbol{D}}$ is the set of *redundant inferences w.r.t.* D, are such that the analogs of conditions in Definition 3.20 hold:

   for every $D \setminus D' \subseteq \mathsf{R}_{\mathrm{Cl}}(D)$, (**R1**) $D \notin \boldsymbol{T}_{\boldsymbol{\mathcal{L}}}$ implies $D' \notin \boldsymbol{T}_{\boldsymbol{\mathcal{L}}}$, (**R2**) $\mathsf{R}_{\mathrm{Cl}}(D) \subseteq \mathsf{R}_{\mathrm{Cl}}(D')$ and (**R3**) $\mathsf{R}_{\mathrm{Inf}}(D) \subseteq \mathsf{R}_{\mathrm{Inf}}(D')$. $\mathbf{R}$ is *effective* if in addition (**R4**) $\pi \in \mathsf{R}_{\mathrm{Inf}}(D)$ for every $\pi \in \mathrm{Inf}_{\boldsymbol{D}}$ with $\pi(d) \subseteq D$.

The basic notions for inference systems and redundancy criteria from the definitions 2.39, 2.40, 3.24 and 3.26 are easily modified for abstract calculi. A clause set N is called $\boldsymbol{\mathcal{C}}$*-saturated* if N is $(\boldsymbol{\mathcal{S}}_p, \mathbf{R}_p)$*-saturated for some* $p \in \boldsymbol{P}$. The calculus $\boldsymbol{\mathcal{C}}$ is *(refutationally) complete* if for every $p \in \boldsymbol{P}$, $(\boldsymbol{\mathcal{S}}_p, \mathbf{R}_p)$ *is complete* for $\boldsymbol{D}$. ⋘

For example, the (ground version) of the *ordered resolution calculus with selection* $\mathcal{OR}^{0\succ}_{Sel}$ is a calculus based on the ground clause logic, i.e., $\boldsymbol{\mathcal{L}} := (\mathrm{Cl}^0_\Sigma, \Box, \{N \subseteq \mathrm{Cl}^0_\Sigma \mid N \textit{is satisfiable}\})$. For simplicity, we identify the set of (ground) clauses with

the correspondent logic (i.e., here $\boldsymbol{\mathcal{L}} = \mathrm{Cl}^0_\Sigma$). $\mathcal{OR}^{0\succ}_{Sel}$ is parametrized by the set $\boldsymbol{P} := \{(\succ, Sel) \mid \text{“}\succ\text{” } \textit{is admissible}\}$ of admissible orderings and selection functions. For every choice of a parameter $p = (\succ, Sel) \in \boldsymbol{P}$, the inference system $\boldsymbol{\mathcal{S}}_p := \mathcal{OR}^{0\succ}_{Sel}$ is defined in System 5 and $\mathbf{R}_p := \mathsf{R}^{\mathcal{S}\succ}_{\mathrm{gr}}$ is the standard redundancy criterion for ground clauses. The calculus $\mathcal{OR}^{0\succ}_{Sel}$ is complete because, as it was shown in Corollary 3.30, $(\boldsymbol{\mathcal{S}}_p, \mathbf{R}_p)$ is complete for any choice of admissible parameters. Similarly, $\mathcal{OR}^{\succ}$ is a calculus for the logic $\boldsymbol{\mathcal{L}} := \mathrm{Cl}_\Sigma$ and is parametrized by $\boldsymbol{P} := \{\succ \mid \text{“}\succ\text{” } \textit{is admissible}\}$. It is complete by Theorem 3.39.

The completeness criterion formulated in Corollary 3.28 can be extended without considerable modifications to abstract calculi as follows:

Lemma A.2 (Completeness Criterion for Calculi). *Let $\boldsymbol{\mathcal{C}} = (\boldsymbol{\mathcal{L}}, \boldsymbol{P}, \boldsymbol{\mathcal{S}}_p, \mathbf{R}_p)$ be a calculus. Then $\boldsymbol{\mathcal{C}}$ is complete iff for every $\boldsymbol{\mathcal{C}}$-saturated set $D \subseteq \boldsymbol{D}$, if $D \notin \boldsymbol{T}_{\mathcal{L}}$ then $\Box_{\mathcal{L}} \in D$.*

This criterion motivates the following definition, using which it is possible to transfer completeness of one calculus to completeness of the other calculus:

Definition A.3 (Approximates). Given calculi $\boldsymbol{\mathcal{C}} = (\boldsymbol{\mathcal{L}} = (\boldsymbol{D}, \Box_{\mathcal{L}}, \boldsymbol{T}_{\mathcal{L}}), \boldsymbol{P}, \boldsymbol{\mathcal{S}}_p, \mathbf{R}_p)$ and $\boldsymbol{\mathcal{C}}' = (\boldsymbol{\mathcal{L}}' = (\boldsymbol{D}', \Box_{\mathcal{L}'}, \boldsymbol{T}_{\mathcal{L}'}), \boldsymbol{P}', \boldsymbol{\mathcal{S}}'_p, \mathbf{R}'_p)$, we say that $\boldsymbol{\mathcal{C}}$ *approximates* $\boldsymbol{\mathcal{C}}'$ if there exists a function $\alpha : 2^{\boldsymbol{D}} \mapsto 2^{\boldsymbol{D}'}$ such that for every $D \subseteq \boldsymbol{D}$: (***i***) D is $\boldsymbol{\mathcal{C}}$-saturated then $\alpha(D)$ is $\boldsymbol{\mathcal{C}}'$-saturated, (***ii***) $D \notin \boldsymbol{T}_{\mathcal{L}}$ implies $\alpha(D) \notin \boldsymbol{T}_{\mathcal{L}'}$ and (***iii***) $\Box_{\mathcal{L}} \notin D$ implies $\Box_{\mathcal{L}'} \notin \alpha(D)$. ⋘

Definition A.3 is derived from the concept of *approximation between theorem proving systems* that has been introduced in [Bachmair et al., 1994] to reduce refutational completeness from one system to the other. In Theorem 3.39 we have shown that the calculus $\mathcal{OR}^{\succ}$ approximates its ground version $\mathcal{OR}^{0\succ}$ via the mapping $\mathrm{gr} : N \mapsto N^{\mathrm{gr}}$. This in the end made it possible to transfer completeness of $\mathcal{OR}^{0\succ}$ to completeness of $\mathcal{OR}^{\succ}$. The arguments used in this proof can be generalized to the following

Lemma A.4 (Relative Completeness). *Let $\boldsymbol{\mathcal{C}} = (\boldsymbol{\mathcal{L}}, \boldsymbol{P}, \boldsymbol{\mathcal{S}}_p, \mathbf{R}_p)$ be a calculus that approximates a calculus $\boldsymbol{\mathcal{C}}' = (\boldsymbol{\mathcal{L}}', \boldsymbol{P}', \boldsymbol{\mathcal{S}}'_p, \mathbf{R}'_p)$ via α. Then $\boldsymbol{\mathcal{C}}$ is complete if $\boldsymbol{\mathcal{C}}'$ is complete.*

Proof. Assume that $\boldsymbol{\mathcal{C}}'$ is complete but $\boldsymbol{\mathcal{C}}$ is not. By the completeness criterion for calculi (Theorem A.2), the last means that there exists a $\boldsymbol{\mathcal{C}}$-saturated set $D \subseteq \boldsymbol{D}$ such that $D \notin \boldsymbol{T}_{\mathcal{L}}$ and $\Box_{\mathcal{L}} \notin D$. Since $\boldsymbol{\mathcal{C}}$ approximates $\boldsymbol{\mathcal{C}}'$ via α, we have (***i***) $\alpha(D)$ is $\boldsymbol{\mathcal{C}}'$-saturated, (***ii***) $\alpha(D) \notin \boldsymbol{T}_{\mathcal{L}'}$ and (***iii***) $\Box_{\mathcal{L}'} \notin \alpha(D)$. However by the completeness criterion for $\boldsymbol{\mathcal{C}}'$ from (***i***) and (***ii***) it follows that $\Box_{\mathcal{L}'} \in \alpha(D)$. A contradiction with (***iii***). ⧉

A.1.2 Ground closures

After these preparations, we demonstrate how to establish completeness of the *ordered resolution calculus with selection* $\mathcal{OR}^{\succ}_{Sel}$. For this purpose, we introduce an *intermediate calculus* $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ and show that $\mathcal{OR}^{\succ}_{Sel}$ *approximates* $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ which approximates $\mathcal{OR}^{0\succ}_{Sel}$. This would imply from Corollary 3.30 by Lemma A.4 that $\mathcal{OR}^{\succ}_{Sel}$ is complete.

The key concept of the intermediate calculus $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ is the notion of a ground closure.

Definition A.5 (Ground Closures). A *ground (clause) closure* is a pair $(C^0 \colon C)$, where $C \in \mathrm{Cl}_\Sigma$ is a clause and $C^0 = C\sigma \in \mathrm{Cl}^0_\Sigma$ is a ground instance of C. The set of all ground closures over a signature Σ is defined by GC_Σ.

Given a clause set $N \subseteq \mathrm{Cl}_\Sigma$, we define $N^{\mathrm{gc}} := \{(C^0 \colon C) \in \mathrm{GC}_\Sigma \mid C \in N\}$ to be the set of all ground closures for clauses in N. For a subset $M \subseteq \mathrm{GC}_\Sigma$, we denote by M^{gr} its *ground part*, i.e., $M^{\mathrm{gr}} := \{C^0 \mid (C^0 \colon C) \in N\}$. Note that $N^{\mathrm{gr}} = (N^{\mathrm{gc}})^{\mathrm{gr}}$ for all $N \in \mathrm{Cl}_\Sigma$.

A set $M \subseteq \mathrm{GC}_\Sigma$ of ground closures is *satisfiable* if M^{gr} is satisfiable. The *logic of ground closures* is given by $(\mathrm{GC}_\Sigma, (\Box \colon \Box), \{M \subseteq \mathrm{GC}_\Sigma \mid M \textit{is satisfiable}\})$ which we also denote by GC_Σ. ⋘

Along with the introduced notation for ground closures, we also write closures of the form $(C^0 \vee A^0 \colon C \vee A)$ and $(D^0 \vee \neg B^0 \colon D \vee \neg B)$ respectively as $(C^0 \colon C) \vee (A^0 \colon A)$ and $(D^0 \colon D) \vee \neg(B^0 \colon B)$. However note, that for two ground closures $(C^0 \colon C)$ and $(D^0 \colon D)$, the expression $(C^0 \vee D^0 \colon C \vee D)$ is not necessary a ground closure, if C and D have shared variables.

The intermediate calculus $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ is based on the logic of ground closures GC_Σ and is parametrized by an admissible ordering $\succ$ on *ground* literals and a selection function *Sel* on (full) clauses. The inference rules for any choice of these parameters are given in System 13. It can be easily shown (see the lifting diagram on Figure 7),

Ordered Resolution

$$\mathtt{OR}: \frac{(C^0 \colon C) \vee (\underline{\boldsymbol{A}^0 \colon \boldsymbol{A}})^\star \quad (D^0 \colon D) \vee \neg(\underline{\boldsymbol{A}^0 \colon \boldsymbol{B}})}{(C^0 \colon C\sigma) \vee (D^0 \colon D\sigma)}$$

where (i) $\sigma = \mathrm{mgu}(A, B)$; (ii) A^0 is strictly maximal w.r.t. C^0 and nothing is selected in $C \vee A$, and (iii) either $\neg B$ is selected in $D \vee \neg B$, or, otherwise nothing is selected in $D \vee \neg B$ and $\neg A^0$ is maximal w.r.t. D^0

Ordered Factoring

$$\mathtt{OF}: \frac{(C^0 \colon C) \vee (\underline{A}^0 \colon \underline{B}) \vee (\underline{\boldsymbol{A}^0 \colon \boldsymbol{A}})}{(C^0 \colon C\sigma) \vee (A^0 \colon A\sigma)}$$

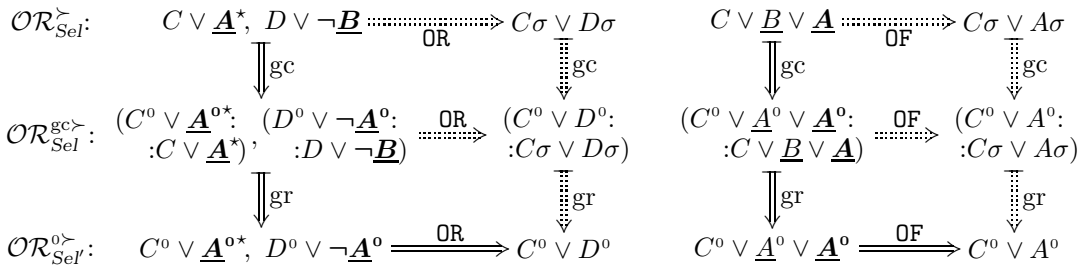
where (i) $\sigma = \mathrm{mgu}(A, B)$; (ii) A^0 is maximal w.r.t. $C^0 \vee B^0$ and nothing is selected in $C \vee A \vee B$.

System 13: The intermediate calculus $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ for $\mathcal{OR}^{\succ}_{Sel}$

that the set of ground closures is closed under inferences of this calculus. Moreover,

it is easy to see that for every inference $\pi' = [(C_1^0{:}\, C_1),..,(C_k^0{:}\, C_k) \vdash (C^0{:}\, C)] \in \mathcal{OR}_{Sel}^{\mathrm{gc}\succ}$ there exists a correspondent inference $\pi = (C_1,..,C_k \vdash C) \in \mathcal{OR}_{Sel}^{\succ}$. So the lifting diagram can be extended for $\mathcal{OR}_{Sel}^{\succ}$ and $\mathcal{OR}_{Sel}^{\mathrm{gc}\succ}$: See the upper part in Figure 29. We exploit this correspondence for defining the notion of redundancy

Figure 29 The lifting diagram for $\mathcal{OR}_{Sel}^{\succ}$



for calculi parametrized by selection functions.

Definition A.6 (Standard Redundancy with Selection Functions). Let $\succ$ be an ordering on ground literals that is admissible according to Definition 3.4. The ordering $\succ$ on ground clauses is extended to ground closures by setting $(C^0{:}\, C) \succ (D^0{:}\, D)$ *iff* $C^0 \succ D^0$. The logical entailment relation $\vDash$ is extended to the set of ground closures as follows: $M_1 \vDash M_2$ *iff* $M_1^{\mathrm{gr}} \vDash M_2^{\mathrm{gr}}$.

Let $\boldsymbol{\mathcal{S}}$ be an inference system on clauses and $\boldsymbol{\mathcal{S}'}$ be a correspondent (via a lifting diagram) inference systems on ground closures, both parametrized with the same admissible ordering $\succ$ and a selection function *Sel*. Given a set of ground closures M, we say that a *ground closure* $(C^0{:}\, C)$ *is redundant w.r.t.* M, if $M_{(C^0:\, C)} \vDash (C^0{:}\, C)$.[20] An *inference* $\pi' : (C_1^0{:}\, C_1),..,(C_k^0{:}\, C_k) \vdash (C^0{:}\, C) \in \mathrm{Inf}_{\mathrm{GC}_\Sigma}$ *is redundant w.r.t.* M *in* $\boldsymbol{\mathcal{S}'}$, if $\pi' \in \boldsymbol{\mathcal{S}'}$ implies that $M_{(C_i^0:\, C_i)} \vDash (C^0{:}\, C)$, for some i with $1 \leq i \leq k$.

Given a clause set N, we say that a *clause* C *is redundant w.r.t.* N if every clause in $\{C\}^{\mathrm{gc}}$ is redundant w.r.t. N^{gc}. An *inference* $\pi = C_1,..,C_k \vdash C \in \mathrm{Inf}_\Sigma$ *is redundant w.r.t.* N *in* $\boldsymbol{\mathcal{S}}$, if every *ground closure* $(\pi^0{:}\, \pi) := (C_1^0{:}\, C_1),..,(C_k^0{:}\, C_k) \vdash (C^0{:}\, C) \in \mathrm{Inf}_{\mathrm{GC}}$ of π is redundant w.r.t. N^{gc} in $\boldsymbol{\mathcal{S}'}$. ⋘

To summarise this definition in a practically usable way, a clause C is redundant w.r.t. N if every closure $(C^0{:}\, C)$ of this clause is redundant w.r.t. the set of ground closures N^{gc} of N, which is when the clause C^0 follows from the set $((N^{\mathrm{gc}})_{(C^0:\, C)})^{\mathrm{gr}} = (N^{\mathrm{gr}})_{C^0}$ of the smaller ground instances from N^{gr}. An inference $\pi \in \boldsymbol{\mathcal{S}}$ is redundant if every ground closure $(\pi^0{:}\, \pi)$ of this inference is redundant

[20] As usual, for a set $D \subseteq \boldsymbol{D}$, ordered by $\succ$ and $d \in D$, we denote $D_d := \{d' \in D \mid d' \prec d\}$

w.r.t. N^{gc}, which is when either $(\pi^0\colon\pi) \notin \boldsymbol{\mathcal{S}}'$ (this is only possible if π^0 does not satisfy the *ordering restrictions for non-selected literals* in π), or the conclusion C^0 of π^0 follows from the set $((N^{\mathrm{gc}})_{(C_i^0:C_i)})^{\mathrm{gr}} = (N^{\mathrm{gr}})_{C_i^0}$ of smaller clauses for some premise C_i^0 of π^0.

Note that the notions of redundant clauses and redundant inferences given in Definition A.6, coincide with those given in Definition 3.36, when the selection function is trivial: $Sel = Sel_0$. So, ground closures are essentially needed for defining redundancy of inferences having selected literals. In the next section, we show how ground closures allow to refine the notion of redundancy even further.

Lemma A.7 (Standard Redundancy Criterion for Ground Closures). *The notion of standard redundancy* $\mathsf{R}_{\mathrm{gc}}^{\boldsymbol{\mathcal{S}}'\succ} = (\mathsf{R}_{\mathrm{GC}}^{\succ}(\cdot), \mathsf{R}_{\mathrm{Inf}_{\mathrm{GC}}}^{\boldsymbol{\mathcal{S}}'\succ}(\cdot))$ *for ground closures given in Definition A.6 is a redundancy criterion for every inference system* $\boldsymbol{\mathcal{S}}'$ *on ground closures and a well-founded ordering* $\succ$ *on ground clauses. If in addition* $\boldsymbol{\mathcal{S}}'$ *is monotone w.r.t.* $\succ$*, then* $\mathsf{R}_{\mathrm{gc}}^{\boldsymbol{\mathcal{S}}'\succ}$ *is effective.*

Proof. The proof of this lemma is identical to the proof of Lemma 3.21, but for ground closures instead of ground clauses. ⧉

Corollary A.8 (Standard Redundancy Criterion with Selection Functions). *The extension of the notion of standard redundancy* $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ} = (\mathsf{R}_{\mathrm{Cl}}^{\succ}(\cdot), \mathsf{R}_{\mathrm{Inf}}^{\boldsymbol{\mathcal{S}}\succ}(\cdot))$ *for inference systems with selection functions given in Definition A.6 a redundancy criterion. In addition, if the correspondent inference system* $\boldsymbol{\mathcal{S}}'$ *on ground closures is monotone, then* $\mathsf{R}^{\boldsymbol{\mathcal{S}}\succ}$ *is effective.*

Proof. Similarly as in the proof of Lemma 3.37, this corollary follows from the fact that for every clause sets N' and N, we have $N \subseteq \mathsf{R}_{\mathrm{Cl}}^{\succ}(N)$ *iff* $N^{\mathrm{gc}} \subseteq \mathsf{R}_{\mathrm{GC}}^{\succ}(N^{\mathrm{gc}})$, and for every set of inferences $S \subseteq \mathrm{Inf}_{\Sigma}$, we have $S \subseteq \mathsf{R}_{\mathrm{Inf}}^{\boldsymbol{\mathcal{S}}\succ}(N)$ *iff* $S^{\mathrm{gc}} \subseteq \mathsf{R}_{\mathrm{Inf}_{\mathrm{GC}}}^{\boldsymbol{\mathcal{S}}'\succ}(N^{\mathrm{gc}})$, where $S^{\mathrm{gc}} \subseteq \mathrm{Inf}_{\mathrm{GC}_{\Sigma}}$ is the set of all ground closures of inferences in S. ⧉

Lemma A.9. $\mathcal{OR}_{Sel}^{\succ}$ *approximates* $\mathcal{OR}_{Sel}^{\mathrm{gc}\succ}$.

Proof. We show that the mapping $\mathrm{gc} : N \mapsto N^{\mathrm{gc}}$, $N \subseteq \mathrm{Cl}_{\Sigma}$ is an approximation function for $\mathcal{OR}_{Sel}^{\succ}$ and $\mathcal{OR}_{Sel}^{\mathrm{gc}\succ}$. First note that conditions (***ii***) and (***iii***) for approximations (see Definition A.3) are trivial by definition of the logic for ground closures (see Definition A.5). In order to demonstrate (***i***), let N be a clause set that is saturated in $\mathcal{OR}_{Sel}^{\succ}$ under some ordering $\succ$ and selection function *Sel*. We show that the set N^{gc} is saturated in $\mathcal{OR}_{Sel}^{\mathrm{gc}\succ}$ under the same parameters. Indeed, for every inference $\pi' = [(C_1^0\colon C_1),..,(C_k^0\colon C_k) \vdash (C^0\colon C)] \in \mathcal{OR}_{Sel}^{\mathrm{gc}\succ}$ there exists a correspondent inference $\pi = (C_1,..,C_k \vdash C) \in \mathcal{OR}_{Sel}^{\succ}$. Moreover, by Definition A.6, if π is redundant w.r.t. N then π' is redundant w.r.t. N^{gc}. Hence the set N^{gc} is saturated in $\mathcal{OR}_{Sel}^{\mathrm{gc}\succ}$. ⧉

Lemma A.10. $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ *approximates* $\mathcal{OR}^{0\succ}_{Sel}$.

Proof. We show that the mapping $\mathrm{gr} : M \mapsto M^{\mathrm{gr}}$, $M \subseteq \mathrm{GC}_\Sigma$ is an approximation function for $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ and $\mathcal{OR}^{0\succ}_{Sel}$. Again, conditions (***ii***) and (***iii***) follow from the definition of the logic for ground closures. In order to demonstrate (***i***), let M be a saturated in $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ set under some ordering $\succ$ and selection function *Sel*. We show that M^{gr} is saturated in $\mathcal{OR}^{0\succ}_{Sel}$ under $\succ$ and a selection function *Sel*′ that is defined for *Sel* using the following notion:

Definition A.11 (Projection of a Selection Function). Given a clause set N we say that a selection function *Sel*′ *is a (ground) projection of* a selection function *Sel from* N, if for every clause $C^0 \in N^{\mathrm{gr}}$, there exists a clause $C \in N$ such that $Sel'(C^0) = Sel(C)\cdot\sigma$ for some substitution σ (substitution is applied to a multiset element-wise). ◈

Note that for every clause set N and every selection function *Sel* there exists at least one projection of *Sel* from N: one can set any well-order $\succ_N$ on N and select for every $C^0 \in N^{\mathrm{gr}}$ the respective selected literals in the minimal w.r.t. $\succ_N$ clause C such that $C^0 = C\sigma$.

Let *Sel*′ be a projection of the selection function *Sel* from $M^{cl} := \{C \in \mathrm{Cl}_\Sigma \mid (C^0 \colon C) \in M\}$. We claim that the set M^{gr} is saturated in $\mathcal{OR}^{0\succ}_{Sel}$ under $\succ$ and *Sel*′. Indeed, for every ground clause $C^0 \in M^{\mathrm{gr}}$ by Definition A.11 we can always find a clause $C \in \mathrm{Cl}_\Sigma$ such that $(C^0 \colon C) \in M$ and $Sel'(C^0) = Sel(C)\cdot\sigma$. Hence for every literal L in C and its correspondent literal L^0 in C^0, we have $L^0 \in Sel'(C^0)$ *iff* $L \in Sel(C)$. Now, for every inference $\pi^0 = (C^0_1,..,C^0_k \vdash C^0) \in \mathcal{OR}^{0\succ}_{Sel'}$ from M^{gr} there exists a correspondent inference $\pi' = [(C^0_1 \colon C_1),..,(C^0_k \colon C_k) \vdash (C^0 \colon C)] \in \mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ form M (see the lifting diagram in Figure 29). Moreover, π' is redundant *iff* π^0 is redundant, since redundancy of inferences on ground closures depends only on their ground components (see Definition A.6). Hence M^{gr} is $\mathcal{OR}^{0\succ}_{Sel}$-saturated. ⧉

Corollary A.12 (Completeness of $\mathcal{OR}^{\succ}_{Sel}$ with Redundancy). $\mathcal{OR}^{\succ}_{Sel}$ *is a refutationally complete calculus.*

Proof. From Lemma A.9 and Lemma A.10 we know that $\mathcal{OR}^{\succ}_{Sel}$ approximates $\mathcal{OR}^{0\succ}_{Sel}$. Since the calculus $\mathcal{OR}^{0\succ}_{Sel}$ is complete (Theorem 3.30), by relative completeness (Lemma A.4), the calculus $\mathcal{OR}^{\succ}_{Sel}$ is also complete. ⧉

Remark A.13. We stress that proving completeness for $\mathcal{OR}^{\succ}_{Sel}$ without redundancy (i.e., with trivial redundancy) is not a big deal and can be done directly without using ground closures. One can show that whenever a clause set N is saturated in $\mathcal{OR}^{\succ}_{Sel}$, then the set N^{gr} is saturated in $\mathcal{OR}^{0\succ}_{Sel'}$ for any projection *Sel*′ of *Sel* from N. Ground closures and intermediate calculus $\mathcal{OR}^{\mathrm{gc}\succ}_{Sel}$ are needed for extending the completeness result with standard redundancy criterion. ◈

A.2 Subsumption

The usage of clause closures can justify an extension of redundancy criterion with *subsumption deletion*. Note that redundancy criterion from Definition A.6 does not make use of the non-ground components of closures. We can refine an ordering on closures to take them into account.

Recall that the *instance ordering* $\gtrsim_i$ on expressions and clauses is a *quasi-ordering* defined for which $E_2 \gtrsim_i E_1$ ($C_2 \gtrsim_i C_1$) if there exists a substitution σ_1 such that $E_2 = E_1\sigma$ ($C_2 = C_1\sigma$), i.e., E_2 is an instance of E_2 (C_2 is an instance of C_1). The strict instance ordering $>_i$ is the antisymmetric part of $\gtrsim_i$, i.e., $C_2 >_i C_1$ *iff* $C_2 \gtrsim_i C_1$ but not $C_1 \gtrsim_i C_2$. It is easy to see that the order $>_i$ is well-founded on expressions and clauses: if $C_2 >_i C_1$ then either (***i***) $|C_2| > |C_1|$, or (***ii***) $|C_2| = |C_1|$, but $\sharp vars[C_2] < \sharp vars[C_1] < |C_1|$.

Let us define the order $\succ$ on ground closures to be the lexicographic combinations of $\succ$ on ground clauses and $>_i$ on general clauses. More precisely, we set

$$(C^0\colon C) \succ (D^0\colon D) \quad \textit{iff} \quad (\boldsymbol{i})\ C^0 \succ D^0, \quad \text{or} \quad (\boldsymbol{ii})\ C^0 = D^0 \text{ and } C >_i D. \tag{41}$$

Note that if $\succ$ is a total and/or well-founded on ground clauses then $\succ$ is respectively total and/or well-founded on ground closures since clauses that are instances of each other should be variants of each other (and we do not distinguish those).

Lemma A.7 and its Corollary A.8 can be reproven when the definition of redundancy (Definition A.6) is modified for ordering (41). This extended notion of redundancy allows one to justify deletion of strictly subsumed clauses.

Recall from Definition 3.43 that a clause C *subsumes* a clause D if $C \lesssim_i D' \subseteq D$ (for some subclause D' of D), and C *strictly subsumes* D if C subsumes D but not the other way round.

Proposition A.14. *Let D be strictly subsumed by C. Then D is redundant w.r.t. $\{C\}$.*

Proof. By Definition 3.43, there exists a substitution σ such that $C\sigma \subseteq D$. since the subsumption between C and D is strict, then either (***i***) this inclusion is strict or (***ii***) σ is not a renaming.

Consider the sets $\{C\}^{\mathrm{gc}}$ and $\{D\}^{\mathrm{gc}}$ of ground closures of C and D. We must show that every ground closure $(D^0\colon D)$ of D follows from some smaller ground closures from $\{C\}^{\mathrm{gc}}$ w.r.t. to ordering (41). Since $D^0 = D\tau^0$ for some ground substitution τ^0, we must have $C^0 := (C\sigma)\tau^0 \subseteq D\tau^0 = D^0$. Hence, the ground closure $(D^0\colon D)$ is a logical consequence of the ground closure $(C^0\colon C) \in \{C\}^{\mathrm{gc}}$. It remains to show that $(D^0\colon D) \prec (C^0\colon C)$.

The last is trivial for case (***i***) above, since then $D^0 \subsetneq C^0$ and hence $D^0 \prec C^0$. If the inclusion is not strict, we have $D^0 = C^0$. Then since only case (***ii***) above

remains possible, we must have that $C >_i D$. This yields $D^0 \prec C^0$, what was required to show. ⧉

Clause closures, which extend ground clause closures, give rise to many other refinements of saturation-based procedures and new theorem proving techniques. Among them are the *basic strategies* [Bachmair, Ganzinger, Lynch & Snyder, 1995], reasoning with *constrained clauses* [Nieuwenhuis & Rubio, 2001] and *instance-based methods* [Ganzinger & Korovin, 2003]. These extensions are not used in this thesis, so we do not cover them here.

Bibliography

Baader, F. & Nipkow, T. [1998], *Term Rewriting and All That*, Cambridge University Press, United Kingdom. 15, 21, 22, 23, 25, 28

Baaz, M., Egly, U. & Leitsch, A. [2001], Normal form transformations, *in* Robinson & Voronkov [2001], chapter 5, pp. 273–333. 94, 95, 97

Bachmair, L. & Ganzinger, H. [1990], On restrictions of ordered paramodulation with simplification, *in* 'Proceedings of the tenth international conference on Automated deduction', Springer-Verlag New York, Inc., pp. 427–441. 3, 4, 34, 44, 59, 67, 71

Bachmair, L. & Ganzinger, H. [1994], 'Rewrite-based equational theorem proving with selection and simplification', *Journal of Logic and Computation* **4**(3), 217–247. 3, 4, 34, 44, 71

Bachmair, L. & Ganzinger, H. [1995], Ordered chaining calculi for first-order theories of binary relations, Technical Report MPI-I-95-2-009, Max-Planck-Institut für Informatik, Saarbrücken, Germany. Revised version to appear in JACM. 4, 72, 76, 78, 89

Bachmair, L. & Ganzinger, H. [1997], Strict basic superposition and chaining, Technical Report MPI-I-97-2-011, Max-Planck-Institut für Informatik, Saarbrücken, Germany. 69, 75, 92

Bachmair, L. & Ganzinger, H. [1998*a*], Equational reasoning in saturation-based theorem proving, *in* W. Bibel & P. Schmitt, eds, 'Automated Deduction — A Basis for Applications', Vol. I, Kluwer, chapter 11, pp. 353–397. 3, 59

Bachmair, L. & Ganzinger, H. [1998*b*], 'Ordered chaining calculi for first-order theories of transitive relations', *Journal of the ACM*. Revised Version of MPI-I-95-2-009. 4, 72, 76, 78, 79, 84, 85, 89

Bachmair, L. & Ganzinger, H. [2001], Resolution theorem proving, *in* Robinson & Voronkov [2001], chapter 2, pp. 19–99. 3, 49, 50, 105, 112

Bachmair, L., Ganzinger, H. & Waldmann, U. [1993], Superposition with simplification as a decision procedure for the monadic class with equality, *in* G. Gottlob, A. Leitsch & D. Mundici, eds, 'Computational Logic and Proof Theory, Third Kurt Gödel Colloquium, KGC'93', Vol. 713 of *Lecture Notes in Computer Science*, Springer, Brno, Czech Republic, pp. 83–96. 3

Bachmair, L., Ganzinger, H. & Waldmann, U. [1994], 'Refutational theorem proving for hierachic first-order theories', *Applicable Algebra in Engineering, Communication and Computing* **5**, 193–212. 49, 119

Bachmair, L., Ganzinger, H., Lynch, C. & Snyder, W. [1995], 'Basic paramodulation', *Information and Computation* **121**(2), 172–192. Revised version of TR MPI-I-93-236, 1993. 125

Brand, D. [1975], 'Proving theorems with the modification method', *SIAM J. Comput.* **4**(4), 412–430. 59

de Nivelle, H. [1998*a*], *Description of Bliksem 1.00.* (A one page description of Bliksem 1.00 for CASC-15). 60

de Nivelle, H. [1998*b*], A resolution decision procedure for the guarded fragment, *in* C. Kirchner & H. Kirchner, eds, 'Proceedings of the 15th International Conference on Automated Deduction (CADE-14)', Vol. 1421 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Lindau, Germany, pp. 191–204. 29

de Nivelle, H. [2000], 'Deciding the E-plus class by an a posteriori, liftable order', *Annals of Pure and Applied Logic* **88**(1), 219–232. 3

de Nivelle, H. [2001], Splitting through new proposition symbols., *in* R. Nieuwenhuis & A. Voronkov, eds, 'LPAR', Vol. 2250 of *Lecture Notes in Computer Science*, Springer, pp. 172–185. 111

de Nivelle, H. [2003], Translation of resolution proofs into short first-order proofs without choice axioms., *in* F. Baader, ed., 'CADE', Vol. 2741 of *Lecture Notes in Computer Science*, Springer, pp. 365–379. 104

de Nivelle, H. & de Rijke, M. [2003], 'Deciding the guarded fragments by resolution', *Journal of Symbolic Computation* **35**, 21–58. 3, 29, 58

de Nivelle, H. & Pratt-Hartmann, I. [2001], A resolution-based decision procedure for the two-variable fragment with equality., *in* T. N. R. Goré, A. Leitsch, ed., 'In: Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR-2001)', Vol. 2083 of *Lect. Notes Artif. Intell.*, Springer, Berlin, pp. 211–225. 3

Degtyarev, A. & Voronkov, A. [2001], Equality reasoning in sequent-based calculi, *in* Robinson & Voronkov [2001], chapter 10, pp. 611–706. 59

Dershowitz, N. [1987], 'Termination of rewriting', *Journal of Symbolic Computation* **3**(1&2), 69–115. Corrigendum: *4*, 3 (December 1987), 409–410. 15, 23

Dershowitz, N. & Plaisted, D. A. [2001], Rewriting, *in* Robinson & Voronkov [2001], chapter 9, pp. 535–610. 15

Fermüller, C., Leitsch, A., Tammet, T. & Zamov, N. [1993], *Resolution Methods for the Decision Problem*, Vol. 679 of *LNAI*, Springer, Berlin, Heidelberg. 3, 28, 29, 42

Fitting, M. [1996], *First-Order Logic and Automated Theorem Proving. Second Edition*, Springer. 5, 11

Ganzinger, H. & de Nivelle, H. [1999], A superposition decision procedure for the guarded fragment with equality, *in* 'Proc. 14th IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 295–305. 3, 116

Ganzinger, H. & Korovin, K. [2003], New directions in instantiation-based theorem proving, *in* 'Proc. 18th IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 55–64. 125

Ganzinger, H., Hustadt, U., Meyer, C. & Schmidt, R. A. [2001], A resolution-based decision procedure for extensions of K4, *in* M. Zakharyaschev, K. Segerberg, M. de Rijke & H. Wansing, eds, 'Advances in Modal Logic, Volume 2', Vol. 119 of *CSLI Lecture Notes*, CSLI, Stanford, USA, chapter 9, pp. 225–246. 3, 74

Ganzinger, H., Nieuwenhuis, R. & Nivela, P. [2002], 'The saturate system'. Software and documentation available at: `http://www.mpi-sb.mpg.de/SATURATE/Saturate.html`. 55

Garey, M. R. & Johnson, D. S. [1979], *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co. 116

Gottlob, G. & Leitsch, A. [1985], 'On the efficiency of subsumption algorithms', *J. ACM* **32**(2), 280–295. 116

Hsiang, J. & Rusinowitch, M. [1991], 'Proving refutational completeness of theorem-proving strategies: The transfinite semantic tree method', *J. ACM* **38**(3), 559–587. 59

Hustadt, U. [1999], Resolution-Based Decision Procedures for Subclasses of First-Order Logic, PhD thesis, Universität des Saarlandes, Saarbrücken, Germany. 3

Hustadt, U. & Schmidt, R. A. [1999], Maslov's class K revisited, *in* H. Ganzinger, ed., 'Automated Deduction—CADE-16', Vol. 1632 of *Lecture Notes in Artificial Intelligence*, Springer, pp. 172–186. 3

Hustadt, U., Motik, B. & Sattler, U. [2004], A decomposition rule for decision procedures by resolution-based calculi., *in* F. Baader & A. Voronkov, eds, 'LPAR', Vol. 3452 of *Lecture Notes in Computer Science*, Springer, pp. 21–35. 3

Joyner Jr., W. H. [1976], 'Resolution strategies as decision procedures', *Journal of the ACM* **23**(3), 398–417. 3

Kamin, S. & Lévy, J.-J. [1980], Two generalizations of the recursive path ordering, University of Illinois at Urbana-Chapaign. Unpublished manuscript. 21

Kazakov, Y. [2005], Saturation-Based Decision Procedures for Extensions of the Guarded Fragment, PhD thesis, Universität des Saarlandes, Saarbrücken, Germany. in preparation. 4, 55, 60

Knuth, D. E. & Bendix, P. B. [1970], Simple word problems in universal algebras., *in* J. Leech, ed., 'Computational Problems in Abstract Algebra', Pergamon Press, Oxford, U. K., pp. 263–297. 16, 17, 21

Kowalski, R. & Hayes, P. J. [1968], Semantic trees in automatic theorem-proving, *in* B. Meltzer & D. Michie, eds, 'Machine Intelligence 4', Edinburgh U. Press, Edinburgh, Scotland, pp. 87–101. 42, 47

Nieuwenhuis, R. & Rubio, A. [2001], Paramodulation-based theorem proving, *in* Robinson & Voronkov [2001], chapter 7, pp. 371–443. 3, 49, 53, 55, 59, 125

Nonnengart, A. & Weidenbach, C. [2001], Computing small clause normal forms, *in* Robinson & Voronkov [2001], chapter 6, pp. 335–367. 94, 96, 97, 105

Riazanov, A. & Voronkov, A. [2001], Splitting without backtracking., *in* B. Nebel, ed., 'IJCAI', Morgan Kaufmann, pp. 611–617. 111

Riazanov, A. & Voronkov, A. [2002], 'The design and implementation of vampire', *AI Commun.* **15**(2), 91–110. 3, 60, 109

Robinson, G. A. & Wos, L. [1969], Paramodulation and theorem proving in first order theories with equality, *in* Meltzer & Mitchie, eds, 'Machine Intelligence 4', Edinburg University Press. 59

Robinson, J. A. [1965], 'A machine-oriented logic based on the resolution principle', *Journal of the ACM* **12**(1), 23–41. 35

Robinson, J. A. & Voronkov, A., eds [2001], *Handbook of Automated Reasoning (in 2 volumes)*, Elsevier and MIT Press. 126, 127, 129, 130

Schmidt, R. A. [1997], Optimised Modal Translation and Resolution, PhD thesis, Universität des Saarlandes, Saarbrücken, Germany. 3

Schulz, S. [2002], 'E – A Brainiac Theorem Prover', *Journal of AI Communications* **15**(2/3), 111–126. 60

Struth, G. [2001], Deriving focused calculi for transitive relations, *in* A. Middeldorp, ed., 'Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA 2001)', Vol. 2051 of *Lecture Notes in Computer Science*, Springer, pp. 291–305. 74

Tammet, T. [1990], The resolution program, able to decide some solvable classes, *in* P. Martin-Löf & G. Mints, eds, 'Proceedings of the International Conference on Computer Logic (COLOG-88)', Vol. 417 of *LNCS*, Springer, pp. 300–312. 3

Weidenbach, C. [2001], Combining superposition, sorts and splitting, *in* Robinson & Voronkov [2001], chapter 27, pp. 1965–2013. 110, 112, 115, 116, 117

Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobalt, C. & Topić, D. [2002], SPASS version 2.0, *in* A. Voronkov, ed., 'Automated deduction, CADE-18 : 18th International Conference on Automated Deduction', Vol. 2392 of *Lecture Notes in Artificial Intelligence*, Springer, Kopenhagen, Denmark, pp. 275–279. 3, 60, 109

Index

A