E-Hyper Tableaux with Distinct Object Identifiers

Markus Bender

Nr. 1/2013

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Kontaktdaten der Verfasser

Markus Bender
Institut für Informatik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
E-Mail: mbender@uni-koblenz.de

E-Hyper Tableaux with Distinct Object Identifiers

Markus Bender

Universität Koblenz-Landau, Institut für Informatik, 56070 Koblenz, Germany
mbender@uni-koblenz.de

Abstract E-KRHyper is a versatile theorem prover and model generator for first-order logic that natively supports equality. Inequality of constants, however, has to be given by explicitly adding facts. As the amount of these facts grows quadratically in the number of these distinct constants, the knowledge base is blown up. This makes it harder for a human reader to focus on the actual problem, and impairs the reasoning process. We extend E-KRHyper's underlying E-hyper tableau calculus to avoid this blow-up by implementing a native handling for inequality of constants. This is done by introducing the unique name assumption for a subset of the constants (the so called distinct object identifiers). The obtained calculus is shown to be sound and complete and is implemented into the E-KRHyper system. Synthetic benchmarks, situated in the theory of arrays, are used to back up the benefits of the new calculus.

1 Introduction

In (automated) theorem proving, there are some problems that need information on the inequality of certain constants [1,3,14]. In most cases this information is provided by adding facts of form $\mathit{false} \leftarrow c_0 = c_1$ to the knowledge base. Such a fact explicitly states that the two constants c_0 and c_1 are unequal. As the amount of these facts grows quadratically in the number of constants, they clutter the knowledge base and distract human readers of the problem from its actual proposition. Additionally, it is safe to assume that a larger knowledge base reduces the performance of a theorem prover in many applications, which is another drawback of explicit inequality facts.

A possibility to avoid such a blow-up is the introduction of native handling of inequality of constants and thus remove the need for inequality facts in the knowledge base. This can be done by using the *unique name assumption* in these reasoning tasks. The unique name assumptions states that two constants are identical if and only if their interpretation is identical. Instead of forcing the unique name assumption onto all constants, we apply it on a subset of the constants, called the *distinct object identifiers*. Implicit handling of non-identical constants makes the problems easier to comprehend and reduces the execution time of reasoning.

Reasoning with the unique name assumption gained relevance as the possibility to use the unique name assumption became part of the Thousands of Problems for Theorem Provers library (TPTP) [26] in the form of *distinct object identifiers*. Moreover, a version of the superposition calculus that treats distinct objects identifiers natively was introduced in [23].

```
sel(sto(A,I,E),I)=E.
sel(sto(A,I,E),J)=sel(A,J):- not(X=Y).
A=B:-sel(A,sk(A,B))=sel(B,sk(A,B)).
sto(sto(sto(sto(sto(a,i0,e0),i1,e1),i3,e3),i4,e4),i2,e2)=
sto(sto(sto(sto(sto(a,i0,e0),i1,e1),i2,e2),i3,e3),i4,e0).
false:-i0=i1.   false:-e0=e1.
false:-i0=i2.   false:-e0=e2.
false:-i0=i3.   false:-e0=e3.
false:-i0=i4.   false:-e0=e4.
false:-i1=i2.   false:-e1=e2.
false:-i1=i3.   false:-e1=e3.
false:-i1=i4.   false:-e1=e4.
false:-i2=i3.   false:-e2=e3.
false:-i2=i4.   false:-e2=e4.
false:-i3=i4.   false:-e3=e4.
```

(a)

```
sel(sto(A,I,E),I)=E.
sel(sto(A,I,E),J)=sel(A,J):- not(X=Y).
A=B:-sel(A,sk(A,B))=sel(B,sk(A,B)).
sto(sto(sto(sto(sto(a,i0,e0),i1,e1),i3,e3),i4,e4),i2,e2)=
sto(sto(sto(sto(sto(a,i0,e0),i1,e1),i2,e2),i3,e3),i4,e0).
```

(b)

Figure 1: A problem with explicit inequality (a) and implicit inequality (b) of constants.

As a motivational example, Figure 1 shows two sets of clauses, which formalize the same problem. In Figure 1a the inequalities are explicitly stated and in Figure 1b implicit knowledge of inequalities is used. This paper shows how to integrate the unique name assumption into the E-hyper tableau calculus, which is an efficient model generation and proof procedure for first-order logic. We provide formal proofs that the adapted calculus is sound and complete. Additionally we show how to incorporate extended calculus into the E-KRHyper theorem prover which is an implementation of the original E-hyper tableau calculus and is a well established theorem prover for first-order logic with equality with many areas of use, including amongst others natural question answering [13], e-learning [6,7] and ontology reasoning [10]. By empiric evaluation, we show that the changed implementation, which is able to use the unique name assumption natively, is superior to the traditional version of E-KRHyper.

2 Preliminaries

2.1 First-Order Logic

The language used is a common first-order logic consisting of the logical symbols $\forall, \exists, \vee, \wedge, \neg$, the elements of the set of *variables* $\mathbb{V}$, the set of *function* symbols $\mathbb{F}$ and the set of *predicate* symbols $\mathbb{P}$. It is denoted by $\mathfrak{L}$ and determined by the fixed signature $\Sigma = (\mathbb{F}, \mathbb{P})$.

$\mathbb{V}$, $\mathbb{F}$, $\mathbb{P}$ are possibly infinite, non-empty and mutually disjoint. The elements of $\mathbb{F}$ and $\mathbb{P}$ have a fixed arity. Function symbols with arity zero are called *constants*, where $\mathbb{C} \subseteq \mathbb{F}$ is the set of constants.

$\mathbb{T}$ is the set of *terms* of $\mathfrak{L}$ and is inductively defined as follows:

1. All constants and variables are terms.
2. If $f \in \mathbb{F}$ has arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

The set of *subterms* of a term u are defined as follows:

1. u is a subterm of u.
2. if $u = f(t_1, \ldots, t_n)$ then all subterms of $t_1, \ldots, t_n$ are subterms of u.

A subterm of u that is not u itself is called *proper subterm*.

A *position* is a sequence of natural numbers that allows us to refer to a specific subterm of a term in the following way: Let t be a term and p be a position then $t|_p$ denotes the subterm of t at position p. Let ϵ be the empty sequence and $t = f(t_1, \ldots, t_n)$, then $t|_\epsilon = t$ and $t|_{i.p} = t_i|_p$ for $1 \leq i \leq n$. We use the notation $t[s]_p$ instead of $t|_p = s$. $t[p/s']$ denotes the term obtained by replacing $t|_p$ with s' at position p in t. If p is obvious or unimportant within the context, then $t[s]$ denotes the term t with the subterm s, and $t[s']$ denotes the same term t except for its subterm s having been replaced by s'.

The *set of variables* of a term t is denoted by $vars(t)$. A term t is called *ground* iff $vars(t) = \emptyset$.

A mapping $\sigma : \mathbb{V} \to \mathbb{T}$, with finite *domain* $dom(\sigma) = \{x \mid x \neq \sigma x\}$ and a finite *range* $ran(\sigma) = \{x\sigma \mid x \neq \sigma x\}$, $x \in \mathbb{V}$ is called *substitution*. A substitution γ with $vars(ran(\gamma)) = \emptyset$ is called *ground*. A substitution $\rho : \mathbb{V} \to \mathbb{V}$ is called *renaming*. Iff $s\sigma = t\sigma$ for the terms s and t and the substitution σ, then σ is a *unifier* for s and t. σ is a *most general unifier* (mgu) iff for any other unifier τ for s and t there is a substitution ψ with $\sigma\psi = \tau$ A term s is an *instance* of a term t (written as $s \gtrsim t$) iff there is a substitution σ such that $s\sigma = t$. A term s is a *variant* of t (written as $s \sim t$) iff there is a renaming ρ such that $s\rho = t$.

The set of *formulæ* of $\mathfrak{L}$ and is inductively defined as follows:

1. If $p \in \mathbb{P}$ has arity n and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a formula.
2. If A, B are formulæ then $\neg A, A \wedge B, A \vee B$ are formulæ.
3. If A is a formula and $x \in \mathbb{V}$ then $\forall x A, \exists x A$ are formulæ.

A formula of form $p(t_1, \ldots, t_n)$ with $p \in \mathbb{P}$ and $t_1, \ldots, t_n \in \mathbb{T}$ is called *atomic formula* or *atom*.

As it eases the introduction of the calculus, we assume that the equality $\simeq$ is the only element in $\mathbb{P}$. This does not restrict our approach, as a formula of form $p(x)$ can be rewritten as $p(x) \simeq \mathtt{t}$, where $\mathtt{t}$ is a special term. Formally this entails that predicates are functions with the signature $\mathbb{T}^* \to \{true, false\}$ but the notions of typed functions and typed logic are not introduced in this paper. For the sake of brevity, we write $p(x)$ instead of $p(x) \simeq \mathtt{t}$.

A *literal* is an atom or the negation of an atom. A literal $K = \overline{L}$ is called the *complement* of L. A literal is *ground* iff its component terms are ground.

The notions of the set of variables, substitutions, renamings, unifiers, instances and variants are extended to literals in the natural way.

A *clause* $C = A_1 \vee \ldots \vee A_m \vee \neg B_1 \vee \ldots \vee \neg B_n$ of this language is a set of literals, usually written as an implication $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ with $m, n \geq 0$. The set

$\mathcal{A} = \{A_1, \ldots, A_m\}$ is called the *head* of the clause C while $\mathcal{B} = \{B_1, \ldots., B_n\}$ is called the *body* of C. Accordingly, $A_1, \ldots, A_m$ denote both the head atoms and the head literals of C, while $B_1, \ldots, B_n$ refer to the body atoms and $\neg B_1, \ldots, \neg B_n$ to the body literals. The notation $A, \mathcal{A} \leftarrow B, \mathcal{B}$ refers to the clause with the head atoms $\{A\} \cup \mathcal{A}$ and the body atoms $\{B\} \cup \mathcal{B}$. A *unit* is a clause consisting of exactly one literal. A clause is *empty* if both its head and its body are empty. The empty clause is denoted by $\square$. A clause is *ground* iff its literals are ground, i.e. do not contain variables. A clause is *pure* if none of its distinct head literals share variables . A substitution π is a *purifying substitution* for C iff $C\pi$ is pure. A substitution σ is a *unifier* for the terms s and t if $s\sigma = t\sigma$. σ is a *most general unifier* (mgu), if for any other unifier τ for s and t there is a substitution ψ with $\sigma\psi = \tau$.

All variables in a clause are taken to be universally quantified. A *clause set* is a conjunction of clauses and is sometimes called a *knowledge base*. Ω_Σ is the *universal set* of clauses, which contains all possible clauses for a given signature Σ.

As we assume that $\simeq$ is the only predicate symbol of the language $\mathfrak{L}$ with signature Σ, the *Herbrand interpretation I* is a set of ground Σ-equations that are considered true in I. The notions of satisfiability and validity are defined as usual. $I \models F$ denotes that I satisfies F, with F being a ground Σ-literal, a ground Σ-clause or a set thereof.

An *E-Interpretation* is an interpretation that is also a congruence relation on the Σ-terms. I^E denotes the smallest congruence relation on the Σ-terms that includes I. $I^E \models F$ denotes that I^E-satisfies F; this will be written as $I \models_E F$, though. If every E-interpretation satisfying F also satisfies F', then this means that F E-entails F', written as $F \models_E F'$.

Creating equivalence classes by using Herbrand models and the properties of the equality (symmetry, transitivity, reflexivity) leads to an easily understandable representation of a model. This is illustrated in Example 1.

Example 1 (E-Interpretation). Assume the set of literals $\{a \simeq b, c \simeq d, a \simeq d, f \simeq g, a \simeq g, h \simeq i, i \simeq j\}$ is a Herbrand model for some formula. Then with the corresponding E-Interpretation $\{\{a, b, c, d, f, g\}, \{i, j, h\}\}$ it is easier to see that the formula is satisfied iff the constants in the two equivalence classes are equal. $\square$

2.2 Term Ordering

For efficient equality reasoning with the E-hyper tableau calculus a *reduction ordering* $\succ$ is needed. It has to be total on ground terms. A reduction ordering is

1. a strict partial ordering (irreflexive, antisymmetric and transitive),
2. well-founded,
3. closed under context - if $s \succ s'$ for $s, s' \in \mathbb{T}$, then $t[p/s] \succ t[p/s']$ for any $t \in \mathbb{T}$ and any position p in t, and finally
4. liftable - if $s \succ t$ for $s, t \in \mathbb{T}$, then $s\sigma \succ t\sigma$ for any substitution σ.

As the specific ordering is not of interest in the context of this work, we refer to [17] for further information. $\succ$ is lifted to atoms, literals and clauses in the natural way and it induces the non-strict ordering $\succeq$. The converse is denoted by $\prec$ and $\preceq$ respectively.

2.3 Negligible Clauses

Another important part in efficient reasoning is to generate as few clauses as possible. For this, it is necessary to define criteria by which clauses can be identified as not useful for the reasoning process.

The first criterion is that of redundancy, which means that a clause is not helpful if it follows from a set clauses that are smaller w.r.t. $>$. Before this informal description is formalized in Definition 1 following [4] the notation $\mathcal{S}_{\mathcal{D}}$ is introduced.

Let $\mathcal{S}'$ be the set of all ground instances of all clauses in a clause set $\mathcal{S}$, then $\mathcal{S}_{\mathcal{D}} = \{C \in \mathcal{S}' \mid \mathcal{D} > C\}$ is the set of all ground instances of all clauses of $\mathcal{S}$ that are smaller w.r.t. $>$ than $\mathcal{D}$.

Definition 1 (Redundancy). *Let $\mathcal{D}$ be a clause, $\mathcal{S}$ a set of clauses.*

- *A ground clause $\mathcal{D}$ is* redundant w.r.t. a clause set $\mathcal{S}$ *iff $\mathcal{S}_{\mathcal{D}} \models_E \mathcal{D}$.*
- *A non-ground clause $\mathcal{D}$ is redundant w.r.t. a clause set $\mathcal{S}$ iff every ground instance of $\mathcal{D}$ is redundant w.r.t. $\mathcal{S}$.*

■

The second criterion is that of non-proper subsumption. A clause is non-properly subsumed by another clause iff it is an instance of this clause. A clause is non-properly subsumed by a set of clauses iff it is an instance of a clause of this particular clause set.

Definition 2 (Non-proper Subsumption). *Let $\mathcal{D}$ be a clause and $\mathcal{S}$ a set of clauses.*

- *A clause $\mathcal{D}$ is* non-properly subsumed *by a clause C iff there is a substitution σ such that $\mathcal{D} = C\sigma$.*
- *A clause $\mathcal{D}$ is non-properly subsumed w.r.t. a clause set $\mathcal{S}$ iff there is a $C \in \mathcal{S}$ that non-properly subsumes $\mathcal{D}$.*

■

A clause is negligible w.r.t. a set of clauses iff it is redundant w.r.t. this set or non-properly subsumed w.r.t. it.

Definition 3 (Negligible Clauses (prelim.)). *Let $\mathcal{D}$ be a clause and $\mathcal{S}$ a set of clauses. A clause $\mathcal{D}$ is* negligible w.r.t. a clause set $\mathcal{S}$ *iff at least one of the following holds:*

- *$\mathcal{D}$ is redundant w.r.t. $\mathcal{S}$.*
- *$\mathcal{D}$ is non-properly subsumed w.r.t. $\mathcal{S}$.*

■

To avoid confusion with the names of definitions, *(prelim.)* is appended to the names of these definitions that are extended later on to indicate that these are preliminary definitions.

In most cases it is obvious what set of clauses $\mathcal{S}$ is meant and therefore the addition *w.r.t. to a clause set $\mathcal{S}$* might be omitted for the sake of brevity.

2.4 Trees and Tableaux

A *tree* is a directed, acyclic graph denoted by a pair $(\mathbb{N}, \mathbb{E})$ consisting of a set of nodes $\mathbb{N}$ and a set of edges $\mathbb{E} \subset (\mathbb{N} \times \mathbb{N})$. A node that has no incoming edges is called *root*. A node with not outgoing edges is called *leaf*.

A *branch* $\mathbf{B}$ in $\mathbf{T}$ is a sequence $\mathbf{N}_0, \ldots, \mathbf{N}_n$ of nodes in $\mathbf{T}$, with $\mathbf{N}_0$ being the root node of $\mathbf{T}$, each $\mathbf{N}_i$ being the immediate predecessor of $\mathbf{N}_{i+1} (0 \leq i < n)$, and $\mathbf{N}_n$ being a leaf of $\mathbf{T}$.

A *initial segment* $\mathbf{B}^j$ of a branch $\mathbf{B} = (\mathbf{N}_i)_{0 \leq i < \nu}$ is defined as a sequence of nodes of $\mathbf{B}$ starting with $\mathbf{N}_0$ and ending with $\mathbf{N}_j$, i.e. $\mathbf{B}^j = (\mathbf{N}_i)_{0 \leq i \leq j}$.

3 E-Hyper Tableau Calculus

3.1 Introduction

The *E-hyper tableau calculus* [9] was developed in 2007 as an extension of the *hyper tableau calculus* [8]. The latter was developed in 1996 as an efficient model generation and proof-procedure for first-order theories. It combines the benefits of tableau calculi, i.e. rich structure for the derivation process, partial models as by-product, and the advantages of the hyper resolution, i.e. the hyper property for resolving negative literals of a clause in a single inference step, universally quantified variables and subsumption as pruning technique [8].

As the hyper tableau calculus lacked the native treatment of equalities, which is mandatory in most proving applications, it was extended by new rules that introduced the superposition calculus' [4] ideas for equality handling to the hyper tableau calculus. This extensions lead to the E-hyper tableau calculus, which is introduced in this section. We only give a brief introduction and refer to [17] and [9] for further information.

An *E-hyper tableau* $\mathbf{T}$ *over a signature* Σ is a pair $(\mathbf{t}, \lambda)$, where $\mathbf{t}$ is a finite, ordered tree and λ is a labelling function assigning an Σ-clause to each node of $\mathbf{t}$.

A *branch* $\mathbf{B}$ in $\mathbf{T}$ is a sequence $\mathbf{N}_0, \ldots, \mathbf{N}_n$ of nodes in $\mathbf{T}$, with $\mathbf{N}_0$ being the root node of $\mathbf{T}$, each $\mathbf{N}_i$ being the immediate predecessor of $\mathbf{N}_{i+1} (0 \leq i < n)$, and $\mathbf{N}_n$ being a leaf of $\mathbf{T}$.

$\lambda(\mathbf{B}) = \{\lambda(\mathbf{N}_0), \ldots, \lambda(\mathbf{N}_n)\}$ is the set of clauses in $\mathbf{B}$, called the *tableau clauses*. The notation $C \in \mathbf{B}$ is used iff $C \in \lambda(\mathbf{B})$.

The notation $\mathbf{B} \cdot C$ represents the tableau branch obtained from attaching a node labelled with C to the leaf of $\mathbf{B}$, where $\mathbf{B} \cdot \mathbf{B}'$ represents the tableau obtained from concatenating $\mathbf{B}$ and the node sequence $\mathbf{B}'$.

A branch in an E-hyper tableau is *closed* iff it contains the empty clause $\square$ otherwise it is *open*. An E-hyper tableau is *closed* iff all of its branches are closed, and it is open otherwise.

3.2 E-Hyper Tableaux Rules

The E-hyper tableau calculus uses the eight rules shown in Figure 2, which can be classified into two different types of rules. The first group consists of the four inference rules sup-left, unit-sup-right, ref, split, which create new clauses. The second group

$$\textsf{sup-left}(\sigma)\ \frac{\mathcal{A} \leftarrow s[u'] \simeq t, \mathcal{B} \qquad u \simeq r \leftarrow}{(\mathcal{A} \leftarrow s[r] \simeq t, \mathcal{B})\sigma} \quad \text{if (1-4) holds}$$

$$\textsf{unit-sup-right}(\sigma)\ \frac{s[u'] \simeq t \leftarrow \qquad u \simeq r \leftarrow}{(s[r] \simeq t \leftarrow)\sigma} \quad \text{if (1-5) holds}$$

$$\textsf{ref}(\sigma)\ \frac{\mathcal{A} \leftarrow s \simeq t, \mathcal{B}}{(\mathcal{A} \leftarrow \mathcal{B})\sigma} \quad \text{if (6) holds}$$

$$\textsf{split}(\pi)\ \frac{A_1, \ldots, A_n \leftarrow}{A_1\pi \leftarrow \quad \ldots \quad A_n\pi \leftarrow} \quad \text{if (7,8) holds}$$

Conditions:

1. u' is not a variable
2. σ is a mgu of u and u'
3. $u\sigma \not\preceq r\sigma$
4. $s\sigma \not\preceq t\sigma$
5. $(s \simeq \mathtt{t})\sigma \not\preceq (u \simeq r)\sigma$
6. σ is a mgu of s and t
7. $n \geq 2$
8. π is a purifying substitution for $A_1, \ldots, A_n \leftarrow$

$$\textsf{Equality}\ \frac{\mathbf{B}}{\mathbf{B} \cdot \mathcal{E}} \quad \text{if (1-4) holds}$$

$$\textsf{Split}\ \frac{\mathbf{B}}{\mathbf{B} \cdot A_1 \leftarrow^d \quad \ldots \quad \mathbf{B} \cdot A_n \leftarrow^d} \quad \text{if (1,5,6) holds}$$

$$\textsf{Del}\ \frac{\mathbf{B} \cdot C^{(d)} \cdot \mathbf{B}_1 \cdot \mathbf{B}_2}{\mathbf{B} \cdot \mathtt{t} \simeq \mathtt{t}^{(d)} \cdot \mathbf{B}_1 \cdot \mathbf{B}_2} \quad \text{if (7,8) holds}$$

$$\textsf{Simp}\ \frac{\mathbf{B} \cdot C^{(d)} \cdot \mathbf{B}_1 \cdot \mathbf{B}_2}{\mathbf{B} \cdot \mathcal{D}^{(d)} \cdot \mathbf{B}_1 \cdot \mathbf{B}_2} \quad \text{if (8-10) holds}$$

Conditions:

1. there is a $C \in \mathbf{B}$
2. there is a fresh variant $\mathcal{D}$ of a positive unit clause in $\mathbf{B}$
3. there is a σ such that $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$ with $\mathsf{R} \in$ {sup-left, unit-sup-right} or $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$
4. $\mathbf{B}$ contains no variant of $\mathcal{E}$
5. there is a π such that $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_n \leftarrow$
6. $\mathbf{B}$ contains no variant of $A_i, i \in \{1, \ldots, n\}$
7. C is negligible w.r.t. $\mathbf{B} \cdot \mathbf{B}_1$
8. $\mathbf{B}_1$ does not contain a decision clause
9. $\mathbf{B} \cdot C \cdot \mathbf{B}_1 \models_E \mathcal{D}$
10. C is redundant w.r.t. $\mathbf{B} \cdot \mathcal{D} \cdot \mathbf{B}_1$

Figure 2: Rules for the E-hyper tableau calculus and their required conditions

consists of the tableaux rules Equality, Split, Del, Simp, which allow the modification or extension an E-hyper tableau.

The first three of the inference rules sup-left, unit-sup-right, ref are adapted from the superposition calculus to deal with equality. If they are applicable, each of the rules takes a set of clauses as input and derives a new clause. The fourth so-called *split* rule is used to to create a branch split from a disjunctive clause head.

The sup-left rule (*superposition left*) applies a positive unit equation $\mathcal{D}$ to a body literal of another clause C. This inference instance will be denoted by $C, \mathcal{D} \Rightarrow_{\mathsf{sup\text{-}left}(\sigma)} \mathcal{E}$, where $\mathcal{E}$ is the resulting clause and σ is a mgu.

The unit-sup-right rule (*unit superposition right*) applies a positive unit equation $\mathcal{D}$ to another positive unit equation C. This inference instance will be denoted by $C, \mathcal{D} \Rightarrow_{\mathsf{unit\text{-}sup\text{-}right}(\sigma)} \mathcal{E}$, where $\mathcal{E}$ is the resulting clause and σ is a mgu.

The ref rule (*reflexivity*) works on a single clause C and removes an equational body literal whose both sides can be unified. This inference instance will be denoted by $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$, where $\mathcal{E}$ is the resulting clause and σ is a mgu.

The split rule (*split*) works on a positive disjunction C. This inference instance will be denoted by $C \Rightarrow_{\mathsf{split}(\pi)} A_1\pi \leftarrow, \ldots, A_m\pi \leftarrow$, where $A_1\pi \leftarrow$, …, $A_m\pi \leftarrow$ are the resulting clauses and π is a purifying substitution.

As these inference rules work only on clauses, the extension rules Equality and Split are introduced to work on tableaux. If **T** is an E-hyper tableau with a branch **B**, then **T** can be extended by application of these two rules. The Equality rule defines how sup-left, unit-sup-right and ref can be applied on an open branch of a tableau, and the Split rule defines how the split rule can be used to create an branch split.

There are the two additional tableaux rules, Del and Simp, which allow the modification of existing nodes' labelling. The Del rule (*deletion*) eliminates redundant or non-properly subsumed clauses (or more specifically, it overwrites such a clause with a trivially true unit clause, thus preserving the node while changing its label). The Simp rule (*simplification*) overwrites a clause with one that is smaller according to the term ordering.

The annotation ^{d} marks the derived clauses as *decision clauses*. A node labelled with a decision clause will be referred to as a *decision node*. The notation $^{(d)}$ indicates that a decision node remains to be a decision node after the application of the Del or Simp rule on the node. The concept of a decision node is needed to restrict the application of the Del and Simp rule in such a way, that the redundancy or the non-proper subsumption of a node is only considered w.r.t. all the clauses in the branch that appear between the target node and the next branch split. This restriction is crucial for the soundness and the completeness of the calculus. (see [9] for details).

Several notions are now extended to the inference rules sup-left, unit-sup-right, ref and split, as well as to their instances:

- An inference is *ground* iff its constituent clauses (premisses and conclusions) are ground. For ground inferences the substitution σ and the purifying substitution π can both be assumed to be the empty substitution ϵ.
- If $C, \mathcal{D} \Rightarrow_{\mathsf{sup\text{-}left}(\sigma)} \mathcal{E}$ is a sup-left inference and
  γ is a substitution such that $C\sigma\gamma, \mathcal{D}\sigma\gamma \Rightarrow_{\mathsf{sup\text{-}left}(\epsilon)} \mathcal{E}\gamma$ is ground,
  then the latter is called a ground instance of $C, \mathcal{D} \Rightarrow_{\mathsf{sup\text{-}left}(\sigma)} \mathcal{E}$.
- If $C, \mathcal{D} \Rightarrow_{\mathsf{unit\text{-}sup\text{-}right}(\sigma)} \mathcal{E}$ is a unit-sup-right inference and
  γ is a substitution such that $C\sigma\gamma, \mathcal{D}\sigma\gamma \Rightarrow_{\mathsf{unit\text{-}sup\text{-}right}(\epsilon)} \mathcal{E}\gamma$ is ground,
  then the latter is called a ground instance of $C, \mathcal{D} \Rightarrow_{\mathsf{unit\text{-}sup\text{-}right}(\sigma)} \mathcal{E}$.
- If $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ is a ref inference and
  γ is a substitution such that $C\sigma\gamma \Rightarrow_{\mathsf{ref}(\epsilon)} \mathcal{E}\gamma$ is ground,
  then the latter is called a ground instance of $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$.
- If $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is a split inference and
  γ is a substitution such that $C\pi \Rightarrow_{\mathsf{split}(\epsilon)} A_1\gamma \leftarrow, \ldots, A_m\gamma \leftarrow$ is ground,
  then the latter inference is a ground instance of the former.
- Let $\mathcal{S}$ be a set of not necessarily ground clauses:
  - A ground inference $C, \mathcal{D} \Rightarrow_{\mathsf{sup\text{-}left}(\epsilon)} \mathcal{E}$ is *redundant w.r.t.* $\mathcal{S}$
    iff $\mathcal{E}$ is redundant w.r.t. $\mathcal{S}_C \cup \{\mathcal{D}\}$.

- A ground inference $C, \mathcal{D} \Rightarrow_{\textsf{unit-sup-right}(\epsilon)} \mathcal{E}$ is *redundant w.r.t.* $\mathcal{S}$ iff $\mathcal{E}$ is redundant w.r.t. $\mathcal{S}_C \cup \{\mathcal{D}\}$.
- A ground inference $C \Rightarrow_{\textsf{ref}(\epsilon)} \mathcal{E}$ is *redundant w.r.t.* $\mathcal{S}$ iff $\mathcal{E}$ is redundant w.r.t. $\mathcal{S}_C$.
- A ground inference $C \Rightarrow_{\textsf{split}(\epsilon)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is *redundant w.r.t.* $\mathcal{S}$ iff there is an i with $1 \leq i \leq m$ such that $A_i \leftarrow$ is redundant w.r.t. $\mathcal{S}_C$.

– For all inference rules sup-left, unit-sup-right, ref and split, a (possibly non-ground) inference is redundant w.r.t. $\mathcal{S}$ iff each of its ground instances is redundant w.r.t. $\mathcal{S}$.

Now it is possible to define the notion of a clause set being *saturated up to redundancy*, which is an important part for the completeness of the calculus.

Definition 4 (Saturation up to Redundancy (prelim.)). *A clause set $\mathcal{S}$ is* saturated up to redundancy *iff for all clauses $C \in \mathcal{S}$ such that C is not redundant w.r.t. $\mathcal{S}$ all of the following hold:*

1. *Every inference $C, \mathcal{D} \Rightarrow_{\textsf{sup-left}(\sigma)} \mathcal{E}$, where $\mathcal{D}$ is a fresh variant of a positive unit clause from $\mathcal{S}$, such that neither $C\sigma$ nor $\mathcal{D}\sigma$ is redundant w.r.t. $\mathcal{S}$, is redundant w.r.t. $\mathcal{S}$.*
2. *Every inference $C, \mathcal{D} \Rightarrow_{\textsf{unit-sup-right}(\sigma)} \mathcal{E}$, where $\mathcal{D}$ is a fresh variant of a positive unit clause from $\mathcal{S}$, such that neither $C\sigma$ nor $\mathcal{D}\sigma$ is redundant w.r.t. $\mathcal{S}$, is redundant w.r.t. $\mathcal{S}$.*
3. *Every inference $C \Rightarrow_{\textsf{ref}(\sigma)} \mathcal{E}$, such that $C\sigma$ is not redundant w.r.t. $\mathcal{S}$, is redundant w.r.t. $\mathcal{S}$.*
4. *Every inference $C \Rightarrow_{\textsf{split}(\sigma)} A_1 \leftarrow, \ldots, A_m \leftarrow$, such that $C\pi$ is not redundant w.r.t. $\mathcal{S}$, is redundant w.r.t. $\mathcal{S}$.*

■

With the notion of an E-hyper tableau and rules to modify E-hyper tableaux, the concept of a *derivation* can now be introduced. A derivation is a series of tableaux that starts with the initial tableau. For all tableaux in this series holds that they are either the initial tableau or they have been derived by applying one of the introduced rules to a tableau of the series. A more formal definition is now given in Definition 5 and then shown in Example 2.

Definition 5 (E-hyper Tableau Derivation (prelim.)). *An* E-hyper tableau derivation *of a set $\{C_1, \ldots, C_n\}$ of Σ-clauses is a possibly infinite sequence of tableaux $\mathbf{D} = (\mathbf{T}_i)_{0 \leq i < \kappa}$ such that*

1. *$\mathbf{T}_0$ is the clausal tableau over Σ that consists of a single branch of length n with the tableau clauses $C_1, \ldots, C_n$, and*
2. *for all $i > 0$, $\mathbf{T}_i$ is obtained from $\mathbf{T}_{i-1}$ by a single application of the Equality, Split, Del or Simp rule to an open branch in $\mathbf{T}_i$.*

■

$$\begin{tikzcd}[column sep=tiny]
\mathbf{T}_0 \\
f(x) \simeq x \leftarrow \arrow[d, no head] \\
f(c) \simeq c \leftarrow \arrow[d, no head] \\
p(f(a)) \simeq \mathtt{t} \leftarrow \arrow[d, no head] \\
q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow p(a) \simeq \mathtt{t}
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=tiny]
 & \mathbf{T}_6 & \\
 & f(x) \simeq x \leftarrow \arrow[d, no head] & \\
 & \mathtt{t} \simeq \mathtt{t} \leftarrow \arrow[d, no head] & \\
 & p(f(a)) \simeq \mathtt{t} \leftarrow \arrow[d, no head] & \\
 & q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow p(a) \simeq \mathtt{t} \arrow[d, no head] & \\
 & p(a) \simeq \mathtt{t} \leftarrow \arrow[d, no head] & \\
 & q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow \mathtt{t} \simeq \mathtt{t} \arrow[d, no head] & \\
 & q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow \arrow[dl, no head] \arrow[dr, no head] & \\
q(f(a)) \simeq \mathtt{t} \leftarrow \arrow[d, no head] & & r(a) \simeq \mathtt{t} \leftarrow \\
q(a) \simeq \mathtt{t} \leftarrow & &
\end{tikzcd}$$

Figure 3: The tableaux $\mathbf{T}_0$ and $\mathbf{T}_6$ of an E-hyper tableau derivation.

Example 2 (E-Hyper Tableau Derivation). Figure 3 shows the tableaux $\mathbf{T}_0$ and $\mathbf{T}_6$ of an E-hyper tableau derivation. The intermediate tableaux $\mathbf{T}_1$ to $\mathbf{T}_5$ are not shown in the figure but are introduced textually.

$\mathbf{T}_0$ is the initial tableau of the derivation and contains four clauses. As it is easy to see there is a σ such that $(f(x) \simeq x \leftarrow)\sigma = f(c) \simeq c \leftarrow$ holds, namely $\sigma = \{x/c\}$ and therefore $f(c) \simeq c \leftarrow$ is non-properly subsumed by $f(x) \simeq x \leftarrow$ and can be rewritten to $\mathtt{t} \simeq \mathtt{t}$ by an application of the Del rule. This leads to the tableau $\mathbf{T}_1$. By applying the Equality rule with underlying inference $p(f(a)) \simeq \mathtt{t} \leftarrow, f(x) \simeq x \leftarrow \Rightarrow_{\text{unit-sup-right}(\sigma)} p(a) \simeq \mathtt{t} \leftarrow$ and $\sigma = \{x/a\}$ on $\mathbf{T}_1$, $\mathbf{T}_2$ is derived. For the sake of simplicity, the following steps are given as a list of underlying inference rules.

1. $\mathcal{C}, \mathcal{D} \Rightarrow_{\text{sup-left}(\sigma)} \mathcal{E}$, with
   $\mathcal{C} = q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow p(a) \simeq \mathtt{t}$
   $\mathcal{D} = p(a) \simeq \mathtt{t} \leftarrow$
   $\sigma = \{\}$
   $\mathcal{E} = q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow \mathtt{t} \simeq \mathtt{t}$
2. $\mathcal{C} \Rightarrow_{\text{ref}(\sigma)} \mathcal{E}$, with
   $\mathcal{C} = q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow \mathtt{t} \simeq \mathtt{t}$
   $\sigma = \{\}$
   $\mathcal{E} = q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow$
3. $\mathcal{C} \Rightarrow_{\text{split}(\pi)} A_1\pi \leftarrow, A_2\pi \leftarrow$, with
   $\mathcal{C} = q(f(x)) \simeq \mathtt{t}, r(x) \simeq \mathtt{t} \leftarrow$
   $\pi = \{x/a\}$
   $A_1\pi = q(f(a)) \simeq \mathtt{t}$
   $A_2\pi = r(a) \simeq \mathtt{t}$
4. $\mathcal{C}, \mathcal{D} \Rightarrow_{\text{unit-sup-right}(\sigma)} \mathcal{E}$, with
   $\mathcal{C} = q(f(a)) \simeq \mathtt{t} \leftarrow$
   $\mathcal{D} = f(x) \simeq x \leftarrow$
   $\sigma = \{\}$
   $\mathcal{E} = q(a) \simeq \mathtt{t} \leftarrow$

These four steps lead from $\mathbf{T}_2$ to $\mathbf{T}_6$. The application of the Del rule is independent and must not have happened as the first step but could have been somewhere between $\mathbf{T}_1$ and $\mathbf{T}_6$. The ordering of the other extension steps is fixed, as they need to be applied consecutively. □

For using the E-hyper tableaux calculus to calculate a model for a formula or to show that a formula is unsatisfiable, we need to define some criterion that states how many derivation steps are necessary to make such a statement. To do so, we need some notations and concepts that are introduced now.

The first one is the concept of a *limit tree*. It represents the overall tree structure of the derivation. Its set of nodes is the union of all nodes of all of the derivations tableaux and its set of edges is the union of all edges of all of the derivations tableaux. If a derivation is finite, the limit tree matches the tree of the last tableau in the derivation. A limit tree is not a tableau, as it has no associated labelling function. A more formal definition of limit tree is given now in Definition 6.

Definition 6 (Limit Tree). *Let* $\mathbf{T} = (\mathbf{t}, \lambda)$ *be an E-hyper tableau with* $\mathbf{t} = (\mathbb{N}, \mathbb{E})$ *a tree consisting of the set of nodes* $\mathbb{N}$ *and the set of edges* $\mathbb{E}$*. Furthermore let the derivation* $\mathbf{D} = ((\mathbb{N}_i, \mathbb{E}_i), \lambda_i)_{0 \le i < \kappa}$*. Then the* limit tree $\mathbf{t}_\infty$ *of the derivation* $\mathbf{D}$ *is defined as:*

$$\mathbf{t}_\infty = \left(\bigcup_{0 \le i < \kappa} \mathbb{N}_i, \bigcup_{0 \le i < \kappa} \mathbb{E}_i \right)$$

■

With the concept of a limit tree the set of *persistent clauses* can now be defined. It is the set of clauses that contains the labels of all the nodes that have not been rewritten by the Del or Simp rule. These clauses are used to generate a model for a given set of formulæ or to show that the set is unsatisfiable. To construct the set of persistent clauses the λ'-function is needed that is defined as follows:

If $\mathbf{N}$ is a node and λ is the labelling function then $\lambda'(\mathbf{N}) := \begin{cases} \{\lambda(\mathbf{N})\} & \text{if } \mathbf{N} \in dom(\lambda) \\ \Omega_\Sigma & \text{otherwise} \end{cases}$

Informally, it can be seen as a wrapper for λ that creates a set containing the result of λ. If λ is undefined for an input the universal set for the language is returned. This behaviour is needed, as we want to build the intersection of all labellings for a node throughout the whole derivation process and it might happen that a λ for a node is not yet defined in a specific derivation step. The set of persistent clauses is then created by joining these intersections for all nodes of a branch.

The definition of persistent clauses is given in Definition 7 and illustrated in Example 3.

Definition 7 (Persistent Clauses). *Let* $\mathbf{t}_\infty = (\bigcup_{0 \le i < \kappa} \mathbb{N}_i, \bigcup_{0 \le i < \kappa} \mathbb{E}_i)$ *be the limit tree of the derivation* $\mathbf{D} = ((\mathbb{N}_i, \mathbb{E}_i), \lambda_i)_{0 \le i < \kappa}$ *and* $\mathbf{B} = (\mathbf{N}_j)_{0 \le j < \nu}$ *be a (possibly infinite) branch in* $\mathbf{t}_\infty$.

Then the set of persistent clauses (of $\mathbf{B}$) *is defined as*

$$\mathbf{B}_\infty = \bigcup_{0 \le i < \nu} \left(\bigcap_{0 \le j < \kappa} \lambda'_j(\mathbf{N}_i) \right)$$

■

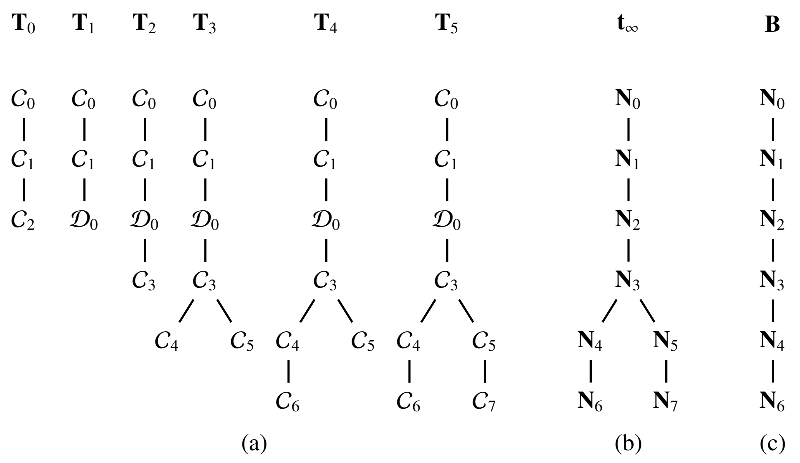


Figure 4: The derivation $\mathbf{D}$, starting with the initial tableau $\mathbf{T}_0$ (a), its limit tree $\mathbf{t}_\infty$ (b) and a single branch of $\mathbf{t}_\infty$(c).

$$\begin{array}{l}
\lambda'_0(\mathbf{N}_0) \cap \lambda'_1(\mathbf{N}_0) \cap \lambda'_2(\mathbf{N}_0) \cap \lambda'_3(\mathbf{N}_0) \cap \lambda'_4(\mathbf{N}_0) \cap \lambda'_5(\mathbf{N}_0) \bigcup \\
\lambda'_0(\mathbf{N}_1) \cap \lambda'_1(\mathbf{N}_1) \cap \lambda'_2(\mathbf{N}_1) \cap \lambda'_3(\mathbf{N}_1) \cap \lambda'_4(\mathbf{N}_1) \cap \lambda'_5(\mathbf{N}_1) \bigcup \\
\lambda'_0(\mathbf{N}_2) \cap \lambda'_1(\mathbf{N}_2) \cap \lambda'_2(\mathbf{N}_2) \cap \lambda'_3(\mathbf{N}_2) \cap \lambda'_4(\mathbf{N}_2) \cap \lambda'_5(\mathbf{N}_2) \bigcup \\
\lambda'_0(\mathbf{N}_3) \cap \lambda'_1(\mathbf{N}_3) \cap \lambda'_2(\mathbf{N}_3) \cap \lambda'_3(\mathbf{N}_3) \cap \lambda'_4(\mathbf{N}_3) \cap \lambda'_5(\mathbf{N}_3) \bigcup \\
\lambda'_0(\mathbf{N}_4) \cap \lambda'_1(\mathbf{N}_4) \cap \lambda'_2(\mathbf{N}_4) \cap \lambda'_3(\mathbf{N}_4) \cap \lambda'_4(\mathbf{N}_4) \cap \lambda'_5(\mathbf{N}_4) \bigcup \\
\lambda'_0(\mathbf{N}_6) \cap \lambda'_1(\mathbf{N}_6) \cap \lambda'_2(\mathbf{N}_6) \cap \lambda'_3(\mathbf{N}_6) \cap \lambda'_4(\mathbf{N}_6) \cap \lambda'_5(\mathbf{N}_6)
\end{array}$$

(a)

$$\begin{array}{cccccc}
\{\mathcal{C}_0\} \cap & \{\mathcal{C}_0\} \cap & \{\mathcal{C}_0\} \cap & \{\mathcal{C}_0\} \cap & \{\mathcal{C}_0\} \cap & \{\mathcal{C}_0\} \bigcup \\
\{\mathcal{C}_1\} \cap & \{\mathcal{C}_1\} \cap & \{\mathcal{C}_1\} \cap & \{\mathcal{C}_1\} \cap & \{\mathcal{C}_1\} \cap & \{\mathcal{C}_1\} \bigcup \\
\{\mathcal{C}_2\} \cap & \{\mathcal{D}_0\} \cap & \{\mathcal{D}_0\} \cap & \{\mathcal{D}_0\} \cap & \{\mathcal{D}_0\} \cap & \{\mathcal{D}_0\} \bigcup \\
\Omega_\Sigma \cap & \Omega_\Sigma \cap & \{\mathcal{C}_3\} \cap & \{\mathcal{C}_3\} \cap & \{\mathcal{C}_3\} \cap & \{\mathcal{C}_3\} \bigcup \\
\Omega_\Sigma \cap & \Omega_\Sigma \cap & \Omega_\Sigma \cap & \{\mathcal{C}_4\} \cap & \{\mathcal{C}_4\} \cap & \{\mathcal{C}_4\} \bigcup \\
\Omega_\Sigma \cap & \Omega_\Sigma \cap & \Omega_\Sigma \cap & \Omega_\Sigma \cap & \{\mathcal{C}_6\} \cap & \{\mathcal{C}_6\}
\end{array}$$

(b)

Table 1: Abstract (a) and concrete (b) calculation of the set of persistent clauses for the derivation shown in Figure 4.

Example 3 (Limit Trees and Persistent Clauses). In Figure 4a a derivation with five tableaux is given. For the sake of brevity neither concrete clauses nor concrete extension rules are given, but it is easy to see that such a derivation is possible.

This derivation has the limit tree $\mathbf{t}_\infty$ shown in Figure 4b. As the shown derivation is finite, the limit tree equals the tree of the last tableau of the derivation $\mathbf{T}_5$. In Figure 4c the leftmost branch of $\mathbf{t}_\infty$ is given as the branch $\mathbf{B}$.

Table 1 shows how the set of the persistent clauses is constructed for the branch $\mathbf{B}$. In Table 1a the definition of the set of persistent clauses is shown without evaluating the labelling functions. In Table 1b λ has been evaluated and the appropriate clauses are used in the formula.

This formula shows that it is possible that some nodes might not have a labelling at a certain derivation step as the tree has not been enough extended yet. For example $\mathbf{N}_3$ is introduced in $\mathbf{T}_2$ and therefore $\lambda'_0(\mathbf{N}_3) = \lambda'_1(\mathbf{N}_3) = \Omega_\Sigma$.

The set of persistent clauses for this example is then:

$$\mathbf{B}_\infty = \{C_0\} \cup \{C_1\} \cup \emptyset \cup \{C_3\} \cup \{C_4\} \cup \{C_6\} = \{C_0, C_1, C_3, C_4, C_6\}$$

Neither C_2 nor $\mathcal{D}_0$ are members of $\mathbf{t}_\infty$, which conforms our intention. C_2 has been rewritten, and therefore it was either redundant or non-properly subsumed and thus is not needed to construct a model or to show the unsatisfiability of the set of clauses. If C_2 was rewritten by the Del rule $\mathcal{D}_0 = \mathtt{t} \simeq \mathtt{t} \leftarrow$, it is easy to see that this clause is not of relevance. If the Simp rule caused the rewrite of C_2, it has been rewritten to a clause $\mathcal{D}_0$ that is smaller w.r.t. $\succ$ and already in the branch and thus $\mathcal{D}_0$ is already in $\mathbf{t}_\infty$.

□

With the concepts of limit trees and persistent clauses defined, the notion of an *exhausted branch* can now be introduced. When a branch is exhausted, all the useful rule applications are done and all following rule applications would not contribute to creating a model. The formal definition is given in Definition 8.

Definition 8 (Exhausted Branch (prelim.)). *Let $\mathbf{t}_\infty$ be a limit tree, $\mathbf{B} = (\mathbf{N}_k)_{0 \le k < \nu}$ be a branch in $\mathbf{t}_\infty$ and $\mathbf{B}^i$ and $\mathbf{B}^j$ initial segments of $\mathbf{B}$. The branch $\mathbf{B}$ is* exhausted *iff it does not contain the empty clause, and for every clause $C \in \mathbf{B}_\infty$ and every fresh variant $\mathcal{D}$ of every positive unit clause in $\mathbf{B}_\infty$ such that neither C nor $\mathcal{D}$ is redundant w.r.t. $\mathbf{B}_\infty$ all of the following hold, for all $i < \nu$ such that $C \in \mathbf{B}^i$ and $\mathcal{D}$ is a variant of a clause in $\mathbf{B}^i$:*

1. *if Equality is applicable to $\mathbf{B}^i$ with underlying inference*
   $C, \mathcal{D} \Rightarrow_{\textsf{sup-left}(\sigma)} \mathcal{E}$,
   and neither $C\sigma$ nor $\mathcal{D}\sigma$ is redundant w.r.t. $\mathbf{B}^i$,
   then there is a $j < \nu$ such that the inference
   $C, \mathcal{D} \Rightarrow_{\textsf{sup-left}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}^j$.
2. *if Equality is applicable to $\mathbf{B}^i$ with underlying inference*
   $C, \mathcal{D} \Rightarrow_{\textsf{unit-sup-right}(\sigma)} \mathcal{E}$,
   and neither $C\sigma$ nor $\mathcal{D}\sigma$ is redundant w.r.t. $\mathbf{B}^i$,
   then there is a $j < \nu$ such that the inference
   $C, \mathcal{D} \Rightarrow_{\textsf{unit-sup-right}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}^j$.
3. *if Equality is applicable to $\mathbf{B}^i$ with underlying inference*
   $C \Rightarrow_{\textsf{ref}(\sigma)} \mathcal{E}$
   and $C\sigma$ is not redundant w.r.t. $\mathbf{B}^i$,
   then there is a $j < \nu$ such that the inference
   $C \Rightarrow_{\textsf{ref}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}^j$.
4. *if Split is applicable to $\mathbf{B}^i$ with underlying inference*
   $C \Rightarrow_{\textsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$

and $C\pi$ is not redundant w.r.t. $\mathbf{B}^i$,
then there is a $j < \nu$ such that the inference
$C \Rightarrow_{split(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant w.r.t. $\mathbf{B}^j$.

■

We are now able to state which kind of derivation is of interest for deriving a model for a set of clauses $\mathcal{S}$ or showing that $\mathcal{S}$ is unsatisfiable, namely a *fair* derivation. A derivation is fair iff it contains a closed tableau or its limit tree has an exhausted branch.

A finite E-hyper tableau derivation of a clause set $\mathcal{S}$ that contains a closed tableau is called *E-hyper tableau refutation of $\mathcal{S}$*.

The concrete connection between an exhausted branch and a model for a clause set is given in section 5.

The E-hyper tableau calculus shown in this section is *complete* and *sound*. Proofs for this properties can be found in [9].

4 Distinct Objects in E-Hyper Tableaux

4.1 The Unique Name Assumption

The *unique name assumption* (UNA) is a convention on how to handle equality of objects in knowledge bases. It defines that two constants denote the same object if and only if the constants are identical [20].

To be more flexible and to conform the TPTP's way of dealing with the unique name assumption [26,25], the unique name assumption is not applied to the whole set of constants, but for a possibly infinite subset of it, called the set of *distinct object identifiers* (DOI) $\mathbb{D} \subseteq \mathbb{C}$. Elements of $\mathbb{D}$ are denoted by i or j.

We can now introduce the UNA more formally: For any interpretation I, the UNA holds for $\mathbb{D}$ iff $i \simeq j \Leftrightarrow I(i) = I(j)$ for all $i, j \in \mathbb{D}$.

With the introduction of $\mathbb{D}$, the language's signature has to be changed to $\Sigma = (\mathbb{D}, \mathbb{F}, \mathbb{P})$. From $\mathbb{D} \subseteq \mathbb{C}$ follows that $\mathbb{D}$ is disjoint to $\mathbb{V}$ and $\mathbb{P}$. Additionally, the term ordering must fulfil the requirement that all distinct object identifiers are smaller w.r.t. the term ordering than any other non-variable term, i.e. $\forall t \in (\mathbb{T} \setminus (\mathbb{V} \cup \mathbb{D})), \forall i \in \mathbb{D} : i \prec t$.

The introduction of the unique name assumption calls for changes of the calculus to cope with the following two new types of formulæ:

1. $\mathcal{A} \leftarrow i \simeq j, \mathcal{B}$, called *object tautology clause*,
2. $i \simeq j \leftarrow$, called *unit contradiction*,

for $i, j \in \mathbb{D}, i \neq j$ and $\mathcal{A}$ a possibly empty set of head literals and $\mathcal{B}$ a possibly empty set of body literals. These to types of formulæ are defined in Definitions 9 and 10.

Definition 9 (Object Tautology Clause). *Let $\mathcal{A}$ be a possibly empty set of head literals, $\mathcal{B}$ be a possibly empty set of body literals and $i, j \in \mathbb{D}$ with $i \neq j$. A clause $\mathcal{D}$ is an* object tautology clause *iff it is of form $\mathcal{A} \leftarrow i \simeq j, \mathcal{B}$.* ■

Definition 10 (Unit Contradiction). *Let $i, j \in \mathbb{D}$ with $i \neq j$. A clause $\mathcal{D}$ is a* unit contradiction *iff it is of form $i \simeq j \leftarrow$.* ■

$$\mathsf{unit\text{-}cont\text{-}right}(\sigma)\ \frac{X \simeq Y \leftarrow}{\Box} \quad \text{if (1,2) holds}$$

$$\mathsf{Inc}\ \frac{\mathbf{B}}{\mathbf{B} \cdot \Box} \quad \text{if (3,4) holds}$$

Conditions:

1. $X, Y \in \mathbb{V} \cup \mathbb{D}$
2. $(X \simeq Y \leftarrow)\sigma = i \simeq j \leftarrow$, where $i, j \in \mathbb{D}$ and $i \neq j$
3. there is a $C \in \mathbf{B}$
4. there is a σ such that $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \Box$

Figure 5: The unit-cont-right rule and the corresponding Inc extension rule for handling unit contradictions.

As the first type of formulæ, object tautology clauses, are trivially true, they cannot contribute in deriving a contradiction. For the sake of efficiency, we do not want to interact with these formulæ, i.e. we want to ignore them. The second type, unit contradictions, are inconsistent by definition. Thus a rule is needed to close a branch if one of its nodes is labelled with a unit contradiction. All the other type of formulæ that involve distinct object identifiers, are dealt with by the unmodified version of the calculus.

4.2 Extending the Calculus

The extension of the E-hyper tableau calculus being introduced in this section is an adaptation of the work by Schulz and Bonacina, who introduced a way of handling distinct object identifiers in the superposition calculus in [23].

The first change concerns the handling of unit contradiction clauses. As they are a sign of an inconsistency in the clause set, a new inference and a new extension rule are needed to close a branch if it contains a unit contradiction. Figure 5 shows the newly introduced unit-cont-right (*unit contradiction right*) inference rule and the corresponding extension rule Inc (*Inconsistency*).

In the second step, a way of dealing with object tautology clauses introduced. As an object tautology clause cannot contribute in closing a branch, it is useless for the refutation attempt and should not be used as a constituent clause for any of the calculus' extension rules. This behaviour can be achieved by extending the concept of negligible clauses to contain object tautology clauses. Therefore Definition 3 is now extended to Definition 11.

Definition 11 (Negligible Clauses). *Let $\mathcal{D}$ be a clause and $\mathcal{S}$ a set of clauses. A clause $\mathcal{D}$ is negligible (w.r.t. a clause set $\mathcal{S}$) iff at least one of the following holds:*

- *$\mathcal{D}$ is redundant (w.r.t. $\mathcal{S}$).*
- *$\mathcal{D}$ is non-properly subsumed (w.r.t. $\mathcal{S}$).*
- *$\mathcal{D}$ is an object tautology clause.*

■

With this step, it is properly taken care of the object tautology clauses, as they are rewritten by the Del rule and therefore inefficient reasoning steps are prevented.

With these changes, some definitions need to be adapted. We start by extending Definition 4 on page 9 (saturation up to redundancy) to Definition 12 by adding that a set $\mathcal{S}$ is not saturated up to redundancy if the unit-cont-right is applicable to any clause in $\mathcal{S}$. Additionally it is amended that object tautology clauses are not considered as constituent clauses for inference rules.

Definition 12 (Saturation up to Redundancy). *A clause set $\mathcal{S}$ is* saturated up to redundancy *iff for all clauses $C \in \mathcal{S}$ such that C is neither an object tautology nor redundant w.r.t. $\mathcal{S}$ all of the following hold:*

1. *Every inference $C, \mathcal{D} \Rightarrow_{\textsf{sup-left}(\sigma)} \mathcal{E}$,*
   where $\mathcal{D}$ is a fresh variant of a positive unit clause from $\mathcal{S}$,
   such that neither $C\sigma$ nor $\mathcal{D}\sigma$ is an object tautology or redundant w.r.t. $\mathcal{S}$, is redundant w.r.t. $\mathcal{S}$.
2. *Every inference $C, \mathcal{D} \Rightarrow_{\textsf{unit-sup-right}(\sigma)} \mathcal{E}$,*
   where $\mathcal{D}$ is a fresh variant of a positive unit clause from $\mathcal{S}$,
   such that neither $C\sigma$ nor $\mathcal{D}\sigma$ is an object tautology or redundant w.r.t. $\mathcal{S}$, is redundant w.r.t. $\mathcal{S}$.
3. *Every inference $C \Rightarrow_{\textsf{ref}(\sigma)} \mathcal{E}$,*
   such that $C\sigma$ is neither an object tautology nor redundant w.r.t. $\mathcal{S}$, is redundant w.r.t. $\mathcal{S}$.
4. *Every inference $C \Rightarrow_{\textsf{split}(\sigma)} A_1 \leftarrow, \ldots, A_m \leftarrow$,*
   such that $C\pi$ is neither an object tautology nor redundant w.r.t. $\mathcal{S}$, is redundant w.r.t. $\mathcal{S}$.
5. *No inference $C \Rightarrow_{\textsf{unit-cont-right}(\sigma)} \square$ is applicable.*

■

In the next step, Definition 5 on page 9 E-hyper tableau derivation is extended to Definition 13 by including that the Inc rule can be used on an open branch of a tableau **T** to extend it.

Definition 13 (E-Hyper Tableau Derivation). *An* E-hyper tableau derivation *of a set $\{C_1, \ldots, C_n\}$ of Σ-clauses is a possibly infinite sequence of tableaux $\mathbf{D} = (\mathbf{T}_i)_{0 \leq i < \kappa}$ such that*

1. *$\mathbf{T}_0$ is the clausal tableau over Σ that consists of a single branch of length n with the tableau clauses $C_1, \ldots, C_n$, and*
2. *for all $i > 0$, $\mathbf{T}_i$ is obtained from $\mathbf{T}_{i-1}$ by a single application of the Equality, Split, Del, Simp or Inc rule to an open branch in $\mathbf{T}_i$.*

■

The last definition that needs to be extended is the Definition 8 on page 13 (exhausted branch), which is extended to Definition 14 by adding that a branch is not exhausted if the Inc rule can be applied to a clause in this branch. Additionally, it is amended that object tautology clauses are not considered as constituent clauses for inference rules.

Definition 14 (Exhausted Branch). *Let $\mathbf{t}_\infty$ be a limit tree, $\mathbf{B} = (\mathbf{N}_k)_{0 \leq k < \nu}$ be a branch in $\mathbf{t}_\infty$ and $\mathbf{B}^i$ and $\mathbf{B}^j$ initial segments of $\mathbf{B}$. The branch $\mathbf{B}$ is* exhausted *iff it does not contain the empty clause, and for every clause $C \in \mathbf{B}_\infty$ and every fresh variant $\mathcal{D}$ of every positive unit clause in $\mathbf{B}_\infty$ such that neither C nor $\mathcal{D}$ is an object tautology or redundant w.r.t. $\mathbf{B}_\infty$ all of the following hold, for all $i < \nu$ such that $C \in \mathbf{B}^i$ and $\mathcal{D}$ is a variant of a clause in $\mathbf{B}^i$:*

1. *if* Equality *is applicable to $\mathbf{B}^i$ with underlying inference*
   $C, \mathcal{D} \Rightarrow_{\textsf{sup-left}(\sigma)} \mathcal{E}$,
   and neither $C\sigma$ nor $\mathcal{D}\sigma$ is an object tautology or redundant w.r.t. $\mathbf{B}^i$,
   then there is a $j < \nu$ such that the inference
   $C, \mathcal{D} \Rightarrow_{\textsf{sup-left}(\sigma)} \mathcal{E}$ *is redundant w.r.t. $\mathbf{B}^j$.*
2. *if* Equality *is applicable to $\mathbf{B}^i$ with underlying inference*
   $C, \mathcal{D} \Rightarrow_{\textsf{unit-sup-right}(\sigma)} \mathcal{E}$,
   and neither $C\sigma$ nor $\mathcal{D}\sigma$ is an object tautology or redundant w.r.t. $\mathbf{B}^i$,
   then there is a $j < \nu$ such that the inference
   $C, \mathcal{D} \Rightarrow_{\textsf{unit-sup-right}(\sigma)} \mathcal{E}$ *is redundant w.r.t. $\mathbf{B}^j$.*
3. *if* Equality *is applicable to $\mathbf{B}^i$ with underlying inference*
   $C \Rightarrow_{\textsf{ref}(\sigma)} \mathcal{E}$
   and $C\sigma$ is neither an object tautology nor redundant w.r.t. $\mathbf{B}^i$,
   then there is a $j < \nu$ such that the inference
   $C \Rightarrow_{\textsf{ref}(\sigma)} \mathcal{E}$ *is redundant w.r.t. $\mathbf{B}^j$.*
4. *if* Split *is applicable to $\mathbf{B}^i$ with underlying inference*
   $C \Rightarrow_{\textsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$
   and $C\pi$ is neither an object tautology nor redundant w.r.t. $\mathbf{B}^i$,
   then there is a $j < \nu$ such that the inference
   $C \Rightarrow_{\textsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ *is redundant w.r.t. $\mathbf{B}^j$.*
5. Inc *is not applicable to $\mathbf{B}^i$ with underlying inference*
   $C \Rightarrow_{\textsf{unit-cont-right}(\sigma)} \Box$.

■

These introduced changes allow the E-hyper tableau calculus to deal with distinct object identifiers in a complete and sound way. Before the completeness and soundness are formally proven in section 5 a derivation example is given to illustrate the introduced changes and their function.

Example 4 (Derivation with the modified E-hyper tableau calculus). Figure 6 shows the tableaux $\mathbf{T}_0$ and $\mathbf{T}_5$ of an E-hyper tableau derivation with distinct object identifiers. The intermediate tableaux $\mathbf{T}_1$ to $\mathbf{T}_4$ are not shown in the figure but are introduced textually.

$\mathbf{T}_0$ is the initial tableau of the derivation and contains three clauses. As there are two distinct objects identifiers i, j as terms in the set of clauses, it makes sense to use the modified version of the calculus. It is easy to see that $p(x) \simeq \mathtt{t}, q(f(x)) \simeq \mathtt{t} \leftarrow i \simeq j, r(g(a)) \simeq \mathtt{t}$ is an object tautology clause, as it contains $i \simeq j$ on the right side and thus the clause can be rewritten to $\mathtt{t} \simeq \mathtt{t}$ by an application of the Del rule. This leads to the tableau $\mathbf{T}_1$.

For the sake of simplicity the following steps are given as a list of underlying inference rules instead of showing all the tableaux.

$$\begin{tikzcd}[column sep=small]
 & \mathbf{T}_0 & \\
 & f(a) \simeq i \leftarrow \arrow[d, dash] & \\
 & p(x) \simeq \mathtt{t}, q(f(x)) \simeq \mathtt{t} \leftarrow i \simeq j, r(g(a)) \simeq \mathtt{t} \arrow[d, dash] & \\
 & i \simeq j, r(f(x)) \simeq t \leftarrow f(x) \simeq i &
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=small]
 & \mathbf{T}_5 & \\
 & f(a) \simeq i \leftarrow \arrow[d, dash] & \\
 & \mathtt{t} \simeq \mathtt{t} \leftarrow \arrow[d, dash] & \\
 & i \simeq j, r(f(x)) \simeq \mathtt{t} \leftarrow f(x) \simeq i \arrow[d, dash] & \\
 & i \simeq j, r(f(a)) \simeq \mathtt{t} \leftarrow i \simeq i \arrow[d, dash] & \\
 & i \simeq j, r(f(a)) \simeq \mathtt{t} \leftarrow \arrow[dl, dash] \arrow[dr, dash] & \\
i \simeq j \leftarrow \arrow[d, dash] & & r(f(a)) \simeq \mathtt{t} \leftarrow \\
\Box & &
\end{tikzcd}$$

Figure 6: The tableaux $\mathbf{T}_0$ and $\mathbf{T}_5$ of an E-hyper tableau derivation with DOI.

1. $\mathcal{C}, \mathcal{D} \Rightarrow_{\textsf{sup-left}(\sigma)} \mathcal{E}$, with
   $\mathcal{C} = i \simeq j, r(f(x)) \simeq t \leftarrow f(x) \simeq i$
   $\mathcal{D} = f(a) \simeq i \leftarrow$
   $\sigma = \{x/a\}$
   $\mathcal{E} = i \simeq j, r(f(a)) \simeq \mathtt{t} \leftarrow i \simeq i$
2. $\mathcal{C} \Rightarrow_{\textsf{ref}(\sigma)} \mathcal{E}$, with
   $\mathcal{C} = i \simeq j, r(f(a)) \simeq \mathtt{t} \leftarrow i \simeq i$
   $\sigma = \{\}$
   $\mathcal{E} = i \simeq j, r(f(a)) \simeq \mathtt{t} \leftarrow$
3. $\mathcal{C} \Rightarrow_{\textsf{split}(\pi)} A_1\pi \leftarrow, A_2\pi \leftarrow$, with
   $\mathcal{C} = i \simeq j, r(f(a)) \simeq \mathtt{t} \leftarrow$
   $\pi = \{\}$
   $A_1\pi = i \simeq j \leftarrow$
   $A_2\pi = r(f(a)) \simeq \mathtt{t} \leftarrow$
4. $\mathcal{C} \Rightarrow_{\textsf{unit-cont-right}(\sigma)} \Box$, with
   $\mathcal{C} = i \simeq j \leftarrow$
   $\sigma = \{\}$

These four steps lead from $\mathbf{T}_1$ to $\mathbf{T}_5$. The application of the Del rule is independent and must not have happened as the first step but could have been somewhere between $\mathbf{T}_1$ and $\mathbf{T}_5$. The ordering of the other extension steps is fixed, as they need to be applied consecutively.

$\Box$

This two changes enable the E-hyper tableau calculus to natively handle distinct object identifiers. The extended calculus is sound and complete, which is proven in the following Section 5.

5 Properties

5.1 Overview

This section shows that the modified E-hyper tableau calculus is still sound and complete. The basis for the proofs is taken from [9] and adapted accordingly.

There are four main parts in this section. In the first part the concept of term rewriting systems is introduced, which are used as a means to construct a model for a set of clauses.

The details how to construct a model for a given set of clauses are given in the second part (Proposition 1). It leads to the result that the shown procedure is complete, i.e. if a clause set has a model the method can derive it (Theorem 1).

So far, the actual E-hyper tableau calculus has not been involved, as only sets of clauses were considered. In part three, we show that if the set of persistent clauses has a model, the set of clauses of the initial tableau is satisfiable, i.e. the E-hyper tableau calculus is complete (Theorem 2). This is done by showing that the set of persistent clauses for an exhausted branch is saturated up to redundancy (Proposition 2). This enables us to use the result of the previous part, i.e. a rewriting system can be a model for a set of clauses.

The fourth and last part is a straight forward proof of the calculus' soundness (Theorem 3).

5.2 Rewrite Systems

This section gives a rough introduction on rewrite systems, as only concepts needed for the following proofs are mentioned. More details on term rewriting can be found in [4,16,12].

Given is a logic $\mathfrak{L}$ with the signature Σ. A *rewrite system* is a possibly infinite set of rewrite rules, where a *rewrite rule* is an expression of the form $l \rightsquigarrow r$ with l and r Σ-terms. If a rewrite system only works on ground terms it is called a *ground rewrite system*. A ground rewrite system R with $l > r$, for every rule $l \rightsquigarrow r \in \mathsf{R}$ is called *ordered rewrite system*. We assume $>$ to be the term ordering introduced in section 2.2.

If the rewrite system does not contain two different rules of the forms $l \rightsquigarrow r$ and $s[l] \rightsquigarrow t$, i.e. no left hand side of a rule can be rewritten by another rule, it is *lhs-irreducible*.

A term rewriting system is called *confluent* iff all of its terms are confluent. A term s is confluent iff the following holds: If there are rewriting rules $s \rightsquigarrow^* u$ and $s \rightsquigarrow^* v$ then there are rules such that $u \rightsquigarrow^* t$ and $v \rightsquigarrow^* t$, i.e. different ways of rewriting s finally yield the same result. This property is also called *Church-Rosser*.

If an ordered ground rewrite system is lhs-irreducible it is a *convergent* ground rewrite system, i.e. it is confluent and does terminate. For two given Σ-terms s and t and a convergent rewrite system R, $\mathsf{R} \models_E s \simeq t$ holds iff there is exactly one Σ-term u such that $s \rightsquigarrow^*_{\mathsf{R}} u$ and $t \rightsquigarrow^*_{\mathsf{R}} u$, where $\mathsf{R} \models_E \mathcal{S}$ denotes that the interpretation $\{l \simeq r \mid l \rightsquigarrow r \in \mathsf{R}\}$ satisfies $\mathcal{S}$.

If $s \rightsquigarrow^*_{\mathsf{R}} u$ and $t \rightsquigarrow^*_{\mathsf{R}} u$, s and t are *joinable* by R.

If $s \rightsquigarrow^*_{\mathsf{R}} t$ and there is no rewrite rule with left hand side t, t is called the *(R-)normal form* of s.

R and its variations will always denote a ground lhs-irreducible rewrite system in the following section.

One way of proving that is often used in this section is well-founded induction on the ordering of the terms. As the base cases are obvious in the most cases, they are not shown.

5.3 Model Construction

This section shows how a given set of clauses $\mathcal{S}$ induces a ground lhs-irreducible rewrite system $\mathrm{R}_{\mathcal{S}}$ and how the rewrite system can be used to show the satisfiability of $\mathcal{S}$. We assume that $\mathcal{S}$ does not contain unit contradictions. It is safe to make this assumption for our purpose, as later on the set of persistent clauses will be the set we are examining and by definition of an exhausted branch, this set must not contain unit contradictions. The main idea is to interpret a ground positive unit clause as a rewrite rule, i.e. $l \simeq r$ and $r \simeq l$ can be seen as the rewrite rule $l \rightsquigarrow r$ if $l > r$ or as $r \rightsquigarrow l$ if $r > l$.

The first step is to define how the rewrite system and thus the potential model is constructed (see Definition 15). Then we introduce and prove Lemma 1 to show that if a rewrite system is a model for a clause then the extension of this rewrite system is still a model for this clause. Lemma 2 then claims the fact that the rewrite system for a clause is an extension of the rewrite system for a smaller (w.r.t. $>$) clause. Proposition 1 claims the model construction abilities of the introduced method. It is needed in the proof of Theorem 1, which states that if a clause set is saturated up to redundancy and does not contain the empty clause, this clause set is satisfiable.

The rewrite system $\mathrm{R}_{\mathcal{S}}$ for $\mathcal{S}$ can be developed by using induction on the term ordering to create the rewrite rules for all clauses of $\mathcal{S}$ starting with the smallest one. Each clause $C \in \mathcal{S}$ has two sets of rewrite rules associated with it, where R_C contains the rewrite rules of all clauses that are smaller than C and ϵ_C contains the rewrite rule for C itself if it is a positive unit or is empty otherwise. Therefore $\mathrm{R}_{\mathcal{S}}$ can be derived by joining all R_C for all ground Σ-clauses C of $\mathcal{S}$.

Before a more formal definition is given in Definition 15, the notion $\bigcup_{C>\mathcal{D}} \epsilon_{\mathcal{D}}$ is introduced that denotes the union of $\epsilon_{\mathcal{D}}$ for all clauses $\mathcal{D}$ such that $C > \mathcal{D}$ holds.

Definition 15 (Rewrite System Construction). *Let $\mathcal{S}$ be a set of clauses and $C \in \mathcal{S}$ a clause.*

– $\epsilon_C = \begin{cases} \{l \rightsquigarrow r\} & \text{if } C = l \simeq r \leftarrow \text{ is a ground instance of a positive unit in } \mathcal{S}, \\ & l > r, \text{ and } l \text{ is irreducible w.r.t. } \mathrm{R}_C \\ \emptyset & \text{otherwise} \end{cases}$

– $\mathrm{R}_C = \bigcup_{C>\mathcal{D}} \epsilon_{\mathcal{D}}$.

– $\mathrm{R}_{\mathcal{S}} = \bigcup_{C\in\mathcal{S}} \epsilon_C$

■

For future use, it is useful to know that a superset of a rewrite system satisfies at least all the clauses that are satisfied by the original rewrite system. This property is formalized in Lemma 1 and its validity is shown by the following proof.

Lemma 1. *Let $\mathcal{S}$ be a clause set, $C = \mathcal{A} \leftarrow \mathcal{B} \in \mathcal{S}$ a ground clause and R and R' rewrite systems such that $\mathrm{R}_C \subseteq \mathrm{R} \subseteq \mathrm{R}' \subseteq \mathrm{R}_{\mathcal{S}}$ holds. If $\mathrm{R} \models_E C$ then $\mathrm{R}' \models_E C$.* ■

Proof. Suppose $\mathrm{R} \models_E C$ holds. Let A be a head literal of $\mathcal{A}$.

We now prove that if $\mathrm{R} \models_E \mathcal{A} \leftarrow \mathcal{B}$ holds $\mathrm{R}' \models_E \mathcal{A} \leftarrow \mathcal{B}$ holds, as well. The main idea is to examine two different cases, where in the first we assume $\mathrm{R} \models_E \mathcal{B}$ and in the second $\mathrm{R} \not\models_E \mathcal{B}$.

Case 1 ($\mathrm{R} \models_E \mathcal{B}$) Suppose $\mathrm{R} \models_E \mathcal{B}$. With $\mathrm{R} \models_E \mathcal{B}$ and $\mathrm{R} \models_E \mathcal{A} \leftarrow \mathcal{B}$, $\mathrm{R} \models_E A$ must hold. As first-order logic with equality is monotonous and $\mathrm{R} \subseteq \mathrm{R}'$, $\mathrm{R}' \models_E A$ holds and thus $\mathrm{R}' \models_E \mathcal{A} \leftarrow \mathcal{B}$.

Case 2 ($\mathrm{R} \not\models_E \mathcal{B}$) Suppose $\mathrm{R} \not\models_E \mathcal{B}$. This case is proven by contradiction. Therefore we assume $\mathrm{R}' \models_E \mathcal{B}$ and $\mathrm{R}' \not\models_E A$ for any A in $\mathcal{A}$, which leads to $\mathrm{R}' \not\models_E \mathcal{A} \leftarrow \mathcal{B}$. For $\mathrm{R}' \models_E \mathcal{B}$ and $\mathrm{R} \not\models_E \mathcal{B}$ there must be a $B = s \simeq t$ in $\mathcal{B}$ such that $\mathrm{R}' \models_E B$ but $\mathrm{R} \not\models_E B$. In other words $s \simeq t$ is joinable by R' but not by R, i.e. there are rewrite rules in $(\mathrm{R}' \setminus \mathrm{R})$ to rewrite s and t to a common normal form u.

Every rule $l \rightsquigarrow r \in \mathrm{R}_{\mathcal{S}}$ is obtained from a ground instance of $l \simeq r \leftarrow \in \mathcal{S}$. As we assume $l \rightsquigarrow r \in (\mathrm{R}' \setminus \mathrm{R})$ and $\mathrm{R}_C \subseteq \mathrm{R}$ it follows $l \rightsquigarrow r \notin \mathrm{R}_C$. By definition of R_C (see Definition 15) for all clauses $\mathcal{D}$ that are not in $\mathrm{R}_{\mathcal{S}}$, $\mathcal{D} \geq C$ holds. As C looks like $A_1, \ldots, A_n \leftarrow s \simeq t, B_1, \ldots, B_m$, it entails $C \neq l \simeq r \leftarrow$ and thus $l \simeq r \leftarrow > C$ holds.

By the definition of the reduction ordering (see Definition 2.2) it follows that $l \rightsquigarrow r$ cannot be used for rewriting s or t. As no rule in $(\mathrm{R}' \setminus \mathrm{R})$ can be used to rewrite s and t the R'- and R-normal forms of s and t are the same and therefore $\mathrm{R}' \models_E s \simeq t$ and $\mathrm{R} \models_E s \simeq t$ holds.

As this is a contradiction to our assumption that $\mathrm{R}' \models_E \mathcal{B}$ holds but $\mathrm{R}' \models_E A$ does not, this entails $\mathrm{R}' \not\models_E \mathcal{B}$ or $\mathrm{R}' \models_E A$. Thus $\mathrm{R}' \models_E \mathcal{A} \leftarrow \mathcal{B}$. □

At this point it is useful to state that the rewrite rules of a clause $\mathcal{D}$ are a subset of the rewrite rules for a clause C that is larger than $\mathcal{D}$ according to the term ordering. This property is formalized in Lemma 2 and its validity is shown by the following proof, which is basically the application of the rewrite system construction rules (see Definition 15).

Lemma 2. *Let $\mathcal{S}$ be a clause set and $C, \mathcal{D} \in \mathcal{S}$ be ground. If $C > \mathcal{D}$ then* $\mathrm{R}_{\mathcal{D}} \cup \epsilon_{\mathcal{D}} \subseteq \mathrm{R}_C$. ■

Proof. Suppose $C, \mathcal{D} \in \mathcal{S}$ be ground and $C > \mathcal{D}$. By definition of the construction procedure for the rewrite system (see Definition 15) $\mathrm{R}_C = \bigcup_{C > \mathcal{E}} \epsilon_{\mathcal{E}}$ and $\mathrm{R}_{\mathcal{D}} = \bigcup_{\mathcal{D} > \mathcal{E}} \epsilon_{\mathcal{E}}$.

With $C > \mathcal{D}$ it follows $\epsilon_{\mathcal{D}} \subseteq \mathrm{R}_C$ and $\mathrm{R}_{\mathcal{D}} \subseteq \mathrm{R}_C$, which in combination entails $\mathrm{R}_{\mathcal{D}} \cup \epsilon_{\mathcal{D}} \subseteq \mathrm{R}_C$. □

The following proposition (see Proposition 1) is the core element of this part. It states that $\mathrm{R}_C \cup \epsilon_C \models_E C$. Additionally, it states that a clause C is either redundant to a set of clauses $\mathcal{S}$ and R_C already satisfies C or else when C is not redundant w.r.t. $\mathcal{S}$, extension of R_C by ϵ_C will satisfy C.

Before we can introduce the proposition we need to introduce the notion $\mathcal{S}_C$ that is defined as follows: For a clause set $\mathcal{S}$, a clause $C \in \mathcal{S}$ and a grounding substitution γ, $\mathcal{S}_C$ is the set of all ground clauses that are smaller than C, i.e. $\mathcal{S}_C = \{\mathcal{D} \in \mathcal{S} \mid C > \mathcal{D}\gamma\}$.

As the proposition has two cases, the proof is done in two cases. The first one is just a straight forward approach by using induction on the ordering of the clauses. The second case takes different forms of clauses into account and either shows by contradiction that this kind of clause cannot appear, or that the property holds.

Proposition 1 (Model Construction). *Let $\mathcal{S}$ be a clause set that is saturated up to redundancy and $\square \notin \mathcal{S}$. Then for every ground instance C of every clause from $\mathcal{S}$ the following holds:*

1. *If $\mathcal{S}_C \models_E C$ then $\epsilon_C = \emptyset$ and $\mathrm{R}_C \models_E C$*
2. *If $\mathcal{S}_C \not\models_E C$ then $\mathrm{R}_C \cup \epsilon_C \models_E C$*

■

Proof. The proposition's two cases are proven separately by combining well-founded induction on the ground instances of $\mathcal{S}$ and contradiction. For the induction we choose a ground clause $C \in \mathcal{S}$ and assume the proposition holds for all ground instances $\mathcal{D} \in \mathcal{S}$ with $C \succ \mathcal{D}$.

Case 1 ($\mathcal{S}_C \models_E C$) Suppose $\mathcal{S}_C \models_E C$. By combining the conclusions of both of the proposition's cases and using well-founded induction on the term ordering $\mathrm{R}_{\mathcal{D}} \cup \epsilon_{\mathcal{D}} \models_E \mathcal{D}$ can be concluded for every clause $\mathcal{D} \in \mathcal{S}_C$ with $C \succ \mathcal{D}$. With Lemma 2 $\mathrm{R}_{\mathcal{D}} \cup \epsilon_{\mathcal{D}} \subseteq \mathrm{R}_C$ holds, which leads to $\mathrm{R}_C \models_E \mathcal{D}$ by using Lemma 1. As $\mathrm{R}_C \models_E \mathcal{D}$ holds for all $\mathcal{D} \in \mathcal{S}_C$, $\mathrm{R}_C \models_E \mathcal{S}_C$ holds and with the premisses of this case $\mathcal{S}_C \models_E C$, $\mathrm{R}_C \models_E C$ holds as desired.

As it is proven that $\mathcal{S}_C \models_E C$ holds, it remains to be shown that $\epsilon_C = \emptyset$. This is done by trying to derive a contradiction. Therefore we assume $\epsilon_C = \{l \rightsquigarrow r\}$ with $C = l \simeq r \leftarrow$. For l and r to be equal in the E-interpretation induced by the convergent rewrite system R_C both terms must be joinable, i.e. they must have the same normal form w.r.t. R_C. Therefore there must be a rewrite rule in R_C that rewrites l. If l can be rewritten it is not irreducible, which is a contradiction to the definition for $\epsilon_C \neq \emptyset$ (see Definition 15).

Case 2 ($\mathcal{S}_C \not\models_E C$) Suppose $\mathcal{S}_C \not\models_E C$. This entails that C is not redundant w.r.t. $\mathcal{S}_C$ and therefore not redundant w.r.t. $\mathcal{S}$, and C a ground instance of a clause $\mathcal{E} \in \mathcal{S}$, i.e. $\mathcal{E}\gamma = C$ for a grounding substitution γ, $\mathcal{E}$ cannot be redundant w.r.t. $\mathcal{S}$. The proof of the proposition's second case is now done by analysing different structures for the clause C and then trying to show that this kind of clause cannot appear or to show that $\mathrm{R}_C \cup \epsilon_C \models_E C$ holds. For trying to derive a contradiction we use $\mathcal{S} \not\models_E \mathcal{E}$ and the definition of saturated up to redundancy (see Definition 12)

1. $C = (\mathcal{E}[x])\gamma$ and $x\gamma$ is reducible w.r.t. . R_C.
   Suppose $C = \mathcal{E}\gamma$, for some clause $\mathcal{E} \in \mathcal{S}$ and some (grounding) substitution γ and $\mathcal{E}$ contains a variable x, i.e. $\mathcal{E}[x]$. Suppose, as well that $x\gamma$ is reducible w.r.t. R_C. We show by contradiction that this case cannot appear.
   If $x\gamma$ is reducible, there must be a rule $l \rightsquigarrow r \in \mathrm{R}_C$ and l must occur in $x\gamma$, i.e. $x\gamma[l]$. We now assume a (grounding) substitution γ' that is similar to γ in such a way that both substitution are identical with the exception that where γ has l, γ' has r, i.e. the rewrite rule $l \rightsquigarrow r$ has been applied. Thus $x\gamma' = x\gamma[r]$. As only larger terms are rewritten, $l \succ r$ holds and therefore $\mathcal{E}\gamma \succ \mathcal{E}\gamma'$, which in combination with the induction hypothesis leads to $\mathrm{R}_{\mathcal{E}\gamma'} \cup \epsilon_{\mathcal{E}\gamma'} \models_E \mathrm{R}_{\mathcal{E}\gamma}$. From $\mathcal{E}\gamma \succ \mathcal{E}\gamma'$ and Lemma 2 follows $\mathrm{R}_{\mathcal{E}\gamma'} \cup \epsilon_{\mathcal{E}\gamma'} \subseteq \mathrm{R}_{\mathcal{E}\gamma}$, which with Lemma 1 leads to $\mathrm{R}_{\mathcal{E}\gamma} \models_E \mathcal{E}\gamma'$.
   Because of $l \rightsquigarrow r \in \mathrm{R}_C$, $\mathcal{E}\gamma = C$ and by definition of γ' conclude with congruence $\mathrm{R}_C \models_E C$, which is a contradiction to $\mathcal{S}_C \not\models_E C$.

2. $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$ and $s\gamma = t\gamma$.
   Suppose $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$ for some clause $\mathcal{A} \leftarrow s \simeq t, \mathcal{B} \in \mathcal{S}$ and some grounding substitution γ and $s\gamma = t\gamma$ holds. We show by contradiction that this case cannot appear.
   If $s\gamma = t\gamma$ there is an inference $\mathcal{A} \leftarrow s \simeq t, \mathcal{B} \Rightarrow_{\mathsf{ref}(\sigma)} (\mathcal{A} \leftarrow \mathcal{B})\sigma$ with σ being the mgu of s and t and part of the grounding substitution γ, i.e. $\gamma = \sigma\delta$ for a substitution δ. By the definition of saturation up to redundancy (see Definition 12) and the proposition's premiss that $\mathcal{S}$ is saturated up to redundancy such an inference is redundant. Therefore the clause $(\mathcal{A} \leftarrow \mathcal{B})\sigma$ is redundant w.r.t. $\mathcal{S}_C$, i.e. $\mathcal{S}_C \models_E (\mathcal{A} \leftarrow \mathcal{B})\sigma$, which trivially entails $\mathcal{S}_C \models_E \mathcal{A} \leftarrow s \simeq t, \mathcal{B}$. As $C = \mathcal{A} \leftarrow s \simeq t, \mathcal{B}$ this is a contradiction to $\mathcal{S}_C \not\models_E C$.
3. $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$ with $(s \simeq t)\gamma = i \simeq j$, with $i, j \in \mathbb{D}$ and $i \neq j$.
   Suppose $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$ for some clause $\mathcal{A} \leftarrow s \simeq t, \mathcal{B} \in \mathcal{S}$ and some grounding substitution γ and $(s \simeq t)\gamma = i \simeq j$ holds with i and j are two non-identical distinct object identifiers. We show by contradiction that this case cannot appear.
   As i and j are two non-identical members of $\mathbb{D}$ and the unique name assumption applies to $\mathbb{D}$ the equation $i \simeq j$ can never be true. With the false literal $i \simeq j$ in the body of the clause the whole clause becomes true. From this follows that C can be written as $\mathtt{t} \simeq \mathtt{t} \leftarrow$, i.e. $C = \mathtt{t} \simeq \mathtt{t} \leftarrow$.
   As $\mathcal{S}_C \models_E \mathtt{t} \simeq \mathtt{t} \leftarrow$ holds trivially it is a contradiction to $\mathcal{S}_C \not\models_E C$.
4. $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$ and $s\gamma \succ t\gamma$ and $s\gamma$ is irreducible w.r.t. R_C.
   Suppose $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$ for some clause $\mathcal{A} \leftarrow s \simeq t, \mathcal{B} \in \mathcal{S}$ and some grounding substitution γ and $s\gamma \neq t\gamma$ holds. Furthermore assume that $s\gamma$ is the larger side of the equation $(s \simeq t)\gamma$, i.e. $s\gamma \succ t\gamma$ and that $s\gamma$ is irreducible w.r.t. R_C.
   As $s\gamma$ is irreducible w.r.t. R_C and therefore $s\gamma$ and $t\gamma$ are not joinable w.r.t. R_C, $\mathrm{R}_C \not\models_E s\gamma \simeq t\gamma$ holds. With $s\gamma \simeq t\gamma$ in the body of the clause C this trivially entails $\mathrm{R}_C \models_E C$, which is one part that was to be shown.
   The second part of the first cases' conclusion holds trivially, as C is not a positive unit and by definition of the rewrite system (see Definition 15) $\epsilon_C = \emptyset$ holds.
5. $C = (s \simeq t \leftarrow)\gamma$ with $(s \simeq t \leftarrow)\gamma = i \simeq j \leftarrow$, $i, j \in \mathbb{D}$ and $i \neq j$.
   Suppose $C = (s \simeq t \leftarrow)\gamma$ for some positive unit clause $(s \simeq t \leftarrow) \in \mathcal{S}$ and some grounding substitution γ and $(s \simeq t)\gamma = i \simeq j$ holds, where i and j are two non-identical distinct object identifiers. We show by contradiction that this case cannot appear.
   If such a C exists the inference $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\gamma)} \square$ is applicable. This is clearly a contradiction to the preconditions of the model construction in Proposition 1, as $\mathcal{S}$ is required to be saturated up to redundancy and by case 4 of the definition of saturation up to redundancy 12 a set where unit-cont-right is applicable is not saturated up to redundancy.
6. $C = (s \simeq t \leftarrow)\gamma$ and $s\gamma \succ t\gamma$ and $s\gamma$ is irreducible w.r.t. R_C.
   Suppose $C = (s \simeq t \leftarrow)\gamma$ for some positive unit clause $(s \simeq t \leftarrow) \in \mathcal{S}$ and some grounding substitution γ and $s\gamma \neq t\gamma$. Furthermore assume that $s\gamma$ is the larger side of the equation $(s \simeq t)\gamma$, i.e. $s\gamma \succ t\gamma$ and that $s\gamma$ is irreducible w.r.t. R_C.
   Thus, by definition of the rewrite system (see Definition 15) $\epsilon_C = \{s \rightsquigarrow t\}$, which trivially entails $\mathrm{R}_C \cup \epsilon_C \models_E C$

7. $C = (A_1, \ldots, A_m \leftarrow)\gamma$ and $m \geq 2$.
   Suppose $C = \mathcal{E}\gamma$ for some positive non-unit clause $\mathcal{E} = (A_1, \ldots, A_m \leftarrow) \in \mathcal{S}$ and some grounding substitution γ and $m \geq 2$. Furthermore assume $\gamma = \pi\delta$ for a purifying substitution π and some (possibly empty) substitution δ. We show by contradiction that this case cannot appear.
   From the assumption of case 2 follows that C is not redundant w.r.t. $\mathcal{S}$, which entails that $\mathcal{E}$ is not redundant w.r.t. $\mathcal{S}$. By the definition of saturation up to redundancy (see Definition 12) and the requirement that $\mathcal{S}$ is saturated up to redundancy conclude that $\mathcal{E} \Rightarrow_{\mathsf{split}(\pi)} A_1\pi \leftarrow, \ldots, A_m\pi \leftarrow$ is redundant w.r.t. $\mathcal{S}$. This entails that in particular its ground instance $C \Rightarrow_{\mathsf{split}(\epsilon)} A_1\gamma \leftarrow, \ldots, A_m\gamma \leftarrow$ is redundant w.r.t. $\mathcal{S}$. By definition of redundancy (see Definition 1) $\mathcal{S}_C \models_E A_i\gamma$ holds for some i with $1 \leq i \leq m$, which entails $\mathcal{S}_C \models_E C$.
8. $C = (\mathcal{E}[s])\gamma$ and $s\gamma$ is reducible at a non-variable position.
   In this case we need to consider the clauses that do not fall into any of the previous cases. Therefore we analyse which kinds of formulæ are not yet treated, which leads to two cases:
   (a) $\mathcal{E} = \mathcal{A} \leftarrow s \simeq t, \mathcal{B}$ with $s\gamma \succ t\gamma$, $s\gamma$ is reducible w.r.t. R_C and γ is a grounding substitution.
   (b) $\mathcal{E} = s \simeq t \leftarrow$ with $s\gamma \succ t\gamma$, $s\gamma$ is reducible w.r.t. R_C and γ is a grounding substitution.

   As both cases can be treated in a similar way, we use a shared approach to show by contradiction that both kinds of clauses cannot appear.
   As $s\gamma$ is reducible by R_C there must be a rule $l \rightsquigarrow r \in \mathrm{R}_C$ that rewrites $s\gamma$. As case 2.1. already deals with the application of rewrite rules at a variable position we now assume that $s\gamma$ is not rewritten at or below a variable position. More formally $s\gamma[l]_p$ is a non-variable position of s for any p.
   By construction of the rewrite rules (see Definition 15), the rule $l \rightsquigarrow r$ is obtained from the ground instance of a positive unit clause in $\mathcal{S}$. Let $\mathcal{F} = l' \simeq r' \leftarrow$ be a fresh variant of the appropriate unit clause and assume that γ is extended in such way that $l'\gamma = l$ and $r'\gamma = r$.
   As $l \rightsquigarrow r \in \mathrm{R}_C$ and thus $l'\gamma \rightsquigarrow r'\gamma \in \mathrm{R}_C$ $C \succ \mathcal{F}\gamma$ must hold. As $C \succ \mathcal{F}\gamma$ and by the induction hypothesis the proposition holds for all clauses smaller (w.r.t. $\succ$) than C, it has to hold for $\mathcal{F}$.
   Considering the first case of the proposition, i.e. $\mathcal{S}_{\mathcal{F}\gamma} \models_E \mathcal{F}\gamma$, which requires $\epsilon_{\mathcal{F}\gamma} = \emptyset$ and thus $l\gamma \rightsquigarrow r\gamma \notin \mathrm{R}_{\mathcal{S}}$. Therefore this case is not possible and the second case of the proposition, i.e. $\mathcal{S}_{\mathcal{F}\gamma} \not\models_E \mathcal{F}\gamma$, stating that $\mathcal{F}\gamma$ is redundant w.r.t. $\mathcal{S}$, must hold.
   We now need to treat the two different kinds of formulæ separately for one step as follows:
   (a) For $\mathcal{E} = \mathcal{A} \leftarrow s \simeq t, \mathcal{B}$ consider the ground sup-left rule

$$(\mathcal{A}\gamma \leftarrow s\gamma[l'\gamma]_p \simeq t\gamma, \mathcal{B}\gamma), \mathcal{F}\gamma \Rightarrow_{\mathsf{sup\text{-}left}(\epsilon)} \mathcal{A}\gamma \leftarrow s\gamma[r'\gamma]_p \simeq t\gamma, \mathcal{B}\gamma \tag{1}$$

   Because p is a position of a non-variable term in s, say, l'', the sup-left inference

$$(\mathcal{A} \leftarrow s[l'']_p \simeq t, \mathcal{B}), \mathcal{F} \Rightarrow_{\mathsf{sup\text{-}left}(\sigma)} (\mathcal{A} \leftarrow s[r']_p \simeq t, \mathcal{B})\sigma \tag{2}$$

exists, where σ is a mgu of l' and l'' and $\gamma = \sigma\delta$ for some substitution δ. The ground sup-left inference (1) then is a ground inference of the sup-left inference (2).

(b) For $\mathcal{E} = s \simeq t \leftarrow$ consider the ground unit-sup-right rule

$$(s\gamma[l'\gamma]_p \simeq t\gamma \leftarrow), \mathcal{F}\gamma \Rightarrow_{\text{unit-sup-right}(\epsilon)} s\gamma[r'\gamma]_p \simeq t\gamma \leftarrow \quad (3)$$

Because p is a position of a non-variable term in s, say, l'', the unit-sup-right inference

$$(s[l'']_p \simeq t \leftarrow), \mathcal{F} \Rightarrow_{\text{unit-sup-right}(\sigma)} (s[r']_p \simeq t \leftarrow)\sigma \quad (4)$$

exists, where σ is a mgu of l' and l'' and $\gamma = \sigma\delta$ for some substitution δ. The ground unit-sup-right inference (3) then is a ground inference of the unit-sup-right inference (4).

As we concluded that $\mathcal{F}\gamma$ is not redundant w.r.t. $\mathcal{S}$, the more general clause $\mathcal{F}\sigma$ can neither be redundant w.r.t. $\mathcal{S}$.
By case 2 of the definition of saturation up to redundancy (see Definition 12) the inferences (2) and (4) are redundant w.r.t. $\mathcal{S}$ and thus their ground instances (1) and (3) are redundant w.r.t. $\mathcal{S}$, as well.
For the sake of brevity and to treat both cases a) and b) at once we introduce the new clause $\mathcal{G}$, which we assume to be the conclusion of the inference (1) or the inference (2). For the following it makes no difference if $\mathcal{G} = \mathcal{A}\gamma \leftarrow s\gamma[r'\gamma]_p \simeq t\gamma, \mathcal{B}\gamma$ or $\mathcal{G} = s\gamma[r'\gamma]_p \simeq t\gamma \leftarrow$, as it holds for both cases.
By definition of redundancy $\mathcal{S}_C \cup \{\mathcal{F}\}\gamma \models_E \mathcal{G}$. By induction over the ordering of the clauses and the combination of both conclusions of the proposition we derive $R_{\mathcal{H}} \cup re_{\mathcal{H}} \models_E \mathcal{H}$ for every $\mathcal{H} \in \mathcal{S}_C$. In combination with lemma 2 this leads to $R_{\mathcal{H}} \cup \epsilon_{\mathcal{H}} \subseteq R_C$, which with lemma 1 leads to $R_C \models_E \mathcal{H}$, for every clause $\mathcal{H} \in \mathcal{S}_C$. This is equivalent with $R_C \models_E \mathcal{S}_C$.
As $\mathcal{F} = (l' \simeq r')\gamma$ is present as a rewrite rule $l'\gamma \rightsquigarrow r'\gamma \in R_C$ thus $l \rightsquigarrow r \in R_C$, it follows trivially that $R_C \models_E \mathcal{F}\gamma$. In combination with $R_C \models_E \mathcal{S}_C$ and $\mathcal{S}_C \cup \{\mathcal{F}\}\gamma \models_E \mathcal{G}$ conclude $R_C \models_E \mathcal{G}$.
From $l \rightsquigarrow r \in R_C$ conclude by congruence $R_C \models_E C$, which in combination with $R_C \models_E \mathcal{S}_C$ is a contradiction to $\mathcal{S}_C \not\models_E C$.

□

The last step of this part concerns the static completeness (see Theorem 1), which claims that a clause set that is saturated up to redundancy and does not contain the empty clause, is E-satisfiable. To show this the set is E-satisfiable it suffices to prove that there is a model for this set. This is basically done straight forward by using Proposition 1 to show that there is a model for such a set.

Theorem 1 (Static Completeness). *Let $\mathcal{S}$ be a clause set saturated up to redundancy. If $\Box \notin \mathcal{S}$ then $\mathcal{S}$ is E-satisfiable.* ■

Proof. Suppose $\Box \notin \mathcal{S}$. For $\mathcal{S}$ to be E-satisfiable there must be an E-Model. We therefore show that $R_{\mathcal{S}}$ is an E-model for $\mathcal{S}$ by showing that $R_{\mathcal{S}} \models_E C\gamma$ for an arbitrary chosen clause $C \in \mathcal{S}$ and an arbitrary chosen grounding substitution γ. Proposition 1 leads to $R_{C\gamma} \cup \epsilon_{C\gamma} \models_E C\gamma$, which with Lemma 1 leads to $R_{\mathcal{S}} \models_E C\gamma$, what was to be shown. □

This concludes the second part of this section. So far we have only dealt with of sets of clauses and the E-hyper tableau calculus was not involved. In the following chapter we use the properties gathered so far in relation with the E-hyper tableau calculus rules and properties to prove its completeness.

5.4 Completeness

We can now use the results about static completeness to prove Theorem 2, which is the main contribution of this part and states that the modified version of the E-hyper tableau calculus is complete, i.e. if a fair derivation of a set of clauses is not a refutation, then the set of clauses is E-satisfiable.

To prove Theorem 2 by using Theorem 1 we need to have a set of clauses that is saturated up to redundancy. Thus Proposition 2 is introduced and proven, which states that the set of persistent clauses of an exhausted branch of a fair derivation is saturated up to redundancy.

Additionally, a couple of lemmas that are introduced now, are needed for the proves. The first one is Lemma 3, which states that if a clause C is satisfied by the union of the set of clauses of an initial segment of a branch and the set of clauses $\mathcal{S}$, then C is satisfied by the union of $\mathcal{S}$ and the set of persistent clauses of this branch, as well.

Lemma 3. *Let C_1 and C_2 be ground clauses, $\mathcal{S}$ a set of ground clauses, $\mathbf{D}$ a derivation, $\mathbf{t}_\infty$ the limit tree of $\mathbf{D}$ and $\mathbf{B}$ a branch of $\mathbf{t}_\infty$. Furthermore let $\mathbf{B}^j$ be the initial segment of $\mathbf{B}$ and $\mathbf{B}_\infty$ the set of persistent clauses for $\mathbf{B}$. If $(\mathbf{B}^j)_{C_1} \cup \mathcal{S} \models_E C_2$ for some $j < \nu$ then $(\mathbf{B}_\infty)_{C_1} \cup \mathcal{S} \models_E C_2$.* ■

Proof. To prove that $(\mathbf{B}_\infty)_{C_1} \cup \mathcal{S} \models_E C_2$ holds we use well-founded induction and assume that the lemma holds for all clauses C_1' with $C_1 \succ C_1'$.

If $(\mathbf{B}^j)_{C_1} \subseteq (\mathbf{B}_\infty)_{C_1}$ then the result follows from the monotonicity of first-order logic with equality. If $(\mathbf{B}^j)_{C_1} \nsubseteq (\mathbf{B}_\infty)_{C_1}$ we can use the compactness of first-order logic with equality to remove the clauses that are in $(\mathbf{B}_\infty)_{C_1}$ but not in $(\mathbf{B}^j)_{C_1}$. Thus we define $(\mathbf{B}^j)_{C_1}$ to be a finite subset of $(\mathbf{B}^j)_{C_1}$ for which the entailment in the lemma's premiss ($(\mathbf{B}^j)_{C_1} \cup \mathcal{S} \models_E C_2$) holds.

Let $\mathbf{B}' = (\mathbf{B}^j)_{C_1} \setminus (\mathbf{B}_\infty)_{C_1}$ be these clauses from $(\mathbf{B}^j)_{C_1}$ that are not an instance of any persisting clause in $(\mathbf{B}_\infty)_{C_1}$. We now choose a $C' \in \mathbf{B}'$ that by construction is a ground clause of $(\mathbf{B}^j)_{C_1}$ that is not in $(\mathbf{B}_\infty)_{C_1}$, i.e. $C' \in (\mathbf{B}^j)_{C_1}$ and $C' \notin (\mathbf{B}_\infty)_{C_1}$.

If $C' \in (\mathbf{B}^j)_{C_1}$ but $C' \notin (\mathbf{B}_\infty)_{C_1}$ the clause C' was removed from the clause set $\mathbf{B}_k$ by the application of the Del or Simp rule at a certain step $k < \kappa$. Therefore C'must be an object tautology clause, or non-properly subsumed or redundant. We now consider each possibility.

1. C is an object tautology clause
   Suppose C was removed from $\mathbf{B}_k$ because it was an object tautology clause, i.e. $C\sigma$ is like $\mathcal{A}, \leftarrow i_1 \simeq i_2, \mathcal{B}$ with $i_1, i_2 \in \mathbb{D}$ and $i_1 \neq i_2$.
   As i_1 and i_2 are two non-identical members of $\mathbb{D}$ and the unique name assumption applies to $\mathbb{D}$ the equation $i_1 \simeq i_2$ can never be true. With the false literal $i_1 \simeq i_2$ in the body of the clause the whole clause becomes true.
   As there can be no non-tautological clauses $\mathcal{D}$ such that *true* $\models_E \mathcal{D}$ the lemma holds trivially.

2. C is non-properly subsumed
   Suppose C was removed from $\mathbf{B}_k$ because it was non-properly subsumed by a clause $\mathcal{D} \in \mathbf{B}_k$. C must be a proper instance of $\mathcal{D}$, as by the derivation rules Equality and Split no derived clause set $\mathbf{B}_i$ can contain a clause and a variant of it. The converse relation to proper instantiation, called proper generalisation, is well founded. Thus, by induction on this ordering, there is a clause $\mathcal{D}'$ in $\mathbf{B}_\infty$ that non-properly subsumes C. As C' is an instance of C and C is an instance of $\mathcal{D}'$, C' is an instance of $\mathcal{D}'$. With $\mathcal{D}' \in \mathbf{B}_\infty$, C' is an instance of a persisting clause in $\mathbf{B}_\infty$, which is a contradiction to the construction of $\mathbf{B}'$, as $\mathbf{B}'$ contains these clauses of $(\mathbf{B}^j)_{C_1}$ that are not an instance of any persisting clause in $\mathbf{B}_\infty$. Therefore this case is impossible.
3. C is redundant
   Suppose C was removed from $\mathbf{B}_k$ because it and its instance C' was redundant w.r.t. a specific subset $\mathbf{B}''$ of the derived branch $\mathbf{B}_{k+1}$, where $\mathbf{B}''$ is the branch specified in the definition of the Del and Simp derivation rules. Because $\mathbf{B}'' \subseteq \mathbf{B}_{k+1}$ it follows that C' is redundant w.r.t. $\mathbf{B}_{k+1}$, i.e. $(\mathbf{B}_{k+1})_{C'} \models_E C'$. By monotonicity of first-order logic with equality $(\mathbf{B}_{k+1})_{C'} \cup \mathcal{S} \models_E C'$ holds.
   As $C' \in \mathbf{B}'$ and $\mathbf{B}' \subseteq (\mathbf{B}^j)_{C_1}$ it follows that $C' \prec C_1$. By induction then
   $$(\mathbf{B}_\infty)_{C'} \cup \mathcal{S} \models_E C'. \tag{5}$$
   $C' \prec C_1$ leads to $(\mathbf{B}_\infty)_{C'} \subseteq (\mathbf{B}_\infty)_{C_1}$, which in combination with (5) and the monotonicity of first-order logic with equality entails
   $$(\mathbf{B}_\infty)_{C_1} \cup \mathcal{S} \models_E C'. \tag{6}$$
   This entailment allows us to replace the clause C' in the premiss $(\mathbf{B}^j)_{C_1}$ by the stronger set $(\mathbf{B}_\infty)_{C_1} \cup \mathcal{S}$. That is from $(\mathbf{B}^j)_{C_1} \cup \mathcal{S} \models_E C_2$ and (6) follows
   $$((\mathbf{B}_\infty)_{C_1} \cup \mathcal{S}) \cup \left((\mathbf{B}^j)_{C_1} \setminus \{C'\}\right) \cup \mathcal{S} \models_E C_2. \tag{7}$$
   As this has to hold for all members of $\mathbf{B}'$, (7) can be extended to
   $$((\mathbf{B}_\infty)_{C_1} \cup \mathcal{S}) \cup \left((\mathbf{B}^j)_{C_1} \setminus \mathbf{B}'\right) \cup \mathcal{S} \models_E C_2. \tag{8}$$
   With the definition of $\mathbf{B}' = (\mathbf{B}^j)_{C_1} \setminus (\mathbf{B}_\infty)_{C_1}$, which implies $(\mathbf{B}^j)_{C_1} \setminus \mathbf{B}' \subseteq (\mathbf{B}_\infty)_{C_1}$ and (8) $(\mathbf{B}_\infty)_{C_1} \cup \mathcal{S} \models_E C_2$ follows immediately.

□

The result of Lemma 3 allows a straight forward proof of Lemma 4, which states that if a clause is redundant to the set of clauses for an initial segment of a branch it is redundant to the set of persistent clauses of this branch, as well.

Lemma 4. *Let C be a clause, $\mathbf{D}$ a derivation, $\mathbf{t}_\infty$ the limit tree of $\mathbf{D}$ and $\mathbf{B}$ a branch of $\mathbf{t}_\infty$. Furthermore let $\mathbf{B}^j$ be the initial segment of $\mathbf{B}$ and $\mathbf{B}_\infty$ the set of persistent clauses for $\mathbf{B}$. If C is redundant w.r.t. $\mathbf{B}^j$ for some $j < \nu$ then C is redundant w.r.t. $\mathbf{B}_\infty$.* ■

Proof. Suppose C is redundant w.r.t. $\mathbf{B}^j$ for some $j < \nu$. To show that C is redundant w.r.t. $\mathbf{B}_\infty$ it suffices to show that an arbitrary ground clause of C is redundant w.r.t. $\mathbf{B}_\infty$. Therefore let $\mathcal{D} = C\gamma$ for a grounding substitution γ. As C is redundant w.r.t. $\mathbf{B}^j$ its instance $\mathcal{D}$ is redundant w.r.t. $\mathbf{B}^j$, i.e. $(\mathbf{B}^j)_\mathcal{D} \models_E \mathcal{D}$. Lemma 3 leads to the conclusion $(\mathbf{B}_\infty)_\mathcal{D} \models_E \mathcal{D}$, i.e. $\mathcal{D}$ is redundant w.r.t. $\mathbf{B}_\infty$. □

For the proof of Proposition 2 four more lemmas are needed. The first three claim that if an application of the sup-left, unit-sup-right, ref and split rule is redundant w.r.t. the set of clauses of an initial segment of a branch, the application of these rules is redundant w.r.t. the set of persistent clauses for this branch.

The fourth lemma claims that if the unit-cont-right is not applicable to the set of clauses of an initial segment of a branch, it is not applicable to the set of persistent clauses.

Lemma 5 formalizes this statement for the sup-left and unit-sup-right inference rules. The according proof is straight forward by applying definitions and Lemma 3.

Lemma 5. *Let C be a clause, $\mathcal{D}$ be a positive unit clause,* $\mathbf{D}$ *a derivation,* $\mathbf{t}_\infty$ *the limit tree of* $\mathbf{D}$ *and* $\mathbf{B}$ *a branch of* $\mathbf{t}_\infty$*. Furthermore let* $\mathbf{B}^j$ *be the initial segment of* $\mathbf{B}$ *and* $\mathbf{B}_\infty$ *the set of persistent clauses for* $\mathbf{B}$*. Any inference $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$, where $\mathsf{R} \in \{$sup-left, unit-sup-right$\}$ that is redundant w.r.t.* $\mathbf{B}^j$*, for some $j < \nu$, is redundant w.r.t.* $\mathbf{B}_\infty$. ■

Proof. Suppose an inference $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$, where $\mathsf{R} \in \{$sup-left, unit-sup-right$\}$ is redundant w.r.t. $\mathbf{B}^j$, for some $j < \nu$. To show that $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$ is redundant w.r.t. to $\mathbf{B}_\infty$ it suffices to show that an arbitrary ground instance of the inference is redundant w.r.t. $\mathbf{B}_\infty$. Let γ be an arbitrary grounding substitution such that $\gamma = \sigma\delta$ and $C\gamma, \mathcal{D}\gamma \Rightarrow_{\mathsf{R}(\epsilon)} \mathcal{E}\delta$ is a ground instance of $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$, we show that $C\gamma, \mathcal{D}\gamma \Rightarrow_{\mathsf{R}(\epsilon)} \mathcal{E}\delta$ is redundant w.r.t. $\mathbf{B}_\infty$.

As $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}^j$ it follows trivially that $C\gamma, \mathcal{D}\gamma \Rightarrow_{\mathsf{R}(\epsilon)} \mathcal{E}\delta$ is redundant w.r.t. $\mathbf{B}^j$, i.e. $(\mathbf{B}^j)_{C\gamma} \cup \{\mathcal{D}\gamma\} \models_E \mathcal{E}\gamma$, which with Lemma 3 leads to $(\mathbf{B}^i)_{C\gamma} \cup \{\mathcal{D}\gamma\} \models_E \mathcal{E}\gamma$, i.e. $C\gamma, \mathcal{D}\gamma \Rightarrow_{\mathsf{R}(\epsilon)} \mathcal{E}\delta$ is redundant w.r.t. $\mathbf{B}_\infty$, which was to be shown. □

Lemma 6 formalizes this statement for the ref inference rules. The according proof is similar to that of Lemma 5.

Lemma 6. *Let C be a clause, $\mathcal{D}$ be a positive unit clause,* $\mathbf{D}$ *a derivation,* $\mathbf{t}_\infty$ *the limit tree of* $\mathbf{D}$ *and* $\mathbf{B}$ *a branch of* $\mathbf{t}_\infty$*. Furthermore let* $\mathbf{B}^j$ *be the initial segment of* $\mathbf{B}$ *and* $\mathbf{B}_\infty$ *the set of persistent clauses for* $\mathbf{B}$*. Any inference $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ that is redundant w.r.t.* $\mathbf{B}^j$*, for some $j < \nu$, is redundant w.r.t.* $\mathbf{B}_\infty$. ■

Proof. Suppose an inference $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}^j$, for some $j < \nu$. To show that $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ is redundant w.r.t. to $\mathbf{B}_\infty$ it suffices to show that an arbitrary ground instance of the inference is redundant w.r.t. $\mathbf{B}_\infty$. Let γ be an arbitrary grounding substitution such that $\gamma = \sigma\delta$ and $C\gamma \Rightarrow_{\mathsf{ref}(\epsilon)} \mathcal{E}\delta$ is a ground instance of $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ we show that $C\gamma \Rightarrow_{\mathsf{ref}(\epsilon)} \mathcal{E}\delta$ is redundant w.r.t. $\mathbf{B}_\infty$.

As $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}^j$ it follows trivially that $C\gamma \Rightarrow_{\mathsf{ref}(\epsilon)} \mathcal{E}\delta$ is redundant w.r.t. $\mathbf{B}^j$, i.e. $(\mathbf{B}^j)_{C\gamma} \cup \emptyset \models_E \mathcal{E}\gamma$, which with Lemma 3 leads to $(\mathbf{B}^i)_{C\gamma} \cup \emptyset \models_E \mathcal{E}\gamma$, i.e. $C\gamma \Rightarrow_{\mathsf{ref}(\epsilon)} \mathcal{E}\delta$ is redundant w.r.t. $\mathbf{B}_\infty$, which was to be shown. □

Lemma 7 takes care of the remaining rule of the original calculus, namely split. Again the proof is a straight forward application of definitions but this time in combination with Lemma 4.

Lemma 7. *Let C be a positive clause, π a purifying substitution for C, $\mathbf{D}$ a derivation, $\mathbf{t}_\infty$ the limit tree of $\mathbf{D}$ and $\mathbf{B}$ a branch of $\mathbf{t}_\infty$. Furthermore let $\mathbf{B}^j$ be the initial segment of $\mathbf{B}$ and $\mathbf{B}_\infty$ the set of persistent clauses for $\mathbf{B}$. If the inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant w.r.t. $\mathbf{B}^j$, for some $j < \nu$, then it is redundant w.r.t. $\mathbf{B}_\infty$.* ■

Proof. Suppose an inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant w.r.t. $\mathbf{B}^j$, for some $j < \nu$. To show that $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant w.r.t. to $\mathbf{B}_\infty$ it suffices to show that an arbitrary ground instance of the inference is redundant w.r.t. $\mathbf{B}_\infty$. Let γ be an arbitrary grounding substitution such that $\gamma = \pi\delta$ and $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is a ground instance of $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ we show that $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow \delta$ is redundant w.r.t. $\mathbf{B}_\infty$.

As $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant w.r.t. $\mathbf{B}^j$ it follows trivially that $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is redundant w.r.t. $\mathbf{B}^j$, i.e. $A_i\delta \leftarrow$ for some i with $1 \leq i \leq m$ is redundant w.r.t. $\mathbf{B}^j$. With Lemma 4 $A_i\delta \leftarrow$ is redundant w.r.t. $\mathbf{B}_\infty$, which entails that $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow \delta$ is redundant w.r.t. to $\mathbf{B}_\infty$. □

The last lemma needed to prove Proposition 2 is Lemma 8. The proof is done by contradiction and application of the appropriate definitions.

Lemma 8. *Let C be a positive unit, $\mathbf{D}$ a derivation, $\mathbf{t}_\infty$ the limit tree of $\mathbf{D}$ and $\mathbf{B}$ a branch of $\mathbf{t}_\infty$. Furthermore let $\mathbf{B}^j$ be the initial segment of $\mathbf{B}$ and $\mathbf{B}_\infty$ the set of persistent clauses for $\mathbf{B}$. If an inference $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \Box$ is not applicable to $\mathbf{B}^j$, for some $j < \nu$ and some substitution σ, then it is not applicable to $\mathbf{B}_\infty$.* ■

Proof. Suppose an inference $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \Box$ is not applicable to $\mathbf{B}^j$, for some $j < \nu$. To show that $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \Box$ is not applicable to $\mathbf{B}_\infty$ if it is not applicable to $\mathbf{B}^j$, we show by way of contradiction that it is not possible for $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \Box$ to be applicable to $\mathbf{B}_\infty$ but to be not applicable to $\mathbf{B}^j$.

For $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \Box$ to be applicable in $\mathbf{B}_\infty$ there must be a clause $C \in \mathbf{B}_\infty$ and a grounding substitution σ such that $C\sigma = i_1 \simeq i_2 \leftarrow$ with $i_1, i_2 \in \mathbb{D}$ and $i_1 \neq i_2$.

From the definition of the set of persistent clauses (see Definition 7) follows that $\mathbf{B}^j = \mathbf{B}_\infty \uplus \mathbf{B}$, where $\mathbf{B}$ is the set of clauses that have been rewritten in the derivation. By contradiction assume that $C \in \mathbf{B}$. That is C has been rewritten by the Del or Simp rule. It is easy to see that there is no possibility that C can be rewritten and thus $C \notin \mathbf{B}$. Thus $C \in \mathbf{B}^j$ must hold.

But if $C \in \mathbf{B}^j$, then $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \Box$ is applicable, which clearly is a contradiction. □

Now Proposition 2, which states that with a fair derivation the set of persistent clauses of an exhausted branch is saturated up to redundancy, is introduced and proven. The main idea of the proof is to show that all the requirements of the definition of saturation up to redundancy (see Definition 12) are fulfilled.

Proposition 2 (Exhausted branches are saturated up to redundancy). *If* $\mathbf{B}$ *is an exhausted branch of a limit tree of some fair derivation then* $\mathbf{B}_\infty$ *is saturated up to redundancy.* ■

Proof. Suppose $\mathbf{B}$ is an exhausted branch of a limit tree of some fair derivation. To show that $\mathbf{B}_\infty$ is saturated up to redundancy it suffices to choose an arbitrary clause $C \in \mathbf{B}_\infty$ that is not redundant w.r.t. and prove that the properties for saturation up to redundancy (see Definition 12) hold for C.

Before we take care of the four properties of saturation up to redundancy, notice that if there is branch $\mathbf{B}^j$ with $j < \nu$ and C is redundant w.r.t. $\mathbf{B}^j$ it follows from lemma 4 that C is redundant w.r.t. $\mathbf{B}_\infty$ and nothing remains to be shown.

Therefore suppose that C is not redundant w.r.t. $\mathbf{B}^j$, for all $j < \nu$.

1. $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \square$
   By Definition 14 a branch $\mathbf{B}$, where Inc is applicable with underlying inference $C \Rightarrow_{\mathsf{unit\text{-}cont\text{-}right}(\sigma)} \square$ is not exhausted. Therefore there must be no $C \in \mathbf{B}$ such that $C\sigma$ is like $i_1 \simeq i_2 \leftarrow$ with $i_1, i_2 \in \mathbb{D}$ and $i_1 \neq i_2$.
   From Lemma 8 it follows that if unit-cont-right is not applicable to $\mathbf{B}$ it is neither applicable to $\mathbf{B}_\infty$.
   Thus there is no clause in $\mathbf{B}_\infty$ such that the inference rule unit-cont-right is applicable, which concludes the fourth case of the definition of saturation up to redundancy (see Definition 12).
2. $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$, where $\mathsf{R} \in \{\mathsf{sup\text{-}left}, \mathsf{unit\text{-}sup\text{-}right}\}$
   Suppose there is an inference $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$, where $\mathsf{R} \in \{\mathsf{sup\text{-}left}, \mathsf{unit\text{-}sup\text{-}right}\}$, σ is a substitution and $\mathcal{D}$ is a fresh variant of a positive unit clause from $\mathbf{B}_\infty$.
   To show that $\mathbf{B}_\infty$ is saturated up to redundancy it suffices to show one of the following: $C\sigma$ is redundant w.r.t. $\mathbf{B}_\infty$, $\mathcal{D}\sigma$ is redundant w.r.t. $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}_\infty$.
   If there is a $j < \nu$ such that $C\sigma$ is redundant w.r.t. $\mathbf{B}^j$, then by Lemma 4 $C\sigma$ is redundant w.r.t. $\mathbf{B}_\infty$, which concludes this case. The same holds for $\mathcal{D}\sigma$.
   Hence we assume that neither $C\sigma$ nor $\mathcal{D}\sigma$ is redundant w.r.t. $\mathbf{B}^j$ for all $j < \nu$.
   To show that $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}_\infty$ it suffices to show that an arbitrary ground instance $C\gamma, \mathcal{D}\gamma \Rightarrow_{\mathsf{R}(\epsilon)} \mathcal{E}\delta$ of the inference $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$ with the grounding substitution $\gamma = \sigma\delta$ and some substitution δ is redundant w.r.t. to $\mathbf{B}_\infty$.
   As $C \in \mathbf{B}_\infty$ there must be an $i < \nu$ such that for all j with $i \leq j < \nu$, $C \in \mathbf{B}^j$. And, as $\mathcal{D}$ is a variant of a clause in $\mathbf{B}_\infty$, there must be an i' such that for all j' with $i' \leq j' < \nu$, $\mathcal{D}$ is a variant of a clause in $\mathbf{B}_{j'}$. Without loss of generality assume that $i \geq i'$ and thus $\mathcal{D}$ is a variant of a clause in $\mathbf{B}^j$ for all $i \leq j < \nu$.
   Under these conditions, the derivation rule $\mathsf{Equality}$ is applicable to $\mathbf{B}_i$ with underlying inference $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$ unless $\mathcal{E}$ is a variant of a clause in $\mathbf{B}_i$, which would entail that the inference is redundant w.r.t. $\mathbf{B}_i$ and conclude this proof.
   By assumption $C\sigma$ and $\mathcal{D}\sigma$ are not redundant w.r.t. $\mathbf{B}^j$ for every $j < \nu$. As $\mathbf{B}$ is an exhausted branch and the definition of exhausted branches (see Definition 14) states there is a $k < \nu$ such that the inference $C, \mathcal{D} \Rightarrow_{\mathsf{R}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}_k$ by Lemma 5 follows that this inference is redundant w.r.t. $\mathbf{B}_\infty$.
   This holds for its ground inference $C\gamma, \mathcal{D}\gamma \Rightarrow_{\mathsf{R}(\epsilon)} \mathcal{E}\delta$, as well.

3. $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$
   Suppose there is an inference $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$, where σ is a substitution.
   To show that $\mathbf{B}_\infty$ is saturated up to redundancy it suffices to show that $C\sigma$ is redundant w.r.t. $\mathbf{B}_\infty$ or $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}_\infty$.
   If there is a $j < \nu$ such that $C\sigma$ is redundant w.r.t. $\mathbf{B}^j$, then by Lemma 4 $C\sigma$ is redundant w.r.t. $\mathbf{B}_\infty$, which concludes this case.
   Hence we assume that $C\sigma$ is not redundant w.r.t. $\mathbf{B}^j$ for all $j < \nu$.
   To show that $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}_\infty$ it suffices to show that an arbitrary ground instance $C\gamma \Rightarrow_{\mathsf{ref}(\epsilon)} \mathcal{E}\delta$ of the inference $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ with the grounding substitution $\gamma = \sigma\delta$ and some substitution δ is redundant w.r.t. to $\mathbf{B}_\infty$.
   As $C \in \mathbf{B}_\infty$ there must be an $i < \nu$ such that for all j with $i \leq j < \nu$, $C \in \mathbf{B}^j$.
   Under these conditions, the derivation rule Equality is applicable to $\mathbf{B}_i$ with underlying inference $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ unless $\mathcal{E}$ is a variant of a clause in $\mathbf{B}_i$, which would entail that the inference is redundant w.r.t. $\mathbf{B}_i$ and conclude this proof.
   By assumption $C\sigma$ is not redundant w.r.t. $\mathbf{B}^j$ for every $j < \nu$. As $\mathbf{B}$ is an exhausted branch and the definition of exhausted branches (see Definition 14) states there is a $k < \nu$ such that the inference $C \Rightarrow_{\mathsf{ref}(\sigma)} \mathcal{E}$ is redundant w.r.t. $\mathbf{B}_k$ by Lemma 6 follows that this inference is redundant w.r.t. $\mathbf{B}_\infty$.
   This holds for its ground inference $C\gamma \Rightarrow_{\mathsf{ref}(\epsilon)} \mathcal{E}\delta$, as well.
4. $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow \ldots., A_m \leftarrow$
   Suppose there is an inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$, where π is a purifying substitution.
   To show that $\mathbf{B}_\infty$ is saturated up to redundancy it suffices to show that $C\pi$ is redundant w.r.t. $\mathbf{B}_\infty$ or $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow \ldots., A_m \leftarrow$ is redundant w.r.t. $\mathbf{B}_\infty$.
   If there is a $j < \nu$ such that $C\sigma$ is redundant w.r.t. $\mathbf{B}^j$ then by Lemma 4 $C\sigma$ is redundant w.r.t. $\mathbf{B}_\infty$, which concludes this case.
   Hence we assume that $C\sigma$ is not redundant w.r.t. $\mathbf{B}^j$ for all $j < \nu$.
   To show that $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow \ldots., A_m \leftarrow$ is redundant w.r.t. $\mathbf{B}_\infty$ it suffices to show that an arbitrary ground instance $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow \ldots., A_m\delta \leftarrow$ of the inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow \ldots., A_m \leftarrow$ with the grounding substitution $\gamma = \pi\delta$ and some substitution δ is redundant w.r.t. to $\mathbf{B}_\infty$.
   As $C \in \mathbf{B}_\infty$ there must be an $i < \nu$ such that for all j with $i \leq j < \nu$, $C \in \mathbf{B}^j$.
   Under these conditions, the derivation rule Split is applicable to $\mathbf{B}_i$ with underlying inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow \ldots., A_m \leftarrow$ unless $A_h \leftarrow$ for some $1 \leq h \leq m$ is a variant of a clause in $\mathbf{B}_i$, which would entail that the inference is redundant w.r.t. $\mathbf{B}_i$ and conclude this proof.
   By assumption $C\sigma$ is not redundant w.r.t. $\mathbf{B}^j$ for every $j < \nu$. As $\mathbf{B}$ is an exhausted branch and the definition of exhausted branches (see Definition 14) states there is a $k < \nu$ such that the inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow \ldots., A_m \leftarrow$ is redundant w.r.t. $\mathbf{B}_k$ by Lemma 7 follows that this inference is redundant w.r.t. $\mathbf{B}_\infty$.
   This holds for its ground inference $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow \ldots., A_m\delta \leftarrow$, as well.

□

For the proof of the following theorem an additional lemma is needed. Lemma 9 states that if a rewrite system entails a clause set and a clause is redundant w.r.t. this clause set, the rewrite system entails the clause. The proof is done straight forward.

Lemma 9. *If* $\mathrm{R} \models_E \mathcal{S}$ *and* C *is redundant w.r.t.* $\mathcal{S}$ *then* $\mathrm{R} \models_E C$. ■

Proof. Suppose $\mathrm{R} \models_E \mathcal{S}$ and C is redundant w.r.t. $\mathcal{S}$. To show $\mathrm{R} \models_E C$ it suffices to show that for an arbitrary ground clause $\mathcal{D}$ of C, i.e. $\mathcal{D} = C\gamma$ for a grounding substitution γ, $\mathrm{R} \models_E \mathcal{D}$ holds. As C is redundant w.r.t. $\mathcal{S}$ it follows trivially that $\mathcal{D}$ is redundant w.r.t. $\mathcal{S}$, i.e. $\mathcal{S}_{\mathcal{D}} \models_E \mathcal{D}$. With the assumption $\mathrm{R} \models_E \mathcal{S}$ and $\mathcal{S}_{\mathcal{D}} \subset \mathcal{S}$ we can conclude $\mathrm{R} \models_E \mathcal{D}$. □

Theorem 2 states the completeness of the modified E-hyper tableau calculus, i.e. if a fair derivation for a clause set is not a refutation, this clause set is satisfiable.

The proof is straight forward and mainly relies on the theorem of static completeness (see Theorem 1) and the proposition that exhausted branches are saturated up to redundancy.

Theorem 2 (Completeness of E-Hyper Tableaux). *Let* $\mathcal{S}$ *be a clause set and* $\mathbf{D}$ *a fair derivation of* $\mathcal{S}$. *If* $\mathbf{D}$ *is not a refutation the set* $\mathcal{S}$ *is satisfiable.* ■

Proof. By definition $\mathbf{D}$'s limit tree has an exhausted branch $\mathbf{B}$.

As $\mathbf{B}$ is an exhausted branch, by definition it does not contain the empty clause. Furthermore by proposition 2 $\mathbf{B}_\infty$ is saturated up to redundancy and because of $\square \notin \mathbf{B}$ it follows $\square \notin \mathbf{B}_\infty$.

By theorem 1 $\mathbf{B}_\infty$ is satisfiable, as there is a rewrite system $\mathrm{R}_{\mathbf{B}_\infty}$ that is an E-model for $\mathbf{B}_\infty$, i.e. $\mathrm{R}_{\mathbf{B}_\infty} \models_E \mathbf{B}_\infty$.

To prove the theorem, it suffices to show that $\mathrm{R}_{\mathbf{B}_\infty} \models_E \mathcal{S}$, which can be done by showing that for any clause $C \in \mathcal{S}$ $\mathrm{R}_{\mathbf{B}_\infty} \models_E C$ holds.

By definition of derivation $\mathbf{B}_0$ is the single branch of the initial tableau of the derivation $\mathbf{D}$ and thus equivalent with $\mathcal{S}$. Therefore assume $C \in \mathbf{B}_0$.

If $C \in \mathbf{B}_\infty$, $\mathrm{R}_{\mathbf{B}_\infty} \models_E C$ follows immediately from $\mathrm{R}_{\mathbf{B}_\infty} \models_E \mathbf{B}_\infty$. Therefore suppose $C \notin \mathbf{B}_\infty$.

If $C \in \mathbf{B}_0$ but $C \notin \mathbf{B}_\infty$ the clause C was removed from the clause set $\mathbf{B}_k$ by the application of the Del or Simp rule at a certain step. Therefore C must be an object tautology clause, or non-properly subsumed or redundant. We now consider each possibility.

1. C is an object tautology clause
   Suppose C was removed from $\mathbf{B}_k$ because it was an object tautology clause, i.e. $C\sigma$ is like $\mathcal{A}, \leftarrow i_1 \simeq i_2, \mathcal{B}$ with $i_1, i_2 \in \mathbb{D}$ and $i_1 \neq i_2$.
   As i_1 and i_2 are two non-identical members of $\mathbb{D}$ and the unique name assumption applies to $\mathbb{D}$ the equation $i_1 \simeq i_2$ can never be true. With the false literal $i_1 \simeq i_2$ in the body of the clause the whole clause becomes true. Thus $\mathrm{R}_{\mathbf{B}_\infty} \models_E C$ follows trivially.
2. C is non-properly subsumed
   Suppose C was removed from $\mathbf{B}_k$ because it was non-properly subsumed by a clause $\mathcal{D} \in \mathbf{B}_k$. C must be a proper instance of $\mathcal{D}$, as by the derivation rules Equality and Split no derived clause set $\mathbf{B}_i$ can contain a clause and a variant of it. The converse relation to non-proper subsumption, called proper generalisation, is well founded. Thus, by induction on this ordering, there is a clause $\mathcal{D}'$ in $\mathbf{B}_\infty$ that non-properly subsumes C. As $\mathcal{D}' \in \mathbf{B}_\infty$ applies and $\mathrm{R}_b bi \models_E \mathcal{D}'$ holds, $\mathrm{R}_{\mathbf{B}_\infty} \models_E C$ holds, as well.

3. C is redundant
   Suppose C was removed from $\mathbf{B}_k$ because it was redundant w.r.t. a specific subset $\mathbf{B}'$ of the derived branch $\mathbf{B}_{k+1}$, where $\mathbf{B}'$ is specified in the definition of the Del and Simp derivation rules. Because $\mathbf{B}' \subseteq \mathbf{B}_{k+1}$ it follows trivially that C is redundant w.r.t. $\mathbf{B}_{k+1}$. By Lemma 4 C is redundant w.r.t. $\mathbf{B}_\infty$, which with Lemma 9 leads to $R_{\mathbf{B}\infty} \models_E C$.

□

With the completeness of the modified E-hyper tableau calculus shown and proven, the third part of the properties is concluded. The following and final part now takes care of the soundness.

5.5 Soundness

Now the soundness of the extended E-hyper tableau calculus is introduced and proven. In contrast to the completeness this part is rather straight forward and compact. It starts with Lemma 10, which states that if a premiss of a derivation rule Equality, Split, Del or Simp is E-satisfiable, one conclusion is E-satisfiable, as well. The proof is done straight forward by showing this statement for each of the four derivation rules.

Lemma 10. *For each of the derivation rules Equality, Split, Del and Simp holds, if the premiss of the rule is E-satisfiable, then one of its conclusions is E-satisfiable, as well.* ■

Proof. The claim is proven be examining each rule on its own.

For the Equality rule we take a look at the sup-left and unit-sup-right rule. If the premiss of such a rule is E-satisfiable then there is an E-model I. With the axioms of congruence it follows immediately that I is an E-model for the conclusion, as well. For ref the claims follows immediately from the reflexivity. With sup-left, unit-sup-right and ref being the underlying inference rules for the Equality rule, it can be concluded that the claim holds for Equality.

For Split, assume an E-model I for the premiss $\mathbf{B}$. With $A_1, \ldots, A_m \leftarrow \in \mathbf{B}$ and the purifying substitution π, I is an E-model for $(A_1, \ldots, A_m \leftarrow)\pi$. As all variables are implicitly universally quantified I has to be an E-model for $\forall(A_1, \ldots, A_m \leftarrow)$ Due to purification the set of variables of each A_i is disjunct, which allows writing $\forall A_1\pi \vee \ldots \vee \forall A_m\pi$ instead of $\forall(A_1\pi \vee \ldots \vee A_m\pi)$. Thus I is an E-model for one of $B \cdot A_1\pi \leftarrow^d, \ldots, B \cdot A_m\pi \leftarrow^d$.

For Del the claim holds directly from its definition.

For Simp assume an E-model I for the premiss. From the definition of the Simp rule(see Fig. 2) $(\mathbf{B} \cdot C \cdot \mathbf{B}_1) \models_E \mathcal{D}$ for clauses C, $\mathcal{D}$ and branches $\mathbf{B}$, $\mathbf{B}_1$ follows that $\mathcal{D}$ holds in I. □

The next step is Lemma 11, which states that if the derivation rule Inc is applicable, its premiss is E-unsatisfiable. The proof is straight forward and uses the definition of the Inc derivation rule.

Lemma 11. *If the derivation rule Inc is applicable, its premiss is E-unsatisfiable.* ■

Proof. For Inc to be applicable, there must be a $C \in \mathbf{B}$ and a substitution σ such that the inference $C \Rightarrow_{\text{unit-cont-right}(\sigma)} \square$ can be applied. Therefore C and σ must be of such a form that $C\sigma = i_1 \simeq i_2 \leftarrow$ with $i_1, i_2 \in \mathbb{D}$ and $i_1 \neq i_2$.

As i_1 and i_2 are two non-identical distinct object identifiers and the unique name assumption applies to the set of distinct object identifiers this equation can never be true. Therefore there is no E-model for $i_1 \simeq i_2 \leftarrow$ and as C is clause of $\mathbf{B}$, the branch $\mathbf{B}$ is E-unsatisfiable. □

Finally both just introduced lemmas are used to prove Theorem 3, which states the soundness of the extended E-hyper tableau calculus, i.e. if a clause set has an E-hyper tableau refutation it is E-unsatisfiable.

Theorem 3 (Soundness of E-Hyper Tableaux). *Let $\mathcal{S}$ be a clause set that has a refutation. Then $\mathcal{S}$ is E-unsatisfiable.* ■

Proof. Assume a refutation leads to the closed tableau $\mathbf{T}$. From Lemma 11 and the contrapositive of Lemma 10 we conclude that if a tableau $\mathbf{T}_i$ of a derivation contains only E-unsatisfiable branches, this holds for its predecessor $\mathbf{T}_{i-1}$, as well. Following the definition of a refutation, the final tableau $\mathbf{T}$ only consists of E-unsatisfiable branches.

By induction on the length of the refutation we can conclude that the initial tableau $\mathbf{T}_0$, consisting of one branch with the tableau clauses from $\mathcal{S}$ is E-unsatisfiable. □

6 Implementation and Evaluation

6.1 E-KRHyper

To evaluate the modified calculus, it was incorporated into the *E-KRHyper* system. The E-KRHyper system [19,18] is an automated theorem prover that implements the E-hyper tableau calculus. It is based on the KRHyper system [27,28], which is an implementation of the original hyper tableau calculus, which had no native handling of equalities. Both systems are well established and used in different areas. The areas of use include amongst others natural question answering [13], e-learning [6,7] and ontology reasoning [10].

E-KRHyper is written in the OCaml programming language[15]. It is a strongly and statically typed functional language that allows the usage of other programming paradigms, as well, and offers the possibility to create high performance programs.

The changes to the prover can be grouped into three parts:

1. The parser and lexer have been adapted to conform to the TPTP syntax for distinct objects, i.e. constants in double quotes, like `"con"`,are treated as distinct object identifiers.
2. The Inc and unit-cont-right() rules have been incorporated into the code.
3. Changes that allow us to handle object tautology clauses as being negligible have been done.

To avoid confusion, the code of the unchanged version of the E-KRHyper is called *traditional* code. Some more details on the implementation can be found in [11]. With a suitable implementation at hand, we are now able to evaluate the impact of the changed calculus.

6.2 Test Conditions

All of the following benchmarks have been executed on the same machine with the following specifications: CPU: Intel Core 2 Quad (Q9550) @ 2.83GHz, Memory: 4GB PC2-6400, Operating System: openSUSE 11.3, Kernel Version: 2.6.34.10-0.2-desktop, OCaml Version: 3.12.0.

For showing that the introduced changes improve E-KRHyper's behaviour if distinct object identifiers are used in problems, appropriate examples are needed. Unfortunately the TPTP contains only eight problems that utilise distinct object identifiers where five cannot be used with E-KRHyper due to technical issues. This leaves three usable examples for evaluating our approach, which is hardly suitable. Thus a variation of the synthetic benchmarks STORECOMM (SC) and STORECOMM-INVALID (SCI) [2,1] was chosen for evaluating our approach.

Both problem classes are situated in the theory of arrays.

A test case from SC is the task to show that two given permutations of unique store operations on an array result in the same array. A test case from SCI is the task to show that two given sequences of unique store operations that differ in at least one stored element at an index do not result in the same array. This definition of SCI differs from the original definition [1], where SCI is the task to show that two given sequences of unique store operations that differ in at least one unique store operation do not result in the same. We have chosen our version of SCI for technical reasons and continue to call it STORECOMM-INVALID or SCI in the context of this work. The term *unique store operation* states, that each index of an array is written to exactly one time.

As the theory of arrays is not natively supported by E-KRHyper, we need axioms to describe the theory of arrays. We start by introducing the function $sel : ARRAY \times INDEX \rightarrow ELEMENT$, which returns the element that is stored at the given index of the given array, and the function $sto : ARRAY \times INDEX \times ELEMENT \rightarrow ARRAY$, which returns an array that is constructed by storing a given element at the given index of a given array. Additionally we need the skolem function $sk : ARRAY \times ARRAY \rightarrow INDEX$ as a helper function, as neither the E-hyper tableau nor E-KRHyper is able to handle existentially quantified variables. These three operations are sufficient for our purpose and allow us to introduce the following axioms of the theory of arrays to E-KRHyper.

$$sel(sto(A, I, E), I) = E \tag{9}$$

$$sel(sto(A, I, E), J) = sel(A, J) \Leftarrow I \neq J \tag{10}$$

$$A = B \Leftarrow sel(A, sk(A, B)) = sel(B, sk(A, B)) \tag{11}$$

Due to some technicalities, Axiom 10 cannot be used in this form. Thus we rewrite it to the semantically equivalent formula 12.

$$I = J, sel(sto(A, I, E), J) = sel(A, J) \tag{12}$$

This formula contains I and J in both subformulæ of the disjunctive head, i.e. it is not pure. Thus E-KRHyper looks for an appropriate purifying substitution when this clause is used in the proving process, which does not terminate for this case. Hence the formula must be adopted to prevent the infinite search for a purifying substitution, which we achieve by adding the domain predicates $index(I)$ and $index(J)$ to the body of the clause. This guarantees a pure head, when the split-rule is applied and thus prevents the not regularly terminating search for a purifying substitution. This modification leads to the formula 13.

$$I = J, sel(sto(A, I, E), J) = sel(A, J) \Leftarrow index(I), index(J) \tag{13}$$

Thus equations 9, 11 and 13 form the axioms for the theory of arrays, which are used in the test cases.

For evaluation we need four different kinds of test cases: SC without native handling of distinct object identifiers, SC with native handling of DOI, SCI without native handling of DOI, and SCI with native handling of DOI.

To create a test case, four parameters are needed: A list $p = 0, \ldots, n-1$, a permutation of this list, called q, a flag v that indicates if we want to generate a test case for SC or SCI and a flag d that indicates if this test case uses distinct object identifiers or not.

Independent of the chosen parameters every test case contains the three axioms that describe the theory of arrays. Additionally every test contains n unique predicates of form $index(i_x)$ with $0 \leq x < n$ introducing the constants that represent the arrays indices. If distinct object identifiers are used, these predicates look like $index(''i_x'')$.

If no distinct object identifiers are used, we need to express that all indices are distinct, which is done by introducing $\binom{n}{2}$ unique predicates of form $false$:- $i_x = i_y$ with $(x, y) \in C_2^n$, where C_2^n is the set of 2-combinations over $\{0, \ldots, n-1\}$. Additionally we need to express that all elements are distinct, which is done by introducing $\binom{n}{2}$ unique predicates of form $false$:- $e_x = e_y$ with $(x, y) \in C_2^n$.

The actual property to be proven is then added by the equality predicate $T_{n,v,d}(q) = T_{n,v,d}(p)$, where $T_{k,v,d}(l)$ is defined as follows:

$$T_{k,v,d}(l) = \begin{cases} a & \text{if } k = 0 \\ sto(T_{k-1,v,d}(l), i_{l(k)}, e_0) & \text{if } k = 1 \text{ and } v = 0 \text{ and } d = 0 \\ sto(T_{k-1,v,d}(l), ''i_{l(k)}'', ''e_0'') & \text{if } k = 1 \text{ and } v = 0 \text{ and } d = 1 \\ sto(T_{k-1,v,d}(l), i_{l(k)}, e_{l(k)}) & \text{if } 0 \leq k < n \text{ and } v = 1 \text{ and } d = 0 \\ sto(T_{k-1,v,d}(l), ''i_{l(k)}'', ''e_{l(k)}'') & \text{if } 0 \leq k < n \text{ and } v = 1 \text{ and } d = 1 \end{cases}$$

For illustration, Figure 7 shows four files for the four different type of test cases for an array with length two. For a better overview, the axioms are shown once and then referred to by the meta symbol «AXIOMS» in the specific examples.

For evaluation, we created test cases for array length from 5 to 95 elements with 20 different samples for each length, which results in $10 * 20 * 4 = 400$ files. The problems are then flattened, i.e. nested function calls are eliminated by introducing new constants. This step is sound and can always be executed and preliminary tests have shown that it leads to shorter execution times. We split the 400 files into four lists, depending on the fact if they involve distinct objects or not and if they cover SC or SCI. Four instances of the E-KRHyper are started where each works on one list. The execution time and outcome of each problem is saved.

```
sel(sto(A,I,E),I) = E.

I = J; sel(sto(A,I,E),J) = sel(A,J) :-
    index(I),
    index(J).

A = B :- sel(A,sk(A,B)) = sel(B,sk(A,B)).
```

(a) The AXIOMS.

```
<<AXIOMS>>

index(i0).
index(i1).

false :- i0 = i1.
false :- e0 = e1.

sto(sto(a,i1,e1),i0,e0) = sto(sto(a,i0,e0),i1,e1).
```

(b) SC without distinct object identifiers.

```
<<AXIOMS>>

index("i0").
index("i1").

sto(sto(a,"i1","e1"),"i0","e0") = sto(sto(a,"i0","e0"),"i1","e1").
```

(c) SC with distinct object identifiers.

```
<<AXIOMS>>

index(i0).
index(i1).

false :- i0 = i1.
false :- e0 = e1.

sto(sto(a,i1,e1),i0,e0) = sto(sto(a,i0,e0),i1,e0).
```

(d) SCI without distinct object identifiers.

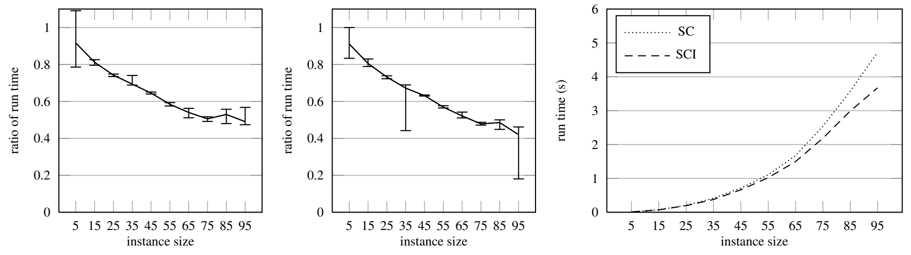
```
<<AXIOMS>>

index("i0").
index("i1").

sto(sto(a,"i1","e1"),"i0","e0") = sto(sto(a,"i0","e0"),"i1","e0").
```

(e) SCI with distinct object identifiers.

Figure 7: Examples for SC and SCI files with size two.



(a) Run time gain for SC (b) Run time gain for SCI (c) Run times for SCI and SC using DOI

Figure 8: Relative execution times for S-COMM and S-COMM-INV.

6.3 Analysis

The outcome of all test cases was as expected, i.e. for cases that are valid (all cases for SC), the prover derived that they are valid, and for cases that are invalid (all cases for SCI), the prover derived that they are invalid.

For an easier comparison of the results with and without DOI, they were set into relation by dividing the runtimes with DOI by the according runtimes without DOI. The result of this operation can be seen in Fig 8a for SC and Fig.8b for SCI, where the graph shows the average, minimal and maximal runtime, for each size. It is easy to see that the version with DOI outperforms the one without DOI and that the difference grows with the size of the arrays. We assume that the specific structure of the problems influences the runtime, which might lead to the visible fluctuations in the graphs. This has not yet been investigated. We compared the results of the DOI version of E-KRHyper with the results of the prover E, which is also able to handle DOI [23]. E solved the 200 problems for SC with an average runtime of 0.09second and the 200 problems for SCI in an average runtime of 0.08 seconds, where E-KRHyper needed 1.5 respectively 1.27 seconds. Fig. 8c shows that for E-KRHyper with DOI the runtime in relation to the array size is exponential.

7 Related Work

As the topic of reasoning with the unique name assumption or distinct object identifiers has not yet gotten a lot of attention in the field of automated reasoning, there is only one paper known to us that is strongly related to our work.

In [23], Schulz and Bonacina show a way to handle distinct object identifiers in the superposition calculus. The authors introduced four new rules that correspond two the two rules we introduced and deal with exactly the same two types of formulæ.

To evaluate their approach they extended a version of the E-prover [21,22] and used instances of the *STORECOMM* and the original version of *STORECOMM-INVALID* benchmark classes [2,1]. To rate the performance, they compared four different systems: CVC [24], which is a validity checker where the axioms for the theory of arrays is part of the actual system, CVC-Lite [5], which is the successor of CVC and has native support for the theory of arrays, as well, a version of the E-prover with support for distinct objects identifiers and a version of the E-prover that does not support DOI and thus needs additional facts to define the inequality of distinct array indices.

The results of these tests indicate a significant improvement. The execution times of the version of E that supports DOI for the valid cases are identical with the runtime of CVC and with the invalid cases they are even lower. The version of E that does not support DOI and therefore relies on additional facts for the reasoning process is considerably slower, which supports the claim, that native handling of the unique name assumption can be beneficial for certain reasoning tasks.

As E's superiority in this test scenario was easy to see in the preliminary tests we did, no thorough comparison of E and our approach was done

Another field, known to us, that implicitly uses the unique name assumption, is the reasoning in many valued logics [3,14].

8 Conclusion

Using the unique name assumption instead of facts to define inequalities of constants reduces the number of clauses in knowledge bases and thus allows the reader to focus on the parts that are of actual importance for a problem. A smaller set of clauses allows faster reasoning, as well.

We have introduced a sound and complete extension of the E-hyper tableau calculus with native handling of the unique name assumption and we implemented the calculus in the E-KRHyper system.

This implementation was then used for evaluating whether the use of distinct object identifiers has an impact on the outcome or needed time for proving a problem. The evaluation shows that the implemented changes significantly improve the execution times for problems with distinct object identifiers.

Another observation made in the evaluation process was the scattering of the execution times for some samples. We suppose that the structure of the problem, i.e. the order of the store operations, has an impact on the execution time and the difference of the execution times between the traditional and modified version of the E-KRHyper. Thus future work is needed to perform a thorough study on the correlation of array size, order of store operations and execution time for a single sample to elaborate and support this assumption.

Independent of the actual execution time and the speed-up of the execution time by using distinct object identifiers, we learned that we cannot compete with other systems, like the E prover, in this class of benchmarks.

Acknowledgements Our gratitude goes to Ulrich Furbach, Christian Schwarz and Viorica Sofronie-Stokkermans for their helpful comments on early versions of this paper. Additionally we thank Björn Pelzer for sharing his knowledge on E-KRHyper and Stephan Schulz for his help on finding the right settings to use with E in our benchmark.

References

1. Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 2009.
2. Alessandro Armando, Maria Paola Bonacina, Aditya Kumar Sehgal, and Silvio Ranise. High-performance deduction for verification: A case study in the theory of arrays. In *Notes of the Workshop on Verification, Third Federated Logic Conference (FLoC02)*, 2002.
3. Matthias Baaz and Christian G. Fermüller. Resolution-based theorem proving for manyvalued logics. *J. Symb. Comput.*, 1995.
4. Leo Bachmair and Harald Ganzinger. Equational reasoning in saturation-based theorem proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction — A Basis for Applications*. Kluwer, 1998.
5. Clark W. Barrett and Sergey Berezin. CVC Lite: A new implementation of the cooperating validity checker category B. In Rajeev Alur and Doron Peled, editors, *CAV 2004, Proc.* Springer, 2004.
6. Peter Baumgartner and Ulrich Furbach. Living books, automated deduction and other strange things. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*. Springer, 2005.
7. Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, and Alex Sinner. Living book - deduction, slicing, and interaction. *J. Autom. Reason.*, 2004.
8. Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper tableaux. In José Júlio Alferes, Luís Moniz Pereira, and Ewa Orlowska, editors, *JELIA '96, Proc.* Springer, 1996.
9. Peter Baumgartner, Ulrich Furbach, and Björn Pelzer. Hyper tableaux with equality. In Frank Pfenning, editor, *CADE-21, Proc.* Springer, 2007.
10. Peter Baumgartner and Renate A. Schmidt. Blocking and other enhancements for bottom-up model generation methods. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR 2006, Proc.* Springer, 2006.
11. Markus Bender. Extending the E-hyper tableau calculus for reasoning with the unique name assumption. Diplomarbeit, University of Koblenz-Landau, January 2012. Available at `userp.uni-koblenz.de/~mbender/store/da.pdf`.
12. Marc Bezem, Jan Willem Klop, and Roel de Vrijer. *Term Rewriting Systems*. Cambridge University Press, 2003.
13. Ulrich Furbach, Ingo Glöckner, and Björn Pelzer. An application of automated reasoning in natural language question answering. *AI Commun.*, 2010. PAAR Special Issue.
14. Harald Ganzinger and Viorica Sofronie-Stokkermans. Chaining techniques for automated theorem proving in many-valued logics. In *ISMVL 2000, Proc.*. IEEE Computer Society, 2000.
15. INRIA. Website of OCaml. `http://caml.inria.fr/`, 2005-2013. Accessed Jan., 27th 2013.
16. Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier and MIT Press, 2001.
17. Björn Pelzer. E-KRHyper – extending the KRHyper theorem prover with equality reasoning. Diplomarbeit, University of Koblenz-Landau, March 2007.
18. Björn Pelzer. Project website of E-KRHyper. `http://userp.uni-koblenz.de/~bpelzer/ekrhyper/`, 2007-2013. Accessed Jan., 21st 2013.
19. Björn Pelzer and Christoph Wernhard. System description: E-KRHyper. In Frank Pfenning, editor, *CADE-21, Proc.* Springer, 2007.
20. David Poole, Alan Mackworth, and Randy Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, January 1998.

21. Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 2002.
22. Stephan Schulz. System description: E 0.81. In David A. Basin and Michaël Rusinowitch, editors, *IJCAR 2004 Proc.* Springer, 2004.
23. Stephan Schulz and Maria Paola Bonacina. On handling distinct objects in the superposition calculus. In Boris Konev and Stephan Schulz, editors, *IWIL 2005, Proc.*, 2005.
24. Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: A cooperating validity checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *CAV 2002 Proc.* Springer, 2002.
25. Geoff Sutcliffe. The TPTP website. `www.tptp.org`, 2004-2013. Accessed Jan., 21st 2013.
26. Geoff Sutcliffe. The TPTP problem library and associated infrastructure: The FOF and CNF parts, v3.5.0. *J. Autom. Reason.*, 2009.
27. Christoph Wernhard. System description: KRHyper. Fachberichte informatik, Universität Koblenz-Landau, 2003.
28. Christoph Wernhard. Project website of KRHyper. `http://userp.uni-koblenz.de/~wernhard/krhyper/`, 2003-2013. Accessed Jan., 21st 2013.

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch, Comparing the efficiency of serial and parallel algorithms for training artificial neural networks using computer clusters, Arbeitsberichte aus dem Fachbereich Informatik, 13/2011

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch, A parallel algorithm for selecting activation functions of an artificial network, Arbeitsberichte aus dem Fachbereich Informatik 12/2011

Katharina Bräunlich, Rüdiger Grimm, Andreas Kasten, Sven Vowé, Nico Jahn, Der neue Personalausweis zur Authentifizierung von Wählern bei Onlinewahlen, Arbeitsberichte aus dem Fachbereich Informatik 11/2011

Daniel Eißing, Ansgar Scherp, Steffen Staab, Formal Integration of Individual Knowledge Work and Organizational Knowledge Work with the Core Ontology *strukt*, Arbeitsberichte aus dem Fachbereich Informatik 10/2011

Bernhard Reinert, Martin Schumann, Stefan Müller, Combined Non-Linear Pose Estimation from Points and Lines, Arbeitsberichte aus dem Fachbereich Informatik 9/2011

Tina Walber, Ansgar Scherp, Steffen Staab, Towards the Understanding of Image Semantics by Gaze-based Tag-to-Region Assignments, Arbeitsberichte aus dem Fachbereich Informatik 8/2011

Alexander Kleinen, Ansgar Scherp, Steffen Staab, Mobile Facets – Faceted Search and Exploration of Open Social Media Data on a Touchscreen Mobile Phone, Arbeitsberichte aus dem Fachbereich Informatik 7/2011

Anna Lantsberg, Klaus G. Troitzsch, Towards A Methodology of Developing Models of E-Service Quality Assessment in Healthcare, Arbeitsberichte aus dem Fachbereich Informatik 6/2011

Ansgar Scherp, Carsten Saathoff, Thomas Franz, Steffen Staab, Designing Core Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 5/2011

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch, The prediction of currency exchange rates using artificial neural networks, Arbeitsberichte aus dem Fachbereich Informatik 4/2011

Klaus G. Troitzsch, Anna Lantsberg, Requirements for Health Care Related Websites in Russia: Results from an Analysis of American, British and German Examples, Arbeitsberichte aus dem Fachbereich Informatik 3/2011

Klaus G. Troitzsch, Oleg Kryuchin, Alexander Arzamastsev, A universal simulator based on artificial neural networks for computer clusters, Arbeitsberichte aus dem Fachbereich Informatik 2/2011

Klaus G. Troitzsch, Natalia Zenkova, Alexander Arzamastsev, Development of a technology of designing intelligent information systems for the estimation of social objects, Arbeitsberichte aus dem Fachbereich Informatik 1/2011

Kurt Lautenbach, A Petri Net Approach for Propagating Probabilities and Mass Functions, Arbeitsberichte aus dem Fachbereich Informatik 13/2010

Claudia Schon, Linkless Normal Form for ALC Concepts, Arbeitsberichte aus dem Fachbereich Informatik 12/2010

Alexander Hug, Informatik hautnah erleben, Arbeitsberichte aus dem Fachbereich Informatik 11/2010