IOS Press

A Resolution Calculus with Shared Literals

N. Peltier
CNRS - Leibniz-IMAG Laboratory
46 avenue Félix Viallet
38031 Grenoble Cedex - FRANCE
Nicolas.Peltier@imag.fr

Abstract. We present a resolution calculus for first-order logic using a more concise formalism for representing sets of clauses. The idea is to represent the clause set at hand as a **D**irected **A**cyclic **G**raph, which allows one to share common literals instead of duplicating them, thus yielding a much more compact representation of the search space. We define inference rules operating on this language and we prove their soundness and refutational completeness. We also design simplification rules for pruning the search space. Finally we compare our technique with the usual resolution calculus and we prove (using the pigeonhole example) that our method can reduce the length of the proof by an exponential factor (in propositional logic).

Keywords: Resolution, Sharing, Pigeonhole problem.

1. Introduction

The resolution method [18, 13] is one of the most widely used approaches in first-order theorem proving. Most powerful theorem provers nowadays are based on resolution techniques (using superposition for handling equalities). The *uniformity* and very low *branching factor* of the resolution calculus makes it very efficient in practice (especially when strategies are used to reduce the search space).

However, when compared to other logical formalisms such as natural deduction or sequent calculi, resolution has an important drawback. The length of the obtained proof is at most exponentially shorter than the Herbrand complexity of the clause set at hand (i.e. the number of closed instances needed in the derivation). This implies that resolution proofs are in general non-elementary longer than the

Address for correspondence: Nicolas Peltier
Leibniz-IMAG - 46, avenue Félix Viallet
38031 Grenoble Cedex
France

shortest proof in sequent calculus with cut for instance [19]. Resolution is known to be inefficient on some particular (propositional) formulae or sequences of formulae such as the well-known pigeonhole problem for which the shortest resolution proof is exponential, whereas polynomial proofs exist in more powerful systems (see for instance [12, 7, 4] for more details about this issue).

Various techniques have been proposed for overcoming this problem. In particular, the non-analytic "cut" rule used in sequent calculi can be easily simulated in the context of the resolution calculus by adding a rule dynamically introducing new predicate definitions in the clause sets. This idea has been introduced by Tseitin [20] for propositional proofs and extended to first-order logic in [9]. Definitions of the form $p(\vec{x}) \Leftrightarrow \phi$ where p is a new predicate symbol and ϕ a formula on the variables in $\vec{x}$ can be derived and processed (after transformation into clausal form). Unfortunately, just as the cut rule, this approach is not suitable for pure automatic proof search since no heuristic exists for choosing the right definitions (systematic introduction of all possible definitions is of course unrealistic). Function introduction techniques [2] have also been designed for simulating some particular form of cut introduction. [10] considered various kinds of function introduction rules and analyzed the influence of these rules on the length of the shortest proofs. Still, this approach greatly increases the branching factor of the calculi.

In this paper, our aim is to introduce a technique for reducing proof length which does not suffer from this drawback, thus does not increase too much the search space. The basic idea is to avoid duplicating literals – or disjunctions of literals – during the search (in particular when applying the resolution rule). To this purpose, we allow disjunctions of literals to be *shared* between distinct clauses, which avoids having to explicitly copy them. Consequently, the inferences applied on these literals are *simultaneously* proceeded on *all* the clauses in which it occurs. More precisely, instead of defining a calculus operating on clauses as usual (i.e. on disjunctions of literals), we consider more complex (quantifier-free) formulae, which we called "†-formulae", possibly containing *conjunctions*, *disjunctions*, and *shared subformulae*. This formalism is related to AND-OR graphs that are commonly used to represent the search space of subgoal-reduction strategies (in particular in Horn logic). It has the same theoretical expressive power of clause sets, but is much more concise. Then we define appropriate inference and simplification rules operating on such †-formulae. These rules extend the usual resolution, factorization or subsumption rules to the new language. We shall see that the obtained calculus allows one to factorize some part of the proofs – thus significantly reducing their length – due to the possibility of sharing identical subgoals. It can also prune the search space.

Before entering in the technical details, we illustrate our point on a (deliberately simple) example, in order to allow the reader to grasp the intuitive ideas behind our method.

Let S be the following clause set (p, q, r are predicate symbols, x is a variable, $a_1, \ldots, a_n$ denotes constant symbols).

$$\begin{array}{ll} 1 & p(a_1) \vee \ldots \vee p(a_n) \\ 2 & \neg p(x) \vee q(x) \\ 3 & \neg p(x) \vee r(x) \end{array}$$

Assume that the atoms are ordered as follows: $p(x) > q(y) > r(z)$, with $p(a_1) > p(a_2) \ldots > p(a_n)$ (the ordering is chosen only to illustrate our point). Literal $p(a_1)$ is maximal in Clause 1. We can resolve

Clause 1 with Clauses 2 and 3 yielding respectively the two following clauses:

4	$q(a_1) \vee p(a_2) \vee \ldots \vee p(a_n)$	(res, $1, 2$)
5	$r(a_1) \vee p(a_2) \vee \ldots \vee p(a_n)$	(res, $1, 2$)

At this point $p(a_2)$ becomes maximal in Clauses 4 and 5, thus we can again apply the resolution rule with clauses 2 and 3, yielding

6	$q(a_1) \vee q(a_2) \vee p(a_3) \vee \ldots \vee p(a_n)$	(res, $4, 2$)
7	$r(a_1) \vee q(a_2) \vee p(a_3) \vee \ldots \vee p(a_n)$	(res, $5, 2$)
8	$q(a_1) \vee r(a_2) \vee p(a_3) \vee \ldots \vee p(a_n)$	(res, $4, 3$)
9	$r(a_1) \vee r(a_2) \vee p(a_3) \vee \ldots \vee p(a_n)$	(res, $5, 3$)

By repeating this process for each constant symbol a_i, we get 2^n clauses of the form $\bigvee_{i=1}^{n} p_i(a_i)$, where for all $i \in [1..n]$, $p_i = q$ or $p_i = r$.

Clearly, this huge search space can be significantly reduced by using a more concise and more appropriate representation of clauses. The above clause set can be represented as a disjunction

$$\bigvee_{i=1}^{n} \phi(a_i),$$

where $\phi(a_i)$ denotes the conjunction $p(a_i) \wedge q(a_i) \wedge r(a_i)$.

This conjunction could be obtained from the original clause set as follows. First the resolution rule is applied on 1 and 2, yielding $q(a_1) \vee p(a_2) \ldots \vee p(a_n)$. But instead of duplicating the whole sequence of literals $p(a_2), \ldots, p(a_n)$ we simply "insert" the literal $q(a_1)$ in clause 1. Indeed, by distributivity, the conjunction

$$[p(a_1) \vee p(a_2) \ldots \vee p(a_n)] \wedge [q(a_1) \vee p(a_2) \ldots \vee p(a_n)]$$

is equivalent to

$$[p(a_1) \wedge q(a_1)] \vee p(a_2) \ldots \vee p(a_n).$$

Thus instead of adding a new clause, we merely *replace* the resolved literal in the original clause (i.e. $p(a_1)$) by the conjunction $p(a_1) \wedge q(a_1)$.

This principle can be generalized. In order to apply the resolution rule on a literal l occurring in a formula ϕ and a literal $\neg l$ occurring in a formula ψ, we replace the occurrence of l inside ϕ by the conjunction $l \wedge \psi'$, where ψ' is obtained from ψ by replacing $\neg l$ by *false*. Obviously, the context is not affected by the application of the rule. Of course in the first-order case some variables can be instantiated which implies that some copying is still necessary (as we shall see in Section 3). But our technique reduces this to a minimum.

At this point, we get:

$$[p(a_1) \wedge q(a_1)] \vee p(a_2) \vee \ldots \vee p(a_n)$$

Then we apply the resolution rule on $q(a_1)$ using the clause 3. According to the above principle, we replace the literal $q(a_1)$ by $(q(a_1) \wedge r(a_1))$, hence we get:

$$[p(a_1) \wedge q(a_1) \wedge r(a_1)] \vee p(a_2) \vee \ldots p(a_n)$$

By repeating this process for any $i \in [1..n]$ we get:

$$\bigvee_{i=1}^{n} (p(a_i) \wedge q(a_i) \wedge r(a_i)).$$

In some sense, our approach has similar effects as the adding of additional definitions by the extension rule of Tseitin or Eder for instance (see above). For instance, in the above example, $\phi(a_i)$ could be introduced by an extension step as a name for the formula $p(a_i) \wedge q(a_i) \wedge r(a_i)$ and the clause $\bigvee_{i=1}^{n} \phi(a_i)$ could be inferred. Our technique, though more restricted, has some advantages. First, the choice of the formulae that should be "named" is strongly guided by the context: we do not rely on "blind" generation of arbitrary definitions. Second, the adding of the definitions is made implicit in the calculus.

The rest of this paper is devoted to a formalization of this technique and to the study of some of the properties of the obtained calculus. It is structured as follows.

- In Section 2 we introduce a new language for denoting clauses and sets of clauses, the so-called †-formulae. We describe the syntax and semantics of our language and provide some additional definitions.
- In Section 3 we show how to extend the resolution calculus in order to handle †-formulae instead of clauses. We define generalizations of the resolution an factorization rules, and we adapt the usual subsumption rule. We also define some essentially new simplification rules (i.e. rules that are specific to †-formulae). The obtained calculus is called †-resolution.
- In Section 4.1 we prove the soundness and refutational completeness of †-resolution (with simplification and redundancy criteria).
- In Section 5 we show that †-resolution refutes the pigeonhole formula in a polynomial number of steps (which implies that it cannot be simulated by ordinary resolution).
- Section 6 compares our approach with similar ones in the literature.
- Finally, Section 7 concludes the paper.

2. †-Formulae

In this section we introduce the syntax and semantics of the language of †-formulae. A †-formula can be seen as a special case of a (quantifier-free) first-order formula, in which some of the subformulae can be shared in order to reduce the size of the formula.

2.1. Preliminaries

We first review some basic definitions and notations. Though all the necessary definitions are provided, we assume some familiarity with the usual notions in logic and automated deduction (see for instance [11]).

We assume given 3 disjoint sets of symbols: a set of *function symbols* Σ (including constants), a set of *predicate symbols* Ω, a set of *variables* $\mathcal{V}$. Let *arity* be a function mapping each symbol in $\Sigma \cup \Omega$ to a unique natural number.

The set of *terms* is built inductively as usual on the set of function symbols Σ and the set of variables $\mathcal{V}$. An *atom* is an expression of the form $p(t_1, \dots, t_n)$ where $p \in \Omega$, $n = \mathit{arity}(p)$ and $t_1, \dots, t_n$ are terms. A *literal* is either an atom (positive literal) or the negation of an atom (negative literal). If l is a literal, l^c denotes the literal complementary of l (i.e. $l^c \overset{\text{def}}{=} \neg l$ if l is positive and $(\neg l)^c \overset{\text{def}}{=} l$).

A *clause* is a finite multiset of literals (written as a disjunction). The empty clause is denoted by $\Box$.

If l is a literal (resp. term, atom, clause) then $\mathit{var}(l)$ denotes the set of variables occurring in l. An expression with no variable is said to be *ground*.

A *substitution* σ is a function mapping each variable x to a term denoted by $x\sigma$. As usual, a substitution can be extended into a homomorphism on the set of terms (resp. atoms, literals and clauses). If t, s are two terms (or atoms, literals) then σ is said to be a *unifier* of t, s iff $t\sigma = s\sigma$. It is well-known [1] that any unifiable pair of terms has a most general unifier (m.g.u.), unique up to a renaming.

If $C_1 = l \vee D_1$ and $C_2 = l' \vee D_2$ are two ground clauses and σ is the m.g.u. of l^c and l', then the clause $(D_1 \vee D_2)\sigma$ is called a *resolvent* of C_1, C_2. If $C = (l \vee l' \vee D)$ and σ is the m.g.u. of l, l', then $(l \vee D)\sigma$ is a *factor* of C.

An *interpretation* is a (possibly infinite) set of ground atoms. An interpretation $\mathcal{I}$ satisfies a ground literal l if either l is positive and occurs in $\mathcal{I}$ or l is negative and $l^c \notin I$. An interpretation satisfies a ground clause C if $\mathcal{I}$ satisfies a literal $l \in C$. An interpretation satisfies a set of clauses S if for any $C \in S$ and for any ground substitution σ, $\mathcal{I}$ satisfies $C\sigma$. We write $\mathcal{I} \models S$ if $\mathcal{I}$ satisfies S.

A clause C *subsumes* a clause D iff there exists a substitution σ s.t. $C\sigma \subseteq D$. This is written $C \preccurlyeq^{\mathit{sub}} D$. This notation may be extended to clause sets: if S, S' are two clause sets, then we write $S \preccurlyeq^{\mathit{sub}} S'$ iff for any clause $C' \in S'$ there exists a clause $C \in S$ s.t. $C \preccurlyeq^{\mathit{sub}} C'$.

2.2. Syntax of †-Formulae

Let Λ be a set of *nodes* disjoint from $\Sigma, \Omega, \mathcal{V}$.

Let $\mathcal{L} \subseteq \Lambda$. A *$\mathcal{L}$-clause* is a finite multiset set of nodes occurring in $\mathcal{L}$ (denoted as a disjunction). The empty $\mathcal{L}$-clause is denoted by $\Box$.

Definition 2.1. (†-Formulae) A *†-formula* is a 5-tuple $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ where:

- $\mathcal{L} \subseteq \Lambda$ ($\mathcal{L}$ is the set of nodes in $\mathcal{C}$);
- $\mathcal{M} \subseteq \mathcal{L}$ ($\mathcal{M}$ is the subset of nodes that are labeled by literals);
- $\alpha \in \mathcal{L}$ (α denotes the root of $\mathcal{C}$);
- δ is a function mapping each symbol in $\mathcal{L} \backslash \mathcal{M}$ to a set of $\mathcal{L}$-clauses (δ may be viewed as a transition function, mapping each node not labeled by a literal to a set of $\mathcal{L}$-clauses, i.e. a disjunction of nodes).
- μ is a function mapping each symbol in $\mathcal{M}$ to a literal.

Example 2.1. The tuple $\mathcal{C} = (\{\alpha, \beta, \gamma, \lambda, \zeta\}, \{\gamma, \lambda, \zeta\}, \alpha, \delta, \mu)$ is a †-formula, where:

$$\delta = \{\alpha \mapsto \{\beta \vee \gamma\}, \beta \mapsto \{\lambda, \zeta\}\}$$

and

$$\mu = \{\lambda \mapsto p(f(x)), \zeta \mapsto p(g(x)), \gamma \mapsto \neg p(x)\}.$$

As we shall see $\mathcal{C}$ is equivalent to

$$(p(f(x)) \wedge p(g(x))) \vee \neg p(x).$$

Remark 2.1. The reader should note that we do not require that $\mathcal{L}$ is finite. The †-formulae we consider in the paper are mostly finite. However, for technical reasons, we prefer to keep to possibility of considering infinite †-formulae. This will be useful in the proof of refutational completeness, in which infinite †-formulae are constructed as "limits" of sequences of finite †-formulae (finite †-formulae can easily be constructed and handled by a computer, whereas handling infinite †-formulae would require some mechanism to denote them symbolically).

Given a †-formula $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$, we define inductively a relation $\preceq_{\mathcal{C}}$ on nodes in $\mathcal{L}$ as follows: $\gamma \preceq_{\mathcal{C}} \beta$ iff either $\gamma = \beta$ or there exists $C \in \delta(\beta)$ and $\zeta \in C$ s.t. $\gamma \preceq_{\mathcal{C}} \zeta$. Intuitively, $\gamma \preceq_{\mathcal{C}} \beta$ holds if the node γ occurs behind the node β.

A †-formula $\mathcal{C}$ is said to be *acyclic* if $\prec_{\mathcal{C}}$ is an ordering. **In the rest of the paper we assume (without explicitly mentioning it) that all the considered †-formulae are acyclic.**

Given a †-formula $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ we denote by *var*$(\mathcal{C})$ the set of variables occurring in $\mathcal{C}$, i.e. the set of variables occurring in a literal $\mu(\beta)$, for some $\beta \in \mathcal{M}$. $\mathcal{C}$ is said to be *ground* if *var*$(\mathcal{C}) = \emptyset$.

Two †-formulae $\mathcal{C}_i = (\mathcal{L}_i, \mathcal{M}_i, \alpha_i, \delta_i, \mu_i)$ for $i = 1, 2$ are said to be *disjoint* iff $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$. Clearly, two given †-formulae can always be made disjoint by renaming (automatically) the nodes occurring in one of them in order to satisfy the desired properties (as we shall see, this preserves the semantics of the †-formulae).

If $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ is a †-formula, and σ is a substitution, then $\mathcal{C}\sigma$ denotes the †-formula:

$$(\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu\sigma),$$

where for any $\beta \in \mathcal{M}$, $\mu\sigma(\beta) \stackrel{\text{def}}{=} \mu(\beta)\sigma$.

Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ be a †-formula. For any $\beta \in \mathcal{L}$, we denote by $\mathcal{C}_{|\beta}$ the †-formula $(\mathcal{L}, \mathcal{M}, \beta, \delta, \mu)$. $\mathcal{C}_{|\beta}$ can be seen as a sub-†-formula in $\mathcal{C}$, starting at the "root" β instead of α. $\mathcal{C}_{|\beta}$ is said to be a *sub-†-formula* of $\mathcal{C}$.

The notation $\mathcal{C}_{|\beta}$ can be extended to the case in which β denotes a $\mathcal{L}$-clause or a set of $\mathcal{L}$-clauses. In this case we add a new (i.e. not occurring in $\mathcal{C}$) node α' denoting the clause β. More precisely, if $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ is a †-formula and D is a $\mathcal{L}$-clause, then $\mathcal{C}_{|D}$ denotes the †-formula:

$$(\mathcal{L} \cup \{\alpha'\}, \mathcal{M}, \alpha', \delta \cup \{\alpha' \mapsto \{D\}, \mu),$$

where α' is a node not occurring in $\mathcal{L}$. If S is a set of $\mathcal{L}$-clauses, then $\mathcal{C}_{|S}$ denotes the †-formula:

$$(\mathcal{L} \cup \{\alpha'\}, \mathcal{M}, \alpha', \delta \cup \{\alpha' \mapsto S, \mu),$$

where α' is a node not occurring in $\mathcal{L}$.

Example 2.2. For instance, if $\mathcal{C}$ is the †-formula of Example 2.1, then $\mathcal{C}_{|\beta}$ denotes the †-formula $\mathcal{C}' = (\{\alpha, \beta, \gamma, \lambda, \zeta\}, \{\gamma, \lambda, \zeta\}, \beta, \delta, \mu)$.

$\mathcal{C}_{|\alpha \vee \beta}$ denotes a †-formula of the form $\mathcal{C}'' = (\{\alpha, \beta, \gamma, \lambda, \zeta, \epsilon\}, \{\gamma, \lambda, \zeta\}, \epsilon, \delta', \mu)$ where:

$$\delta' = \{\epsilon \mapsto \{\alpha \vee \beta\}, \alpha \mapsto \{\beta \vee \gamma\}, \beta \mapsto \{\lambda, \zeta\}\}.$$

As we shall see, $\mathcal{C}''$ is equivalent to:

$$(p(f(x)) \wedge p(g(x))) \vee \neg p(x) \vee (p(f(x)) \wedge p(g(x))).$$

2.3. Semantics of †-Formulae

From †-Formulae to Clause Sets

Any †-formula $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ can be associated to a set of clauses $\mathcal{S}(\mathcal{C})$ defined as follows:

Definition 2.2. (Semantics of †-Formulae) Let $\mathcal{C}$ be an †-formula. $\mathcal{S}(\mathcal{C})$ is the smallest set of clauses s.t.:

- If $\alpha \in \mathcal{M}$, then $\mathcal{S}(\mathcal{C}) \overset{\text{def}}{=} \{\mu(\alpha)\}$.
- If $\alpha \notin \mathcal{M}$ and $(\alpha_1 \vee \ldots \vee \alpha_k) \in \delta(\alpha)$ and for any $i \in [1..n]$, $C_i \in \mathcal{S}(\mathcal{C}_{|\alpha_i})$ then $C_1 \vee \ldots \vee C_n \in \mathcal{S}(\mathcal{C})$.

An interpretation $\mathcal{I}$ *satisfies* a †-formula $\mathcal{C}$ (written $\mathcal{I} \models \mathcal{C}$) iff $\mathcal{I} \models \mathcal{S}(\mathcal{C})$.

Note that this definition does not require that $\mathcal{C}$ is finite. The set $\mathcal{S}(\mathcal{C})$ is exactly the set of clauses obtained from $\mathcal{C}$ by transformation into clausal form, if a “naive” transformation is used.

It is easy to see that standard clauses can be seen as a particular case of †-formulae. More precisely, a clause $L_1 \vee \ldots \vee L_n$ is equivalent to the †-formula:

$$(\{\alpha, \beta_1, \ldots, \beta_n\}, \{\beta_1, \ldots, \beta_n\}, \alpha, \{\alpha \mapsto \{(\beta_1 \vee \ldots \vee \beta_n)\}\}, \{\beta_i \mapsto L_i \mid i \in [1..n]\}).$$

Similarly, if $S = \{C_1, \ldots, C_n\}$ is a set of clauses where $C_i \overset{\text{def}}{=} \bigvee_{j=1}^{k_i} L_{ij}$ (where L_{ij} are literals and $k_i \in \mathbb{N}$), then S is equivalent to the †-formula:

$$(\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$$

where:

- $\mathcal{L} \overset{\text{def}}{=} \{\alpha\} \cup \{\beta_{ij} \mid i \in [1..n], j \in [1..k_i]\}$.
- $\mathcal{M} \overset{\text{def}}{=} \mathcal{L} \setminus \{\alpha\}$.
- $\delta(\alpha) \overset{\text{def}}{=} \{\bigvee_{i=1}^{k_i} \beta_{ij} \mid i \in [1..n]\}$.
- $\mu(\beta_{ij}) \overset{\text{def}}{=} L_{ij}$, for any $i \in [1..n], j \in [1..k_i]$.

Thus †-formulae denote uniformly clauses and sets of clauses.

We write $\mathcal{C} \equiv \mathcal{C}'$ if $\mathcal{C}, \mathcal{C}'$ are logically equivalent (i.e. if $\mathcal{S}(\mathcal{C}) \equiv \mathcal{S}(\mathcal{C}')$). We write $\mathcal{C} \sim \mathcal{C}'$ iff $\mathcal{S}(\mathcal{C}) = \mathcal{S}(\mathcal{C}')$ (this implies that $\mathcal{C} \equiv \mathcal{C}'$ but the converse does not hold).

We denote by $\top$ the †-formula $(\{\alpha\}, \emptyset, \alpha, \alpha \mapsto \emptyset, \emptyset)$ and by $\bot$ the †-formula $(\{\alpha\}, \emptyset, \alpha, \alpha \mapsto \{\Box\}, \emptyset)$. Obviously we have $\mathcal{S}(\top) = \emptyset$ ($\top$ is valid) and $\mathcal{S}(\bot) = \{\Box\}$ ($\bot$ is unsatisfiable).

Example 2.3. Let $\mathcal{C} \overset{\text{def}}{=} \{\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu\}$ be a †-formula where:

- $\mathcal{L} \overset{\text{def}}{=} \{\alpha, \beta_1, \beta_2, \beta_3, \beta, \gamma_1, \gamma_2, \gamma\}$.
- $\mathcal{M} \overset{\text{def}}{=} \{\beta_1, \beta_2, \beta_3, \gamma_1, \gamma_2\}$.
- $\delta(\alpha) \overset{\text{def}}{=} \{\beta \vee \gamma\}$, $\delta(\beta) \overset{\text{def}}{=} \{\beta_1, \beta_2, \beta_3\}$, $\delta(\gamma) \overset{\text{def}}{=} \{\gamma_1, \gamma_2\}$.
- $\mu(\beta_i) \overset{\text{def}}{=} p_i$ for any $i = 1, 2, 3$, and $\mu(\gamma_j) \overset{\text{def}}{=} q_j$ for any $j = 1, 2$.

The sets $\{\beta_1, \beta_2, \beta_3\}$ and $\{\gamma_1, \gamma_2\}$ essentially represent conjunctions. It is easy to see that $\mathcal{C}$ is equivalent to the following clause set:

$$\begin{gathered} p_1 \vee q_1 \\ p_2 \vee q_1 \\ p_3 \vee q_1 \\ p_1 \vee q_2 \\ p_2 \vee q_2 \\ p_3 \vee q_2 \end{gathered}$$

Example 2.4. Let $\mathcal{C} \overset{\text{def}}{=} \{\{\alpha, \beta, \beta', \gamma, \gamma', \lambda\}, \{\beta', \gamma', \lambda\}, \alpha, \delta, \mu\}$ be a †-formula where:

- $\delta \overset{\text{def}}{=} \{\alpha \mapsto \{\beta \vee \gamma\}, \beta \mapsto \{\beta', \lambda\}, \gamma \mapsto \{\gamma', \lambda\}\}$.
- $\mu \overset{\text{def}}{=} \{\beta' \mapsto a, \gamma' \mapsto b, \lambda \mapsto c\}$.

$\mathcal{C}$ is equivalent to the following clause set:

$$\{a \vee b, a \vee c, c \vee b, c \vee c\}.$$

The literal c corresponding to the node λ is shared inside the formulae $a \wedge c$ and $b \wedge c$ corresponding to β and γ respectively.

Clearly the size of $\mathcal{S}(\mathcal{C})$ may be exponential w.r.t. the size of $\mathcal{C}$, as evidenced by the following:

Example 2.5. Let $\mathcal{C}_n = (\mathcal{L}_n, \mathcal{M}_n, \alpha, \delta_n, \mu_n)$ be a sequence of †-formulae defined as follows:

- $\mathcal{L}_n = \{\alpha, \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n, \gamma_1, \ldots, \gamma_n\}$.
- $\mathcal{M}_n = \{\beta_1, \ldots, \beta_n, \gamma_1, \ldots, \gamma_n\}$.

- $\delta(\alpha) = \{\alpha_1 \vee \ldots \vee \alpha_n\}$, $\delta(\alpha_i) = \{\beta_i, \gamma_i\}$ for $i = 1, \ldots, n$.
- $\mu(\beta_i) = p_i^1$, $\mu(\gamma_i) = p_i^2$ for $i = 1, \ldots, n$.

The size of $\mathcal{C}_n$ is linear w.r.t. n, but $\mathcal{S}(\mathcal{C}_n) = \{p_1^{i_i} \vee \ldots \vee p_n^{i_n} \mid \forall j \in [1..n], i_j \in \{1, 2\}\}$ contains 2^n distinct clauses.

If δ_1, δ_2 are two functions defined on two disjoint domains $\mathcal{L}_1, \mathcal{L}_2$ respectively, then $\delta_1 \cup \delta_2$ denotes the function defined on the domain $\mathcal{L}_1 \cup \mathcal{L}_2$ as follows: $(\delta_1 \cup \delta_2)(\alpha) \overset{\text{def}}{=} \delta_i(\alpha)$ if $\alpha \in \mathcal{L}_i$.

If δ is a function defined on $\mathcal{L}$ then $\delta\{\alpha \mapsto S\}$ is the function δ' defined on $\mathcal{L} \cup \{\alpha\}$ as follows: $\delta'(\beta) \overset{\text{def}}{=} \delta(\beta)$ if $\beta \neq \alpha$ and $\delta'(\alpha) \overset{\text{def}}{=} S$ (we may have $\alpha \in \mathcal{L}$ or $\alpha \notin \mathcal{L}$).

2.4. A Linear Notation for †-Formulae

The above definition is suitable for mathematical definitions and proofs, but it is not very convenient to use in practice. For the sake of conciseness and readability we introduce a more readable notation for denoting †-formulae. This notation is very close to the one usually used to denote term-graphs [3] for instance.

We write a †-formula as a usual (quantifier-free) formula in negation normal form, constructed on a set of literals using the connectives $\vee$ and $\wedge$. For instance, the †-formula of Example 2.3 can be denoted by the formula

$$(p_1 \wedge p_2 \wedge p_3) \vee (q_1 \wedge q_2).$$

In order to express sharing, we associate a *name* (i.e. a node) to some of the subformulae. This is be done by prefixing the corresponding subformula by the node, as follows: $\alpha{:}(\beta_1{:}a \vee \beta_2{:}b)$. These nodes can be reused afterwards, in order to avoid duplicating the considered subformula. For instance $\alpha{:}(\beta{:}a \vee \beta)$ denote the formula $(a \vee a)$, where the two occurrences of a are shared. Unnamed subformula may be implicitly associated to arbitrary, pairwise different, nodes.

The †-formula in Example 2.4 is denoted by

$$(a \wedge \lambda{:}c) \vee (b \wedge \lambda).$$

The definition of λ is given only once and it can be reused as many times as needed.

This notation is obviously much more readable then the previous one, hence will be used in the forthcoming examples.

Formally, †-formulae can be inductively constructed as follows. If l is a literal, then we also denote by l the †-formula

$$(\{\alpha\}, \{\alpha\}, \alpha, \emptyset, \{\alpha \mapsto l\})$$

where α is a arbitrarily chosen node.

If $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ is a †-formula and $\beta \notin \mathcal{L}$, then $\beta{:}\mathcal{C}$ denotes the †-formula

$$(\mathcal{L} \cup \{\beta\}, \mathcal{M} \cup \{\beta\}, \beta, \delta \cup \{\beta \mapsto \alpha\}, \mu).$$

Finally, if $\mathcal{C}_i = (\mathcal{L}_i, \mathcal{L}'_i, \alpha_i, \delta_i, \mu_i)$ for $i = 1, 2$ are two disjoint †-formulae, then $\mathcal{C}_1 \vee \mathcal{C}_2$ (resp. $\mathcal{C}_1 \wedge \mathcal{C}_2$) denote the †-formula

$$(\mathcal{L}_1 \cup \mathcal{L}_2 \cup \{\alpha\}, \mathcal{L}'_1 \cup \mathcal{L}'_2, \alpha, \delta_1 \cup \delta_2 \cup \{\alpha \mapsto S\})$$

where $S = \{\alpha_1 \vee \alpha_2\}$ (resp $S = \{\alpha_1, \alpha_2\}$) and α is a new node not occurring in $\mathcal{L}_1, \mathcal{L}_2$.

2.5. Transforming †-Formulae

In this section, we introduce some basic transformations operating on †-formulae. These definitions provide useful mathematical tools for handling †-formulae, which will serve as a basis for defining the inference and simplification rules of Section 3 and for proving their soundness.

2.5.1. Replacement of †-Formulae

The first definition allows one to replace a sub-†-formula occurring in a given †-formula $\mathcal{C}$ by a new †-formula.

Definition 2.3. (Replacement) Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ and $\mathcal{C}' = (\mathcal{L}', \mathcal{M}', \alpha', \delta', \mu')$ be two disjoint †-formulae. Let $\beta \in \mathcal{L}$. We denote by $\mathcal{C}[\mathcal{C}']_\beta$ the †-formula

$$(\mathcal{L} \cup \mathcal{L}', (\mathcal{M} \cup \mathcal{M}') \setminus \{\beta\}, \alpha, (\delta \cup \delta')\{\beta \mapsto \alpha'\}, \mu \cup \mu'\}.$$

Note that $\mathcal{C}[\mathcal{C}']_\beta$ is an acylic †-formula since $\mathcal{C}, \mathcal{C}'$ are acyclic and since $\mathcal{L}$ and $\mathcal{L}'$ are disjoint.

If $\mathcal{C}, \mathcal{C}'$ are not disjoint, then $\mathcal{C}[\mathcal{C}']_\beta$ denotes the †-formula $\mathcal{C}[\mathcal{C}'']_\beta$, where $\mathcal{C}''$ is an arbitrary relabeling of $\mathcal{C}'$, disjoint from $\mathcal{C}$.

For instance, $(a \vee (\alpha{:}b \wedge c))[\neg c \wedge d]_\alpha = a \vee (\alpha{:}(\neg c \wedge d) \wedge c)$.

Lemma 2.1. Let $\mathcal{C}, \mathcal{C}'$ be two †-formulae. Let β be a node occurring in $\mathcal{C}$ and let $\mathcal{D} \overset{\text{def}}{=} \mathcal{C}[\mathcal{C}']_\beta$.

Let $\mathcal{I}$ be an interpretation such that $\mathcal{I} \models \mathcal{C}$ and for any ground substitution σ, either $\mathcal{I} \not\models \mathcal{C}_{|\beta}\sigma$ or $\mathcal{I} \models \mathcal{C}'\sigma$. Then $\mathcal{I} \models \mathcal{D}$.

Proof:
The proof follows from the fact that †-formulae denote formulae that are essentially monotone, in that negations only appear on the atomic subformulae. More formally, let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$. Let σ be a ground substitution. Let $\mathcal{I}$ be an interpretation s.t. either $\mathcal{I} \not\models \mathcal{C}_{|\beta}\sigma$ or $\mathcal{I} \models \mathcal{C}'\sigma$. Let $\gamma \in \mathcal{L}$. We prove, by induction on $\preceq_\mathcal{C}$, that if $\mathcal{I} \models \mathcal{C}_{|\gamma}\sigma$ and $\gamma \succeq_\mathcal{C} \beta$ then $\mathcal{I} \models \mathcal{D}_{|\gamma}\sigma$ (we obtain the desired result for $\alpha = \gamma$, since if $\alpha \not\succeq_\mathcal{C} \beta$ then obviously $\mathcal{S}(\mathcal{C})$ does not depend on β, thus $\mathcal{C} \equiv \mathcal{D}$).

- Assume that $\gamma = \beta$. Then $\mathcal{D}_{|\gamma} \equiv \mathcal{C}'$. If $\mathcal{I} \models \mathcal{C}_{|\gamma}\sigma$ we have $\mathcal{I} \models \mathcal{C}_{|\beta}\sigma$ hence $\mathcal{I} \models \mathcal{C}'\sigma$ and $\mathcal{I} \models \mathcal{D}_{|\gamma}\sigma$.

- Assume that $\gamma \succ_\mathcal{C} \beta$. Let $D \in \mathcal{S}(\mathcal{D}_{|\gamma})$. By definition D is either $\mu(\gamma)$ or of the form $D_1 \vee \ldots \vee D_n$ where $\delta(\gamma)$ contains a clause $\alpha_1 \vee \ldots \vee \alpha_n$ s.t. $D_i \in \mathcal{S}(\mathcal{D}_{|\alpha_i})$ for any $i \in [1..n]$. If $D = \mu(\gamma)$ then $D \in \mathcal{S}(\mathcal{C}_{|\gamma})$, thus if $\mathcal{I} \models \mathcal{C}_{|\gamma}\sigma$ we have $\mathcal{I} \models D\sigma$. Otherwise, since $\mathcal{I} \models \mathcal{C}_{|\gamma}\sigma$, hence there exists $i \in [1..n]$ s.t. $\mathcal{I} \models \mathcal{C}_{|\alpha_i}\sigma$. By the induction hypothesis this implies that $\mathcal{I} \models \mathcal{D}_{|\alpha_i}\sigma$. Thus $\mathcal{I} \models D_i\sigma$ and $\mathcal{I} \models D\sigma$.

□

2.5.2. Insertion of †-Formulae

The following notation allows one to insert a clause or a set of clauses at some specific node in a †-formula. The difference with the previous notation is that the clauses corresponding to the node in the initial †-formula are preserved instead of being deleted.

Let $\mathcal{C}, \mathcal{C}'$ be disjoint †-formulae. Let β be a node in $\mathcal{C}$.

We denote by $\mathcal{C}\langle\mathcal{C}'\rangle_\beta$ the †-formula:

$$\mathcal{C}[\mathcal{C}_{|\beta} \wedge \mathcal{C}']_\beta.$$

For instance, $(a \vee (\alpha{:}b \wedge c))\langle\neg c \wedge d\rangle_\alpha = a \vee (\alpha{:}(b \wedge \neg c \wedge d) \wedge c)$.

Lemma 2.2. Let $\mathcal{C}, \mathcal{C}'$ be two †-formulae. Let β be a node in $\mathcal{C}$. Let $\mathcal{I}$ be an interpretation s.t. for any ground substitution σ, either $\mathcal{I} \not\models \mathcal{C}_{|\beta}\sigma$ or $\mathcal{I} \models \mathcal{C}'\sigma$. Then $\mathcal{I} \models \mathcal{C}\sigma$ iff $\mathcal{I} \models \mathcal{C}\langle\mathcal{C}'\rangle_\beta\sigma$.

Proof:
The proof follows from Lemma 2.1. Indeed, we have either $\mathcal{I} \not\models \mathcal{C}_{|\beta}\sigma$ or $\mathcal{I} \models \mathcal{C}_{|\beta}\sigma \wedge \mathcal{C}'\sigma$. □

2.6. Weakening

The following notation allows one to delete, in a given †-formula, the clauses not containing a given literal.

Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ be a †-formula. Let β_1, β_2 be two nodes in $\mathcal{L}$. We write $\beta_1 \bowtie_\mathcal{C} \beta_2$ if there exists $\lambda \preceq_\mathcal{C} \alpha$ and $\zeta_1 \vee \zeta_2 \vee C \in \delta(\lambda)$ s.t. $\zeta_1 \neq \zeta_2$ and $\beta_i \preceq_\mathcal{C} \zeta_i$ for $i = 1, 2$. Informally $\beta_1 \bowtie_\mathcal{C} \beta_2$ iff $\mathcal{S}(\mathcal{C})$ contains a clause of the form $C_1 \vee C_2 \vee D$, where $C_i \in \mathcal{S}(\mathcal{C}_{|\beta_i})$.

Let $\beta \in \mathcal{M}$. We denote by $w_\beta(\mathcal{C})$ the †-formula obtained by replacing by $\top$ each node $\gamma \in \mathcal{L}$ s.t. $\gamma \not\bowtie_\mathcal{C} \beta$.

Clearly, $\mathcal{S}(w_\beta(\mathcal{C}))$ contains all the clauses in $\mathcal{S}(\mathcal{C})$ that contain $\mu(\beta)$. However, not all the clauses in $w_\beta(\mathcal{C})$ contain $\mu(\beta)$.

Proposition 2.1. Let $\mathcal{C}$ be a †-formula. Let $\mathcal{D} = w_\beta(\mathcal{C})$. Then $\mathcal{S}(\mathcal{D}) \subseteq \mathcal{S}(\mathcal{C})$.

Proof:
Immediate, since $\mathcal{D}$ is obtained from $\mathcal{C}$ by replacing some of the nodes by *true*. □

Example 2.6. Let $\mathcal{C}$ be the †-formula (in linear notation): $(\alpha{:}p \wedge \beta{:}q) \vee (\gamma{:}r \wedge \zeta{:}s)$. We have $\alpha \bowtie_\mathcal{C} \gamma$ and $\alpha \bowtie_\mathcal{C} \zeta$. But $\alpha \not\bowtie_\mathcal{C} \beta$. Thus $w_\alpha(\mathcal{C}) = (\alpha{:}p \wedge \top) \vee (\gamma{:}r \wedge \zeta{:}s) \equiv \alpha{:}p \vee (\gamma{:}r \wedge \zeta{:}s)$.

3. The †-Resolution Calculus

We are now in position to define our extended resolution calculus. We adapt existing rules (resolution, factorization and usual simplification rules such as subsumption) in order to handle †-formulae, and we introduce some essentially new simplification rules, that are specific to †-formulae.

3.1. Simplification rules

First, we introduce some simplification rules which are useful for reducing the size of the †-formulae. We shall prove that these rules preserve the semantics of the considered †-formulae.

3.1.1. Reduction

The first simplification rule allows one to remove useless transitions[1], i.e. transitions of the form $\gamma \mapsto \{C\}$, where γ is a node and C a Λ-clause. Such transitions can easily be discarded because γ can be directly replaced by C (this is possible because this does not increase the number of clauses occurring at each node).

For instance, the †-formula

$$\mathcal{C} = (\alpha{:}(\beta{:}a \vee \gamma{:}b) \vee c) \wedge (\alpha \vee d),$$

can be reduced to:

$$\mathcal{C} = (\beta{:}a \vee \gamma{:}b \vee c) \wedge (\beta{:}a \vee \gamma{:}b \vee d).$$

The node α is useless.

We need to introduce a notation. Let σ be a function from Λ to the set of Λ-clauses. σ is extended to operate on Λ-clauses and sets of Λ-clauses using the relations: $\sigma(L_1 \vee \ldots \vee L_n)\sigma \overset{\text{def}}{=} \sigma(L_1) \vee \ldots \vee \sigma(L_n)$ and $\sigma(S) \overset{\text{def}}{=} \{C\sigma \mid C \in S\}$.

If δ is a partial function from Λ to sets of Λ-clauses, $\sigma(\delta)$ denotes the function defined as follows: $\sigma(\delta)(\lambda) \overset{\text{def}}{=} \sigma(\delta(\lambda))$.

The reduction rule is defined as follows:

$$\frac{(\mathcal{L} \cup \{\gamma\}, \mathcal{M}, \alpha, \delta \cup \{\gamma \mapsto \{C\}\}, \mu)}{(\mathcal{L}, \mathcal{M} \setminus \{\gamma\}, \sigma(\alpha), \sigma(\delta), \mu)}$$

where $\sigma = \{\gamma \mapsto C\}$ and either $\alpha \neq \gamma$ or C is a unit clause[2].

Proposition 3.1. Let $\mathcal{C}$ be a †-formula and let $\mathcal{C}'$ be a †-formula obtained by applying the reduction rule on $\mathcal{C}$. $\mathcal{S}(\mathcal{C}) = \mathcal{S}(\mathcal{C}')$.

Proof:
Let $\mathcal{C} = (\mathcal{L} \cup \{\gamma\}, \mathcal{M}, \alpha, \delta \cup \{\gamma \mapsto \{D\}\}, \mu\}$ and $\mathcal{C}' = (\mathcal{L}, \mathcal{M} \setminus \{\gamma\}, \sigma(\alpha), \sigma(\delta), \mu)$ where $\sigma = \{\alpha \mapsto D\}$.

We prove, by induction on $\prec_{\mathcal{C}}$, that for any $\lambda \in \mathcal{L}$, $\mathcal{S}(\mathcal{C}_{|\lambda}) = \mathcal{S}(\mathcal{C}'_{|\sigma(\lambda)})$.

If $\lambda = \gamma$, then by definition since $(\delta \cup \{\gamma \mapsto \{D\}\})(\gamma) = \{D\}$, $C \in \mathcal{S}(\mathcal{C}_{|\lambda})$ iff one of the following holds:

- Either $\gamma \in \mathcal{M}$ and $C = \mu(\gamma)$.
- Or $C \in \mathcal{S}(\mathcal{C}_{|D})$.

Since $\gamma \in$ *dom*(δ) we have $\gamma \notin \mathcal{M}$. Hence we have $C \in \mathcal{S}(\mathcal{C}_{|D})$. Thus we have $\mathcal{S}(\mathcal{C}_{|\lambda}) = \mathcal{S}(\mathcal{C}_{|D})$. By the induction hypothesis, we have $\mathcal{S}(\mathcal{C}_{|D}) = \mathcal{S}(\mathcal{C}'_{|\sigma(D)}) = \mathcal{S}(\mathcal{C}'_{|D})$ (for any $\beta \in D$, since $\beta \prec_{\mathcal{C}} \gamma$, we have $\beta \neq \gamma$). Since $\sigma(\gamma) = D$ this implies that $\mathcal{S}(\mathcal{C}'_{|\sigma(\gamma)}) = \mathcal{S}(\mathcal{C}_{|\gamma})$.

[1]The *transition function* is the function δ mapping nodes to sets of $\mathcal{L}$-clauses, as defined in Definition 2.1.

[2]If C is not a unit clause and $\alpha = \gamma$ then γ cannot be eliminated because it is the root node. If C is unit and $\alpha = \gamma$ then the (unique) node in C becomes the new root node.

Now assume that $\lambda \neq \gamma$. Then $\sigma(\lambda) = \lambda$. $C \in \mathcal{S}(\mathcal{C}_{|\lambda})$ iff either $C = \mu(\lambda)$ or $C = C_1 \vee \ldots \vee C_k$ where $(\alpha_1 \vee \ldots \alpha_k) \in \delta(\lambda)$ and for any $i \in [1..k]$, $C_i \in \mathcal{S}(\mathcal{C}_{|\alpha_i})$. By the induction hypothesis, this last condition is equivalent to: $C_i \in \mathcal{S}(\mathcal{C}'_{|\sigma(\alpha_i)})$. But $\sigma(\delta)(\lambda) = \sigma(\alpha_1) \vee \ldots \sigma(\alpha_k) \in \sigma(\delta)(\lambda)$ iff $\alpha_1 \vee \ldots \vee \alpha_k \in \delta(\lambda)$, thus $C \in \mathcal{S}(\mathcal{C}_{|\lambda})$ iff $C \in \mathcal{S}(\mathcal{C}'_{|\sigma(\lambda)})$. □

3.1.2. Sharing

The above rule allows one to merge identical nodes (i.e. nodes corresponding to the same literal).

$$\frac{(\mathcal{L} \cup \{\gamma\}, \mathcal{M}, \alpha, \delta, \mu)}{(\mathcal{L}, \mathcal{M} \setminus \{\gamma\}, \sigma(\alpha), \sigma(\delta), \mu)}$$

If $\sigma = \{\gamma \mapsto \beta\}$ and $\gamma, \beta \in \mathcal{M}$ and $\mu(\gamma) = \mu(\beta)$.

For instance, the †-formula $(a \wedge b) \vee (b \wedge c) \wedge (a \wedge c)$ can be reduced to $(\alpha{:}a \wedge \beta{:}b) \vee (\beta \wedge \zeta{:}c) \vee (\alpha \wedge \zeta)$.

Proposition 3.2. Let $\mathcal{C}$ be a †-formula and let $\mathcal{C}'$ be a †-formula obtained by applying the sharing rule on $\mathcal{C}$. $\mathcal{S}(\mathcal{C}) = \mathcal{S}(\mathcal{C}')$.

Proof:
The application conditions on the rule ensures that $\mathcal{S}(\mathcal{C}_{|\gamma}) = \mathcal{S}(\mathcal{C}_{|\beta})$ (since $\mu(\gamma) = \mu(\beta)$). Then it is easy to prove, by a straightforward induction on $\prec_{\mathcal{C}}$ (similar to the one of the proof of Proposition 3.1), that we have $\mathcal{S}(\mathcal{C}_{|\lambda}) = \mathcal{S}(\mathcal{C}'_{|\lambda})$, for any node λ. □

3.1.3. Merging

The next rule allows one to reuse existing nodes when possible for denoting clause sets occurring in a given †-formula. This avoids duplication of information.

$$\frac{(\mathcal{L}, \mathcal{M}, \alpha, \delta \cup \{\gamma \mapsto S \cup S_1, \gamma' \mapsto S \cup S_2\}, \mu)}{(\mathcal{L} \cup \{\gamma''\}, \mathcal{M}, \alpha, \delta \cup \{\gamma \mapsto \{\gamma''\} \cup S_1, \gamma' \mapsto \{\gamma''\} \cup S_2, \gamma'' \mapsto S\}, \mu)}$$

where γ'' is a node not occurring in $\mathcal{L}$.

For instance the †-formula $(\alpha{:}a \wedge \beta{:}b \wedge c) \vee \delta{:}(\alpha \wedge \beta)$ can be transformed into $(\delta \wedge c) \vee \delta{:}(\alpha{:}a \wedge \beta{:}b)$.

Proposition 3.3. Let $\mathcal{C}$ be a †-formula and let $\mathcal{C}'$ be a †-formula obtained by applying the merging rule on $\mathcal{C}$. $\mathcal{S}(\mathcal{C}) = \mathcal{S}(\mathcal{C}')$.

Proof:
$\mathcal{C}$ and $\mathcal{C}'$ are respectively of the form: $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu\}$ and $\mathcal{C}' = (\mathcal{L} \cup \{\gamma''\}, \mathcal{M}, \alpha, \delta', \mu)$ where $\delta(\gamma) = S \cup S_1$, $\delta(\gamma') = S \cup S_2$, $\delta'(\gamma) = \{\gamma''\} \cup S_1$, $\delta'(\gamma) = \{\gamma''\} \cup S_2$, $\delta'(\gamma'') = S$. Let C be a clause. C occurs in $\mathcal{C}_{|\gamma}$ iff there exists a clause $D \in \delta(\gamma)$ s.t. $C \in \mathcal{S}(\mathcal{C}_{|D})$. $D \in \delta(\gamma)$ iff either $D \in S$, or $D \in S_1$. By definition, since δ, δ' only differ on $\gamma, \gamma', \gamma''$, and $\mathcal{C}$ is acyclic, for any clause $D \in S \cup S_1$ we have $\mathcal{C}_{|D} = \mathcal{C}'_{|D}$. Thus $\mathcal{S}(\mathcal{C}_{|S}) = \mathcal{S}(\mathcal{C}'_{|S}) = \mathcal{S}(\mathcal{C}'_{|\gamma''})$. Moreover $\mathcal{S}(\mathcal{C}_{|S_1}) = \mathcal{S}(\mathcal{C}'_{|S_1})$. Hence $\mathcal{C}_{|\gamma} = \mathcal{C}'_{|\gamma}$. Thus, according to Lemma 2.1, $\mathcal{C}_{|\gamma}$ can be replaced by $\mathcal{C}'_{|\gamma}$. The same holds for γ'. □

3.1.4. Elimination

The next rule allows one to eliminate useless nodes (i.e. nodes corresponding to empty clause sets).

$$\frac{(\mathcal{L} \cup \{\beta\}, \mathcal{M}, \alpha, \delta \cup \{\beta \mapsto \emptyset\}, \mu)}{(\mathcal{L}, \mathcal{M}, \alpha, \delta', \mu)}$$

If $\delta'(\lambda) \overset{def}{=} \{C \mid C \in \delta(\lambda), \beta \notin C\}$.

For instance, the †-formula $a \vee [b \wedge (c \vee \top)]$ is reducible to $a \vee b$. Indeed, $c \vee \top$ is reducible to $\top$, thus $b \wedge (c \vee \top)$ is reducible to b.

Proposition 3.4. Let $\mathcal{C}$ be a †-formula and let $\mathcal{C}'$ be a †-formula obtained by applying the elimination rule on $\mathcal{C}$. $\mathcal{S}(\mathcal{C}) = \mathcal{S}(\mathcal{C}')$.

Proof:
The application conditions on the rule ensures that $\mathcal{S}(\mathcal{C}_{|\beta}) = \emptyset$ (since $\delta(\beta) = \emptyset$). Thus if a $\mathcal{L}$-clause $\alpha_1 \vee \ldots \vee \alpha_k$ contains β, the set of clauses of the form $C_1 \vee \ldots \vee C_k$ s.t. $C_i \in \mathcal{S}(\mathcal{C}_{|\alpha_i})$ for any $i \in [1..n]$ must be empty. Hence removing such clauses from the $\mathcal{L}$-clause sets $\delta(\lambda)$ does not change the obtained clause set. □

3.1.5. Internal Subsumption

The next rule is more complicated because it affects the clause set $\mathcal{S}(\mathcal{C})$ corresponding to the †-formula $\mathcal{C}$ at hand. The idea is to remove redundant (i.e. subsumed) $\mathcal{L}$-clauses occurring in one of the clause sets $\delta(\lambda)$, where λ is a node in $\mathcal{C}$. To this purpose, we extend the usual notion of subsumption to †-formulae.

From a theoretical point of view we could define the subsumption relation as follows: $\mathcal{C}$ subsumes $\mathcal{C}'$ iff $\mathcal{S}(\mathcal{C})$ subsumes $\mathcal{S}(\mathcal{C}')$. But this definition is not very practical, because it would urge us to explicitly compute the sets $\mathcal{S}(\mathcal{C})$ and $\mathcal{S}(\mathcal{C}')$ for checking subsumption. Since these sets are exponentially bigger than $\mathcal{C}$ and $\mathcal{C}'$ this would be highly inefficient (and all the advantages of using †-formulae would be lost). Thus we introduce another (stronger) definition.

Let S, S' be two sets of Λ-clauses. We write $S \preccurlyeq^{sub} S'$ (S subsumes S') iff for any clause $C \in S'$, there exists a clause $C' \in S$ s.t. $C' \subseteq C$.

Definition 3.1. (†-Subsumption) Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ and $\mathcal{C}' = (\mathcal{L}', \mathcal{M}', \alpha', \delta', \mu')$.

We write $\mathcal{C} \preccurlyeq^{sub} \mathcal{D}$ ($\mathcal{C}$ †-subsumes $\mathcal{D}$) iff for any $\beta \in \mathcal{L}'$ one of the following conditions hold:

- $\delta(\beta) = \{\square\}$.
- $\delta'(\beta) = \emptyset$.
- $\beta \in \mathcal{M}'$ $\beta \in \mathcal{M}$ and $\mu(\beta) = \mu'(\beta)$.
- $\beta \notin \mathcal{M}'$, $\beta \notin \mathcal{M}$ and $\delta(\beta) \preccurlyeq^{sub} \delta'(\beta)$.

Proposition 3.5. If $\mathcal{C} \preccurlyeq^{sub} \mathcal{D}$ then $\mathcal{S}(\mathcal{C}) \preccurlyeq^{sub} \mathcal{S}(\mathcal{D})$.

Proof:
This is an immediate consequence of the definition of $\mathcal{S}(\mathcal{C})$. $\square$

According to Proposition 3.5, if $\mathcal{C}$ is a †-formula, and $\mathcal{D} \wedge \mathcal{D}'$ is a sub-†-formula occurring in $\mathcal{C}$ s.t. $\mathcal{D} \preccurlyeq^{sub} \mathcal{D}'$ then we have $\mathcal{S}(\mathcal{D} \wedge \mathcal{D}') = \mathcal{S}(\mathcal{D}) \cup \mathcal{S}(\mathcal{D}') \equiv \mathcal{S}(\mathcal{D})$ (since $\mathcal{S}(\mathcal{D}) \preccurlyeq^{sub} \mathcal{S}(\mathcal{D}')$). Thus $\mathcal{D}'$ is redundant and can be eliminated. This is the purpose of the following rule. It also takes into account the fact that some of the variables in $\mathcal{D}$ may be instantiated, i.e. one must compute a substitution σ s.t. $\mathcal{D}\sigma \preccurlyeq^{sub} \mathcal{D}'$. However, an important difference with the usual case is that some of the variables in $\mathcal{D}$ may occur in the context in which $\mathcal{D} \wedge \mathcal{D}'$ occurs. These variables cannot be instantiated during subsumption tests, because their values depend on the context. Assume for instance that $\mathcal{C} = p(x) \vee (q(x) \wedge q(a))$. Clearly there exists a substitution $\sigma = \{x \mapsto a\}$ s.t. $q(x)\sigma \preccurlyeq^{sub} q(a)$. However, $q(a)$ is not redundant because σ affects a variable x occurring in the context of $q(x) \wedge q(a)$ (namely in $p(x)$). Removing $q(a)$ from the above †-formula would be clearly incorrect (i.e. would change the semantics of the †-formula). In contrast, if we consider the †-formula $p(x) \vee (q(y) \vee q(a))$ then $q(a)$ is redundant and can be removed because the substitution $y \mapsto a$ does not affect the variable x.

The next definition formalizes this idea: Let E be a set of variables (possibly empty). We write $\mathcal{C} \preccurlyeq^{sub}_{E} \mathcal{D}$ iff there exists a substitution σ s.t. *dom*$(\sigma) \cap E = \emptyset$ and $\mathcal{C}\sigma \preccurlyeq^{sub} \mathcal{D}$.

Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ be a †-formula. If β is a $\mathcal{L}$-clause, we denote by $\mathcal{CV}_{\beta}(\mathcal{C})$ the set of variables x s.t. x occurs in a literal $\mu(\beta')$ for some $\beta' \in \mathcal{M}$ and $\beta' \bowtie_{\mathcal{C}} \beta$.

Intuitively $\mathcal{CV}_{\beta}(\mathcal{C})$ denotes the set of variables occurring in the "context" of $\mathcal{C}_{|\beta}$.

The internal subsumption rule is defined as follows:

$$\frac{(\mathcal{L}, \mathcal{M}, \alpha, \delta \cup \{\beta \mapsto S \cup \{C\}\}, \mu)}{(\mathcal{L}, \mathcal{M}, \alpha, \delta \cup \{\beta \mapsto S\}, \mu)}$$

If S contains a clause D s.t. $\mathcal{C}_{|D} \preccurlyeq^{sub}_{E} \mathcal{C}_{|C}$, where $E = \mathcal{CV}_{\beta}(\mathcal{C})$.

Lemma 3.1. Let $\mathcal{C}$ be a †-formula and let $\mathcal{C}'$ be a †-formula obtained by applying the subsumption rule on $\mathcal{C}$. $\mathcal{S}(\mathcal{C}) \equiv \mathcal{S}(\mathcal{C}')$.

Proof:

We have obviously $\mathcal{S}(\mathcal{C}') \subseteq \mathcal{S}(\mathcal{C})$ (since the rule only deletes clauses in the clause set corresponding to the †-formula β). Thus $\mathcal{C} \models \mathcal{C}'$.

Now, let $\mathcal{I}$ be a model of $\mathcal{C}'$. Let θ be a ground substitution of the variables in $\mathcal{C}$. Let $\gamma = \sigma\theta$.

We have $\mathcal{I} \models \mathcal{C}'\gamma$. Assume that $\mathcal{I} \models \mathcal{C}'_{|\beta}\gamma$. Then $\mathcal{I} \models \mathcal{C}'_{|D}\gamma = \mathcal{C}_{|D}\gamma$, hence by Proposition 3.5, we have $\mathcal{I} \models \mathcal{C}_{|C}\theta$.

Thus by Lemma 2.1, we have $\mathcal{I} \models \mathcal{C}'\gamma\langle\mathcal{C}_{|C}\theta\rangle_{\beta}$. But by definition of $\mathcal{CV}_{\beta}(\mathcal{C})$, all the variables from *dom*(σ) occurring in $\mathcal{C}$ must occur in $\mathcal{C}_{|\beta}$. Thus $\mathcal{C}'\gamma\langle\mathcal{C}_{|C}\theta\rangle_{\beta} = \mathcal{C}'\theta\langle\mathcal{C}_{|C}\theta\rangle_{\beta}$.

Hence $\mathcal{I} \models \mathcal{C}'\theta\langle\mathcal{C}_{|C}\theta\rangle_{\beta} = \mathcal{C}\theta$. $\square$

We have the following:

Proposition 3.6. If $\square \in \mathcal{S}(\mathcal{C})$ then $\mathcal{C}$ can be reduced to $\bot$ by the internal subsumption rule.

Proof:
The proof is by a straightforward induction on $\prec_{\mathcal{C}}$. $\square$

3.2. Unit Simplification

The idea of the last simplification rule is to use the unit clauses occurring in a given †-formula $\mathcal{C}$ in order to simplify $\mathcal{C}$. If a literal l holds, then ϕ is equivalent to $\phi\{l \rightarrow \mathit{true}\}$. Thus $l \wedge \phi \equiv l \wedge \phi'$ where ϕ' is obtained from ϕ by replacing each occurrence of l by $\top$ and each occurrence of l^c by $\bot$. This idea can be formalized by the following rule:

$$\frac{\mathcal{C} : (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)}{\mathcal{C}[\mu(\beta) \wedge \mathcal{C}_{|\lambda}[\bot]_\gamma[\top]_\beta]_\lambda}$$

If $\lambda \in \mathcal{L}$, and $\delta(\lambda)$ contains a clause $\{\beta\}$, where $\beta \in \mathcal{M}$ and $\mu(\gamma) = \mu(\beta)^c$.

For instance, $\alpha : a \wedge ((\alpha \wedge b) \vee c \vee \neg a)$ can be reduced to $\alpha{:}a \wedge (b \vee c)$.

Proposition 3.7. Let $\mathcal{C}$ be a †-formula. Let $\mathcal{C}'$ be a †-formula obtained by applying the unit simplifcation rule on $\mathcal{C}$. $\mathcal{C}' \equiv \mathcal{C}$.

Proof:
The proof follows immediately from Lemma 2.1 and from the above remarks (we have $\mathcal{C}_{|\lambda} \models l$ hence $\mathcal{C}_{|\lambda} \equiv l \wedge \mathcal{C}_{|\lambda}[\bot]_\gamma[\top]_\beta$). □

3.3. Reduced †-Formulae

Lemma 3.2. Let $\mathcal{C}$ be a †-formula. The nondeterministic application of the simplification rule terminates on $\mathcal{C}$.

Proof:
Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$. It is easy to see that all the simplification rules *except the unit simplification rule* strictly decrease the size of the considered †-formula[3].

The unit simplification rule decreases the *number of occurrences* of the literals in $\mathcal{C}$ (i.e. the number of distinct paths – or positions – from α to the considered literals). It is clear that no simplification rule can increase this number. □

Definition 3.2. (Reduced †-Formulae) A †-formula is said to be *reduced* iff it is irreducible by the simplification rules. The †-formula obtained from a †-formula $\mathcal{C}$ by nondeterministic application of the simplification rules (without the unit simplification rule) is called the *reduced form* of $\mathcal{C}$.

3.4. Inference Rules

In this section we extend the usual inference rules, namely the resolution and factorization rules, in order to handle †-formulae instead of clauses.

[3]This is not the case of the unit simplification rule, due to the fact that the nodes in $\mathcal{C}_{|\lambda}$ can occur elsewhere in $\mathcal{C}$. Thus the application of the rule may *increase* the number of nodes, by "duplicating" them.

3.4.1. †-Resolution

Assume we want to apply the resolution rule between two †-formulae $\mathcal{D}$ and $\mathcal{D}'$ on two literals l, l^c occurring respectively at two nodes β, β'. We proceed as follows. First we remove from $\mathcal{D}'$ (some of) the clauses not containing l^c (using the operator $\mathcal{D}' \to w_{\beta'}(\mathcal{D}')$ defined above) and we replace l^c by $\perp$. This yields a new †-formula $\mathcal{D}''$. Then we insert $\mathcal{D}''$ at the node β in $\mathcal{D}$. All the clauses added into $\mathcal{S}(\mathcal{D})$ during this process are resolvents of clauses initially occurring in $\mathcal{S}(\mathcal{D})$ and $\mathcal{S}(\mathcal{D}')$ respectively.

Clearly, one application of the †-resolution rule corresponds to several applications of the usual resolution rule. For instance given the †-formulae $(a \wedge b) \vee c$ and $\neg c \vee (a' \wedge b')$ we derive: $(a \wedge b) \vee (c \wedge a' \wedge b')$. The set of clauses corresponding to the initial †-formulae are $\{a \vee c, b \vee c\}$ and $\{\neg c \vee a', \neg c \vee b'\}$ respectively, and the set of clauses corresponding to the obtained †-formula is $\{a \vee c, b \vee c, a \vee a', a \vee b', b \vee a', b \vee b'\}$. Note that with our technique, this clause set is obtained in only one resolution step, whereas 4 steps are needed if ordinary resolution is used.

Definition 3.3 above formalizes this idea. The resolution is "internal" in the sense that it applies to a pair of †-formulae occurring at a node λ inside a given †-formula. This is due to the fact that in our approach, a set of †-formulae is itself denoted by a †-formula.

Naturally, we also need to take into account the fact that some of the variables can be instantiated during the process (using unification). As for the subsumption rule, we have to ensure that the variables occurring in the context are not instantiated during the process. If it is the case then we need to apply the resolution rule at a higher level in the †-formula – in the worst case at the root position (in which the context is empty). As for the usual resolution rule, some of the variables (those not occurring in the context) are renamed prior to the application of the rule in order to avoid conflicts on the variable names (e. g. if we try to apply the resolution rule on $p(x) \wedge \neg p(f(x))$).

We need to introduce some further notations. Let E be a set of variables. A substitution σ is said to be a *E-unifier* of two literal l, l' iff we have $l\sigma = l'\sigma$ and $x\sigma = x$ for any $x \in E$. As usual, two literals l, l' E-unifiable have a most general E-unifier (unique up to a renaming).

A *E-renaming* is a bijective substitution σ s.t. $x\sigma \in \mathcal{V}$ for any $x \in \mathcal{V}$ and $x\sigma = x$ if $x \in E$.

A †-formula $\mathcal{D}$ is said to be a *E-renaming* of $\mathcal{C}$ iff there exists a E-renaming σ s.t. $\mathcal{D} = \mathcal{C}\sigma$.

Definition 3.3. (†-Resolution) Let $\mathcal{C}$ be a †-formula. Let λ be a node in $\mathcal{C}$ and let C, C' be two Λ-clauses in $\delta(\lambda)$.

Let $\mathcal{D}, \mathcal{D}'$ be two $\mathcal{CV}_\lambda(\mathcal{C})$-renamings of $\mathcal{C}_{|C}$ and $\mathcal{C}_{|C'}$ respectively, by fresh, pairwise distinct, variables, not occurring in $\mathcal{C}$.

Let l, l' be two literals s.t. there exists β, β' occurring in $\mathcal{D}, \mathcal{D}'$ respectively s.t. $\mu'(\beta) = l$ and $\mu''(\beta') = l'$.

Let σ be the most general $\mathcal{CV}_\lambda(\mathcal{C})$-unifier of l^c, l'.

The †-formula
$$\mathcal{R} = \mathcal{C}\langle \mathcal{D}\langle w_{\beta'}(\mathcal{D}')[\perp]_{\beta'}\rangle_\beta \sigma\rangle_\lambda$$
is called an *internal †-resolvent* of $\mathcal{C}$ (w.r.t. the pair of nodes β, β' and the unifying substitution σ).

If the reduced form of $\mathcal{D}\langle w_{\beta'}(\mathcal{D}')[\perp]_{\beta'}\rangle_\beta \sigma$ is subsumed by $\mathcal{C}_{|\lambda}$, then this application of the resolution rule is said to be *redundant* (in this case the obtained †-formula can be reduced to the original one by internal subsumption).

It is worthwhile to mention that if $\sigma = id$, then the obtained resolvent $\mathcal{D}\langle w_{\beta'}(\mathcal{D}')[\perp]_{\beta'}\rangle_\beta \sigma$ subsumes the †-formula $\mathcal{C}_{|C}$ (thus $\mathcal{C}_{|C}$ can be removed).

The reader should note that the †-resolution rule, in contrast to the usual resolution rule, is dissymmetric: the resolvent of $\mathcal{C}$ and $\mathcal{C}'$ is distinct from the resolvent of $\mathcal{C}'$ and $\mathcal{C}$. For instance the resolvent of $p \vee q$ and $\neg q \vee r$ is $p \vee (q \wedge r)$, and the resolvent of $\neg q \vee r$ and $p \vee q$ is $(\neg q \wedge p) \vee q$. As we shall see, *only one of these two resolvents is needed for completeness*.

Lemma 3.3. Let $\mathcal{C}$ be a †-formula. Let $\mathcal{R}$ be a resolvent of $\mathcal{C}$, w.r.t. two nodes β, β' and a unifying substitution σ. $\mathcal{R} \equiv \mathcal{C}$.

Proof:
Obviously, we have $\mathcal{S}(\mathcal{C}) \subseteq \mathcal{S}(\mathcal{R})$ hence $\mathcal{C}$ is a logical consequence of $\mathcal{R}$.

Let $\mathcal{I}$ be an interpretation satisfying $\mathcal{C}$.

Let $\mathcal{C}' = \mathcal{C}\langle \mathcal{D} \wedge \mathcal{D}' \rangle_\lambda \sigma$. Let $C \in \mathcal{S}(\mathcal{C}')$. If C is not an instance of a clause in $\mathcal{S}(\mathcal{C})$, then C is of the form $C_1\sigma \vee C_2\sigma$, where $C_2\sigma \in \mathcal{S}(\mathcal{D} \wedge \mathcal{D}')$ and for any $C_3 \in \mathcal{S}(\mathcal{C}_{|\lambda})$ we have $C_1 \vee C_3 \in \mathcal{S}(\mathcal{C})$. Since σ is a $\mathcal{CV}_\lambda(\mathcal{C})$ unifier, we have $C_1\sigma = C_1$. Let us assume, w.l.o.g., that $C_2 \in \mathcal{S}(\mathcal{D})$ (the case where $C_2 \in \mathcal{S}(\mathcal{D}')$ is similar). By definition there exists a substitution θ s.t. $\mathcal{C} = \mathcal{D}\theta$. Moreover, since θ is a $\mathcal{CV}_\lambda(\mathcal{C})$-renaming, we have $C_1\theta = C_1$. We have $C_2 \in \mathcal{S}(\mathcal{D})$, thus $C_2\theta \in \mathcal{S}(\mathcal{C})$. Thus $C_1 \vee C_2\theta \in \mathcal{S}(\mathcal{C})$. But $C_1 \vee C_2\theta = (C_1 \vee C_2)\theta \preccurlyeq^{sub} (C_1 \vee C_3)\sigma$ (since θ is a renaming). Hence $\mathcal{C}'$ is a logical consequence of $\mathcal{C}$ and $\mathcal{I} \models \mathcal{C}'$.

$\mathcal{R}$ is of the form

$$\mathcal{C}\langle \mathcal{D}\langle w_{\beta'}(\mathcal{D}')[\bot]_{\beta'} \rangle_\beta \sigma \rangle_\lambda$$

where $\mathcal{D}, \mathcal{D}'$ are defined as in Definition 3.3.

Let $\mathcal{R}' = \mathcal{C}'\langle \mathcal{D}\langle w_{\beta'}(\mathcal{D}')[\bot]_{\beta'} \rangle_\beta \rangle_\lambda$. We have $\mathcal{S}(\mathcal{R}) \subseteq \mathcal{S}(\mathcal{R}')$. We show that $\mathcal{I} \models \mathcal{S}(\mathcal{R}')$.

Let θ be an instance of $\mathcal{R}'$. We have $l^c\theta = l'\theta$. Assume that $\mathcal{I} \models \mathcal{D}, \mathcal{D}'$.

Assume that $\mathcal{I} \models l\theta$. Then by Proposition 2.1, $\mathcal{I} \models w_{\beta'}(\mathcal{D}')$. By Lemma 2.1, this implies that if $\mathcal{I} \models w_{\beta'}(\mathcal{D}')[\bot]_{\beta'}$.

Thus $\mathcal{I} \models l\theta \Rightarrow \mathcal{I} \models w_{\beta'}(\mathcal{D}')[\bot]_{\beta'}$. But then by Lemma 2.2, this implies that $\mathcal{I} \models \mathcal{D}\langle w_{\beta'}(\mathcal{D}')[\bot]_{\beta'} \rangle_\beta$.

Thus $\mathcal{I} \models \mathcal{D}, \mathcal{D}' \Rightarrow \mathcal{I} \models \mathcal{D}\langle w_{\beta'}(\mathcal{D}')[\bot]_{\beta'} \rangle_\beta$.

By applying again Lemma 2.2 we conclude that $\mathcal{I} \models \mathcal{R}'$. □

Example 3.1. We consider the following †-formula.

$$\mathcal{C} = p(x) \vee \lambda{:}[(\neg q(x,y) \vee p(y)) \wedge (q(x, f(y)) \vee (r(y) \wedge r(f(y))))].$$

We apply the resolution rule on the node λ, and the clauses $\neg q(x,y) \vee r(y)$ and $q(x, f(y)) \vee (r(y) \wedge r(f(y)))$. First we compute two $\mathcal{CV}_\lambda(\mathcal{C})$-renamings of these two clauses. We obtain $\neg q(x,y') \vee p(y')$ and $q(x, f(y'')) \vee (r(y'') \wedge r(f(y'')))$. Note that x is not renamed since this variable occurs in $\mathcal{CV}_\lambda(\mathcal{C})$. We unify $q(x,y')$ and $q(x, f(y'')$ yielding the substitution $\sigma : \{y' \mapsto f(y'')\}$. Then we replace the literal $q(x, f(y''))$ in $q(x, f(y'')) \vee (r(y'') \wedge r(f(y''))$ by $\bot$ yielding $r(y'') \wedge r(f(y''))$. We insert the obtained †-formula into $\neg q(x,y') \vee p(y')$, at the node corresponding to the literal $\neg q(x,y')$. This yields:

$$\beta{:}(\neg q(x,y') \wedge r(y'') \wedge r(f(y'')) \vee p(y')).$$

Finally we apply the substitution σ and we insert the corresponding †-formula into $\mathcal{C}$ at the node λ. This yields:

$$\mathcal{C} = p(x) \vee \lambda : [(\neg q(x,y) \vee p(y)) \wedge (q(x, f(y)) \vee (r(y) \wedge r(f(y)))) \wedge \mathcal{D}]$$

where

$$\mathcal{D} = \beta{:}(\neg q(x, f(y'')) \wedge r(y'') \wedge r(f(y''))) \vee p(f(y'')).$$

Note that the rule would still be correct if $w_{\beta'}(\mathcal{D}')$ were replaced by $\mathcal{D}'$ in the above definition. The next example shows the interest of applying the operator $w_{\beta'}(\mathcal{D}')$.

Example 3.2. Let $\mathcal{C} = a \vee \beta{:}b$ and $\mathcal{C}' = (\beta'{:}\neg b \vee a') \wedge c$. Let us apply the †-resolution rule on $\mathcal{C}, \mathcal{C}'$. We have $w_{\beta'}(\mathcal{C}') = (\beta'{:}\neg b \vee a')$. Then $w_{\beta'}(\mathcal{C}')[\bot]_{\beta'}$ if a'. Thus the obtained †-formula is $\mathcal{C}\langle a' \rangle_\beta = a \vee (b \wedge a')$.

If we replace $w_{\beta'}(\mathcal{C}')$ by $\mathcal{C}'$, then we obtain $a \vee (b \wedge a' \wedge c)$. The result is correct, but the corresponding clause set contains a clause $a \vee c$ which is redundant (since the clause c already occurs in the clause set corresponding to $\mathcal{C}'$).

3.4.2. Ordering Restriction of †-Resolution

Restriction strategies are essential for the efficiency of inference systems. Ordered resolution aims at reducing the search space (more precisely the branching factor) of the rule by preventing the application of the resolution rule on literals that are non maximal in their parent clauses (according to some fixed ordering $<$ between literals). This principle can be extended to †-formulae. However, since a †-formula corresponds to a set of clauses, the considered literals l, l' may be maximal in some of the clauses and non maximal in the others. Thus, we need to delete from this set, the clauses in which l (resp. l') is non maximal *before* applying the resolution rule. Since a given †-formula can yield an exponential number of distinct clauses, it would be very inefficient to generate this set explicitly. The simplest solution is to replace every literal greater than l by *true*. More formally:

Let $\leq$ be an ordering on literals. Let l be a literal. Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ be a †-formula. We denote by $\mathcal{C}_{|\leq l}$ (resp. $\mathcal{C}_{|<l}$) the †-formula $(\mathcal{L} \cup \{\beta \to \emptyset \mid \beta \in \mathcal{M}'\}, \mathcal{M} \setminus \mathcal{M}', \alpha, \delta, \mu')$ where $\mathcal{M}'$ is the set of nodes β in $\mathcal{M}$ s.t. $\mu(\beta) > l$ (resp. $\mu(\beta) \geq l$) and μ' is the restriction of μ to $\mathcal{M} \setminus \mathcal{M}'$.

Informally $\mathcal{C}_{|\leq l}$ is obtained from $\mathcal{C}$ by replacing any literal greater than l (according to the considered ordering) by *true*. This is equivalent to deleting, in the clause set denoted by the †-formula $\mathcal{C}$, all the clauses containing a literal strictly greater than l.

Note that we may have $\mathcal{M} = \mathcal{M}'$ and in this case $\mathcal{C}_{|\leq l}$ is equivalent to $\top$.

Proposition 3.8. Let $\mathcal{C}$ be a †-formula and let l be a literal. $\mathcal{S}(\mathcal{C}_{|\leq l}) \subseteq \mathcal{S}(\mathcal{C})$. Moreover, for any clause $C \in \mathcal{S}(\mathcal{C}_{|\leq l})$ and for any $l' \in C$, we have $l' \not> l$.

Proof:
Immediate (by a straightforward induction on $\prec_{\mathcal{C}}$). □

We adapt Definition 3.3 in order to integrate ordering restrictions:

Definition 3.4. (Ordered †-Resolution) Let $\mathcal{C}$ be a †-formula. Let λ be a node in $\mathcal{C}$ and let C, C' be two Λ-clauses in $\delta(\lambda)$.

Let $\mathcal{D}, \mathcal{D}'$ be two $\mathcal{CV}_\lambda(\mathcal{C})$-renamings of $\mathcal{C}_{|C}$ and $\mathcal{C}_{|C'}$ respectively, by new variables, not occurring in $\mathcal{C}$.

Let l, l' be two literals s.t. there exists β, β' occurring in $\mathcal{D}, \mathcal{D}'$ respectively s.t. $\mu'(\beta) = l$ and $\mu''(\beta') = l'$. Let σ be the most general $\mathcal{CV}_\lambda(\mathcal{C})$-unifier of l^c, l'.

Let $\mathcal{E} = \mathcal{D}_{|l\leq}$and let $\mathcal{E}' = \mathcal{D}'_{|l\leq}$.
The †-formula

$$\mathcal{C}\langle\mathcal{E}\langle w_{\beta'}(\mathcal{E}')[\bot]_{\beta'}\rangle_{\beta}\sigma\rangle_{\lambda}$$

is called an *internal $\leq$-resolvent* of $\mathcal{C}$ (w.r.t. the pair of nodes β, β' and the unifying substitution σ).

3.4.3. †-Factorization

The factorization rule can be seen as an extension of the sharing rule defined above, in which the instantiation of variables is allowed. More precisely:

Definition 3.5. (†-Factorization) Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ be a †-formula. Let λ be a node in $\mathcal{L}$. Let $C \in \delta(\lambda)$. Let β, β' be two nodes occurring in $\mathcal{C}_{|C}$. Let σ be a most general $\mathcal{CV}_{\lambda}(\mathcal{C})$-unifier of $\mu(\beta)$ and $\mu(\beta')$ and $\delta(\beta) = \delta(\beta') = \emptyset$.
Let $\mathcal{C}'$ be a sharing of $\mathcal{C}\sigma$ w.r.t. β, β'. Then $\mathcal{C}\langle\mathcal{C}'\rangle_{\lambda}$ is called a *factor* of $\mathcal{C}$.

Example 3.3. Let $\mathcal{C} : p(b,x) \vee (q(x,y) \wedge p(y,a))$. By unifying $p(b,x)$ and $p(y,a)$ we get the †-formula: $\mathcal{C} : \alpha{:}p(b,a) \vee (q(a,b) \wedge \alpha)$.
We have $\mathcal{S}(\mathcal{C}) = \{p(b,x) \vee q(x,y), p(b,x) \vee p(y,a)\}$ and $\mathcal{S}(\mathcal{C}) = \{p(b,a) \vee q(a,b), p(b,a)\}$.

Lemma 3.4. Let $\mathcal{C}$ be a †-formula and let $\mathcal{C}'$ be a †-formula obtained from $\mathcal{C}$ by †-factorization. Then $\mathcal{C}' \equiv \mathcal{C}$.

Proof:
The proof follows from Proposition 3.3. □

4. Soundness and Refutational Completeness

4.1. A New Redundancy Criteria

In section 3 we proved that the inference and simplification rules preserve the semantics of the considered †-formula. This entails soundness. Now, we prove that our method is refutationally complete. In fact completeness as such is a rather trivial issue since the †-resolution rule simulates the usual resolution rule: more precisely, any clause set $S = \{C_1, \ldots, C_n\}$ can be represented by a †-formula $\mathcal{C} = \alpha{:}(C_1 \wedge \ldots \wedge C_n)$, and any application of the resolution rule on two clauses C_i, C_j can be simulated by applying the †-resolution rule on the root node α, using the Λ-clauses corresponding to C_i and C_j. Similarly, the factorization rule may be simulated by the †-factorization rule.

However, this strategy is not very useful since it is equivalent to using ordinary resolution and all the advantages of our techniques (i.e. the extensive use of structure sharing) are lost. Therefore, we prove the refutational completeness of our calculus in a much stronger setting, using a more efficient strategy. Namely:

- The simplification rules can be applied as soon as possible on the considered †-formula. In particular, identical sub-†-formulae should be merged when possible.

- The resolution rule is systematically applied at the innermost position in the considered †-formula. Thus resolution rule is applied at root position only if it is necessary. The following example should clarify this point.

Consider the †-formula $\alpha{:}(p(x) \vee \beta{:}(q(y) \wedge \neg q(a))$. The resolution can be applied on the literals $q(y)$ and $\neg q(a)$ yielding the empty clause. By applying the †-resolution rule on the †-formula β, we get: $\alpha : (p(x) \vee \beta : (q(y) \wedge \neg q(a) \wedge \bot)$, i.e. (after simplification by internal subsumption): $\alpha : p(x)$.

However we could also in principle (according to Definition 3.3) apply the same rule on the root †-formula α instead of β. The reader can check that this yields the following †-formula:

$$\alpha : [(p(x) \vee \beta : (q(y) \wedge \neg q(a))) \wedge (p(x') \vee \beta' : ((p(x'') \vee \bot) \wedge \neg q(a))]$$

$$\equiv \alpha : [(p(x) \vee \beta : (q(y) \wedge \neg q(a))) \wedge (p(x') \vee \beta' : (p(x'') \wedge \neg q(a))]$$

Clearly the first †-formula subsumes the second one. The second application of the rule is not only much more complicated than the first one, but also useless.

This idea is formalized by the following:

Definition 4.1. (Usefulness) An application of the resolution rule on a †-formula $\mathcal{C}$, a node λ, two Λ-clauses C, C', two nodes β, β' and a substitution σ is said to be *useless* iff $C = C'$ and if C is of the form $D \vee \lambda'$, where $\lambda' \in \Lambda$, and $\mathcal{C}_{|D}$ contains no variable in *dom*(σ).

Similarly, an application of the factorization rule on a †-formula $\mathcal{C}$, a node λ, a Λ-clause C, and a substitution σ is said to be *useless* iff C is of the form $D \vee \lambda'$, where $\lambda' \in \Lambda$ and $\mathcal{C}_{|D}$ contains no variable in *dom*(σ).

4.2. Derivations and Limits

We need to introduce some definitions.

Definition 4.2. (Derivations) A *derivation* is a sequence $(\mathcal{C}_i)_{i \in I}$ of †-formulae (with either $I = \mathbb{N}$ or $I = [0..n]$ for some $n \in \mathbb{N}$) s.t. for any $i = I \setminus \{0\}$, $\mathcal{C}_i$ is obtained from $\mathcal{C}_{i-1}$ by applying the †-resolution, †-factorization, unit simplification, reduction, merging, sharing, elimination or subsumption rule.

Let $(\mathcal{C}_i)_{i \in I}$ be a derivation, with $\mathcal{C}_i = (\mathcal{L}_i, \mathcal{M}_i, \alpha_i, \delta_i, \mu_i)$. By definition, we must have $\alpha_i = \alpha_j$ for any $i, j \in I$ (no rule can change the root node of the †-formula). Moreover, the rules only add new nodes and modify the value of δ (by removing or adding Λ-clauses), thus we have $\mu_i(\beta) = \mu_j(\beta)$ if $\beta \in \mathcal{M}_i, \mathcal{M}_j$. Let $\mu_\infty = \bigcup_{i \in I} \mu_i$.

A node λ is said to be *persistent*, iff there exists $k \in I$ s.t. $\lambda \in \mathcal{L}_i$ for all $i \in I$ s.t. $i \geq k$. Let $\mathcal{L}_\infty$ be the set of persistent nodes.

Let λ be a persistent node. An $\mathcal{L}_\infty$-clause C is said to be *persistent* for λ, iff there exists $k \in I$ s.t. $C \in \delta_i(\lambda)$ for any $i \in I$ s.t. $i \geq k$. We denote by $\delta_\infty(\lambda)$ the set of persistent clauses for λ. Let $\mathcal{M}_\infty \stackrel{\text{def}}{=} \bigcup_{i \in I} \mathcal{M}_i$.

We denote by *lim*$(\mathcal{C}_i)_{i \in I}$ the †-formula: $(\mathcal{L}_\infty, \mathcal{M}_\infty, \delta_\infty, \mu_\infty)$.

4.3. Saturated †-Formulae

As usual, in order to ensure completeness, we need a notion of fairness:

Definition 4.3. (Fairness) A derivation $(\mathcal{C}_i)_{i \in I}$ is said to be *fair* if the following holds: if for some $i \in I$, the resolution (resp. factorization) rule is applicable on a given node λ in $\mathcal{C}_i$ on two Λ-clauses $C, C' \in \delta_i(\lambda)$ (resp. on a Λ-clause C), then:

- either λ is not persistent,
- or C, C' (resp. C) are not persistent for λ,
- or there exists $j \geq i$ s.t. $j \in I$ and s.t. the application of the resolution (resp. factorization) rule on λ, C, C' or λ, C', C (resp. λ, C) is redundant or useless for $\mathcal{C}_j$.

A †-formula $\mathcal{C}$ is said to be *saturated* iff there is no non-redundant and non-useless application of the resolution or factorization rules on $\mathcal{C}$. The next lemma states a key property of saturated †-formulae.

Lemma 4.1. Any saturated †-formula $\mathcal{C}$ s.t. $\mathcal{S}(\mathcal{C})$ does not contain $\Box$ is satisfiable.

Proof:
Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ be a saturated †-formula. We show that for any $\lambda \in \mathcal{L}$, $S = \mathcal{S}(\mathcal{C}_{|\lambda})$ is saturated (in the usual sense, i.e. that any clause deducible from S by resolution or factorization is subsumed by a clause in S). The proof is by induction on the ordering $\prec_{\mathcal{C}}$.

Let C_1, C_2 be two clauses in $\mathcal{S}(\mathcal{C})$. Assume that the resolution rule is applicable on C_1, C_2. W.l.o.g. we assume that $C_i = l_i \vee C'_i$ where there exists a m.g.u. of l_1, l_2^c s.t. $l_i\sigma$ is $\leq$-maximal in $C_i\sigma$.

By definition there exists two Λ-clauses $D_1, D_2 \in \delta(\alpha)$ s.t. $C_i \in \mathcal{S}(\mathcal{C}_{|D_i})$ (for $i = 1, 2$).

Let β_1, β_2 be the nodes corresponding to the literals l_1, l_2 respectively.

We have $\mu(\beta_i) = l_i$. Thus the resolution rule applies on the nodes β_1, β_2. Assume that this application of the resolution rule is not useless. Then by definition since $\mathcal{C}$ is saturated the application of the rule on $\mathcal{C}_1, \mathcal{C}_2$ or $\mathcal{C}_2, \mathcal{C}_1$ must be redundant. W.l.o.g. we assume that the application of the rule on $\mathcal{C}_1, \mathcal{C}_2$ is redundant (the proof for $\mathcal{C}_2, \mathcal{C}_1$ is similar). Hence $\mathcal{C}_{|\lambda}$ subsumes the †-formula

$$\mathcal{R} = \mathcal{E}_1 \langle w_{\beta_2}(\mathcal{E}_2)[\bot]_{\beta_2} \rangle_{\beta_1} \sigma$$

where $\mathcal{D}_i$ is a $\mathcal{CV}_\lambda(\mathcal{C})$-renamings of $\mathcal{C}_{|D_i}$ respectively, by new variables, not occurring in $\mathcal{C}$ and where $\mathcal{E}_i = \mathcal{D}_{i|l_i \leq}$.

Since l_i is maximal in C_i and $C_i \in \mathcal{S}(\mathcal{C}_{|D_i})$, $C_i\sigma$ occurs in $\mathcal{S}(\mathcal{E}_i)$.

This implies that $\mathcal{S}(w_{\beta'}(\mathcal{E}_2)[\bot]_{\beta'})$ contains C_2 and $\mathcal{S}(\mathcal{E}\langle w_{\beta'}(\mathcal{E}_2)[\bot]_{\beta'}\rangle_\beta)$ contains $C_1 \vee C_2$. Thus $\mathcal{S}(\mathcal{R})$ contains a clause $C_1\sigma \vee C_2\sigma$.

Now assume that the application of the resolution rule is useless.

Then we must have $D_1 = D_2$ and $D_1 = D \vee \gamma$, where $\mathcal{C}_{|D}$ contains no variable in *dom*(σ) (in particular l_1, l_2 cannot occur in $\mathcal{C}_{|D}$).

By definition, C_i is of the form $E'_i \vee E''_i \vee l_i$, where $E'_i \in \mathcal{S}(D_i\mathcal{C}_{|})$ and $E''_i \vee l_i \in \mathcal{S}(\gamma\mathcal{C}_{|})$. $\mathcal{C}_{|\gamma}$ is saturated, hence by the induction hypothesis, $\mathcal{S}(\mathcal{C}_{|\gamma})$ is saturated. Thus $\mathcal{S}(\mathcal{C}_{|\gamma})$ contains $E''_1\sigma \vee E''_2\sigma$.

Therefore $\mathcal{S}(\mathcal{C}_{|\lambda})$ contains the clause $E'_1 \vee E''_1\sigma \vee E''_2\sigma$, i.e. $E'_1\sigma \vee E''_1\sigma \vee E''_2\sigma$ (since $\mathcal{C}_{|D}$ contains no variable in *dom*(σ)). But this clause subsumes $C'_1\sigma \vee C'_2\sigma$.

The proof for the factorization rule is similar.

This implies that S is saturated, hence by completeness of the (usual) resolution and factorization rules, S is either satisfiable or contains $\Box$. $\Box$

The next lemma shows that the limit operator introduced above preserves the semantics of the considered †-formulae, in the sense that all the non-persistent clauses are redundant.

Lemma 4.2. Let $(\mathcal{C}_i)_{i \in I}$ be a fair derivation. $\mathcal{S}(\mathcal{C}_0) \equiv \mathcal{S}(\mathit{lim}(\mathcal{C}_i)_{i \in I})$.

Proof:
Let $\mathcal{D} = \mathit{lim}(\mathcal{C}_i)_{i \in I}$ We have shown that $\mathcal{S}(\mathcal{D})$ is a logical consequence of $\mathcal{S}(\mathcal{C}_0)$.

Now, let C be a clause in $\mathcal{S}(\mathcal{C}_0)$. For any $i \in I$, $\mathcal{C}_i$ is of the form $(\mathcal{L}_i, \mathcal{M}_i, \alpha, \delta_i, \mu_i)$. There exists a clause $C' \in \delta(\alpha)$ s.t. C is subsumed by a clause in $\mathcal{S}(\mathcal{C}_{0|C'})$.

It is well-known that $\preccurlyeq^{sub}$ is a well-founded ordering on clause sets (up to a renaming, see for instance [13]). But according to Lemma 3.1, $\mathcal{C} \preccurlyeq^{sub} \mathcal{D}$ implies $\mathcal{S}(\mathcal{C}) \preccurlyeq^{sub} \mathcal{S}(\mathcal{D})$, thus $\preccurlyeq^{sub}$ is a well-founded ordering on †-formulae (up to renaming). Let C' be a Λ-clause and let $i \in I$ s.t. C is subsumed by $\mathcal{C}_{i|C'}$ and $\mathcal{C}_{|C'}$ is minimal according to $\preccurlyeq^{sub}$.

Assume that C' is not persistent. Then C' must be deleted at some point. But this implies that there exists a clause $D \in \delta(\alpha)$ and $j \in I$ s.t. $\mathcal{C}_{j|D} \preccurlyeq^{sub} \mathcal{C}_{i|C'}$, which is impossible since $\mathcal{C}_{i|C'}$ would not be subsumption-minimal.

Thus C' is persistent. Hence $\mathcal{S}(\mathcal{D}_{|C'}) \subseteq \mathcal{S}(\mathcal{D})$. By definition, $\mathcal{D}_{|C'}$ is obtained from $\mathcal{C}_{i|C'}$ by reduction. Hence $\mathcal{D}_{|C'} \preccurlyeq^{sub} \mathcal{C}_{i|C'}$. Thus C is subsumed by a clause in $\mathcal{S}(\mathcal{D})$. $\Box$

Lemma 4.3. Let $(\mathcal{C}_i)_{i \in I}$ be a fair derivation. $\mathit{lim}(\mathcal{C}_i)_{i \in I}$ is saturated.

Proof:
This follows immediately from the definition. $\Box$

Theorem 4.1. (Soundness and Completeness) Let $(\mathcal{C}_i)_{i \in I}$ be a fair derivation. $\mathcal{C}_0$ is unsatisfiable iff there exists $i \in I$ s.t. $\mathcal{C}_i \equiv \bot$.

Proof:
Assume that there exists $i \in I$ s.t. $\mathcal{C}_i \equiv \bot$. Then $\mathcal{C}_i$ is unsatisfiable. By Lemma 3.3, 3.4, 3.1 and Proposition 3.1, 3.3, 3.2, 3.4, 3.7, we have $\mathcal{C}_0 \equiv \mathcal{C}_1 \equiv \ldots \equiv \mathcal{C}_i$. Thus $\mathcal{C}_0$ is unsatisfiable.

Assume that $\mathcal{C}_0$ is unsatisfiable. Since $(\mathcal{C}_i)_{i \in I}$ is fair, $\mathit{lim}(\mathcal{C}_i)_{i \in I}$ is saturated by Lemma 4.3. By Lemma 4.2, $\mathit{lim}(\mathcal{C}_i)_{i \in I}$ is unsatisfiable. But then according to Lemma 4.1 $\mathcal{S}(\mathit{lim}(\mathcal{C}_i)_{i \in I})$ contains $\Box$. But by definition any clause in $\mathcal{S}(\mathit{lim}(\mathcal{C}_{i i \in I}))$ occurs in $\mathcal{S}(\mathcal{C}_i)$ for some $i \in I$, thus there exists $i \in I$ s.t. $\Box \in \mathcal{S}(\mathcal{C}_i)$. By Proposition 3.6, $\mathcal{C}_i$ can be reduced to $\bot$ by internal subsumption. $\Box$

5. Polynomial Proof of the Pigeonhole Principle

In order to demonstrate the interest of the proposed calculus, we investigate its behavior on a rather well-known (propositional) problem: the *pigeonhole formula*, denoted by PigHPb_n, which states that there is no injective function from a set of $n+1$ pigeons into a set of n holes. It is well-known that PigHPb_n admits no polynomial proof in ordinary resolution [12]. We show that the †-resolution calculus introduced in Section 3 refutes *PigHPb*$_n$ in a number of steps that is polynomial w.r.t. the number of holes. This immediately implies that the standard resolution calculus cannot polynomially simulate †-resolution.

The variable $(i \in j)$ denotes the fact that pigeon i is in hole j. We define the following propositional clauses and clause sets:

- $P_n(i) \stackrel{\text{def}}{=} \bigvee_{j=1}^{n}(i \in j)$ (pigeon i must be in some hole).
- $P_n \stackrel{\text{def}}{=} \bigcup_{i=1}^{n+1}\{P_n(i)\}$ (each pigeon must be in some hole).
- $h_n(i, j_1, j_2) \stackrel{\text{def}}{=} \neg(j_1 \in i) \vee \neg(j_2 \in i)$ (hole i cannot contain pigeons j_1 and j_2).
- $h_n(i) \stackrel{\text{def}}{=} \{h_n(i, j_1, j_2) \mid j_1, j_2 \in [1..n+1], j_1 \neq j_2\}$ (hole i cannot contain two distinct pigeons).
- $H_n \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} h_n(i)$, (no hole contains two distinct pigeons).

The pigeonhole formula is the conjunction of P_n and H_n:

$$\mathit{PigHPb}_n \stackrel{\text{def}}{=} P_n \cup H_n.$$

PigHPb$_n$ is a set of clauses, hence can be seen as a set of †-formulae (since clauses are particular cases of †-formulae).

We shall construct a †-refutation of *PigHPb*$_n$ whose number of steps is polynomial w.r.t. n.

If C is a clause and S a clause set, $C \vee S$ denotes the clause set $\{C \vee D \mid D \in S\}$. Obviously, $\mathit{card}(C \vee S) = \mathit{card}(S)$.

The following formula $T(k, k', k'')$ expresses the fact that the holes $[k+1..n]$ contain *at least* k'' pigeons among the pigeons $[k'..n+1]$.

$$T(k, k', k'') \stackrel{\text{def}}{=} \bigvee_{P \subseteq [k'..n+1], \mathit{card}(P) = k''} \bigwedge_{i \in P} (i \in [k+1..n])$$

where the formula $(i \in [k+1..n])$ expresses the fact that pigeon i must be in some hole between $[k+1..n]$:

$$(i \in [k+1..n]) \stackrel{\text{def}}{=} \bigvee_{j=k+1}^{n} (i \in j).$$

By convention, $\bigvee_{i \in P} \phi_i$ (resp. $\bigwedge_{i \in \emptyset} \phi_i$) is 0 (resp. 1) if P is empty.

We have the following:

Lemma 5.1. For any $k \in [1..n], k' \in [1..n+1], k'' \in [1..n-k'+2]$:

$$T(k, k', k'') \equiv$$

$$((\neg(k' \in [k+1..n])) \Rightarrow T(k, k'+1, k''))$$
$$\wedge T(k, k'+1, k''-1))$$

Moreover:

- *T*$(k, k', k'') =$ *false* if $k'' > (n - k' + 2)$.
- *T*$(k, k', 0) =$ *true*.

Proof:
Assume that $\neg(k' \in [k+1..n])$ holds. Then pigeon k' is not in $[k+1..n]$. Thus *T*(k, k', k'') holds iff the holes $[k+1..n]$ contain at least k'' pigeons among $[k'+1..n+1]$ i.e. iff *T*$(k, k'+1, k'')$ holds. In this case we also have *T*$(k, k'+1, k''-1))$, since obviously *T*$(k, k'+1, k'')) \Rightarrow$ *T*$(k, k'+1, k''-1))$.

If $(k' \in [k+1..n])$ holds then *T*(k, k', k'') holds iff the hole in $[k+1..n]$ contain at least $k''-1$ pigeons among $[k'+1..n+1]$ hence iff *T*$(k, k'+1, k''-1))$ holds.

The second relation follows from the fact that $[k'..n+1]$ contains at most $n+1-k'+1 = n-k'+2$ pigeons.

The last relation follows immediately from the definition. □

Let $\mathcal{C}$ be a †-formula, let $k \in [1..n], k' \in [1..n+1], k'' \in [0..n+1]$. We write $\mathcal{C} \in \mathcal{A}(k, k', k'')$ iff:

- Either $k'' > (n - k' + 2)$ and $\mathcal{C} = \bot$;
- Or $k'' = 0$ and $\mathcal{C} = \top$;
- Or $0 < k'' \leq (n - k' + 2)$, and

$$\mathcal{C} = [(k' \in [k+1..n]) \vee \mathcal{C}'] \wedge \mathcal{C}''$$

  where $\mathcal{C}' \in \mathcal{A}(k, k'+1, k'')$ and $\mathcal{C}'' \in \mathcal{A}(k, k'+1, k''-1)$.

Intuitively, $\mathcal{C} \in \mathcal{A}(k, k', k'')$ iff $\mathcal{C}$ encodes the formula *T*(k, k', k'') (according to Lemma 5.1).

We assume given the following ordering on the propositional variables $(i \in j)$: $(i \in j) < (i' \in j')$ if $j > j'$ or if $j = j'$ and $i < i'$.

Let $\mathcal{C}, \mathcal{D}$ be two †-formulae. We write $\mathcal{C} \subseteq \mathcal{D}$ if $\mathcal{C} = \bigwedge_{i=1}^{n} \mathcal{C}_i, \mathcal{D} = \bigwedge_{i=1}^{m} \mathcal{D}_i$ and $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\} \subseteq \{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$.

Let $\mathcal{C}$ be a †-formula. We write $\mathcal{C} \in \mathcal{K}(k)$ iff there exists $\mathcal{C}' \subseteq \mathcal{C}$ s.t. $\mathcal{C}' \in \mathcal{A}(k, 1, n-k+1)$.

Outline of the proof

1. First we show that *PigHPb*$_n \in \mathcal{K}(0)$ (Lemma 5.2).
2. Then we show that if $\mathcal{C} \in \mathcal{K}(k)$ then one can generate from $\mathcal{C}$ (using a number of steps polynomial w.r.t. the size of $\mathcal{C}$) a †-formula $\mathcal{D}$ s.t. $\mathcal{D} \in \mathcal{K}(k+1)$ (Corollary 5.1).
3. Since the size of each reduced †-formula $\mathcal{C}$ s.t. $\mathcal{C} \in \mathcal{K}(k)$ is polynomial w.r.t. n (this follows from Lemma 5.4 below), we deduce from Point 1 and 2 (using a straightforward induction) that a †-formula $\mathcal{C}_n$ s.t. $\mathcal{C}_n \in \mathcal{K}(n)$ can be constructed from *PigHPb*$_n$ in a polynomial number of steps (indeed, computing the reduced form of a given †-formula $\mathcal{C}$ s.t. $\mathcal{C} \in \mathcal{K}(k)$ can be done in a number of steps that is polynomial w.r.t. the size of $\mathcal{C}$).
4. Finally, we show that $\mathcal{S}(\mathcal{C}_n)$ must contain □ (Lemma 5.6). By Proposition 3.6, this implies that the reduced form of $\mathcal{C}_n$ is $\bot$ which completes the proof.

We start by proving that *PigHPb*$_n \in \mathcal{K}(0)$.

Lemma 5.2. Let $n \in \mathbb{N}$. *PigHPb*$_n \in \mathcal{K}(0)$.

Proof:
By the above definition, it suffices to prove that $\mathcal{C} \subseteq$ *PigHPb*$_n$ for some $\mathcal{C} \in \mathcal{A}(0, 1, n+1)$. We show by induction on k that for any $k \in [0..n+1]$, there exists $\mathcal{C}_k$ s.t. $\mathcal{C}_k \subseteq$ *PigHPb*$_n$ and $\mathcal{C}_k \in \mathcal{A}(0, n+2-k, k)$ (we obtain the desired result for $k = n + 1$).

- If $k = 0$ then the proof is obvious since $\top \in \mathcal{A}(0, n+2, 0)$ and $\top \subseteq \mathcal{C}$. The above property holds for $\mathcal{C}_0 = \top$.

- If $k > 0$ then by induction hypothesis we have $\mathcal{C}_{k-1} \in \mathcal{A}(0, n+3-k, k-1)$ for some $\mathcal{C}_{k-1} \subseteq$ *PigHPb*$_n$. Moreover, since $k \in [1..n+1]$, we have $n+2-k \in [1..n+1]$ hence *PigHPb*$_n$ contains the clause $P_n(n+2-k) = (n+2-k \in [0+1..n])$. We define $\mathcal{C}_k \overset{\text{def}}{=} P_n(n+2-k) \wedge \mathcal{C}_{k-1}$. We have $n - (n+2-k+1) + 2 = k - 1 < k$ hence $\bot \in \mathcal{A}(0, n+2-k+1, k)$. Since $P_n(n+2-k) \vee \bot \ \equiv P_n(n+2-k)$, we have $\mathcal{C}_k \equiv (P_n(n+2-k) \vee \bot) \wedge \mathcal{C}_{k-1}$. Thus we deduce: $\mathcal{C}_k \in \mathcal{A}(0, n+2-k, k)$.

□

Now we show how to construct a †-formula $\mathcal{D} \in \mathcal{K}(k+1)$ from $\mathcal{C} \in \mathcal{K}(k+1)$.

Let $\mathcal{C}$ be a †-formula. Let l be a literal. We denote by *cut*$_l(\mathcal{C})$ the †-formula obtained from $\mathcal{C}_{|\leq l}$ by replacing each occurrence of $\neg l$ by $\bot$.

We need the following:

Lemma 5.3. Let $n \in \mathbb{N}$. Let $\mathcal{C}$ be a reduced †-formula s.t. $\mathcal{C} \in \mathcal{A}(k, k', k'')$, where $k \in [0..n]$, $k' \in [1..n+1]$, $k'' \in [0..n]$. Let $i < k'$. From *PigHPb*$_n \wedge \mathcal{C}$ we can derive in at most $n+1$ steps a †-formula $\mathcal{C}'$ s.t. *cut*$_{(i \in k+1)}(\mathcal{C}') \in \mathcal{A}(k+1, k', k'')$.

Proof:

By definition, any literal occurring in $\mathcal{C}$ occurs in a clause $(u \in [v+1..n])$, for some $u \geq k'$ and $v \geq k''$. By definition of $(u \in [v+1..n])$, this implies that any literal l occurring in S is of the form $(u \in v)$ for some $u \geq k'$ and $v \geq k+1$. Thus according to the chosen ordering we must have $l \leq (k' \in k+1)$.

Let $u \in [k'..n+1]$. Since $i < k'$, *PigHPb*$_n$ contains the clause $\neg(u \in k+1) \vee \neg(i \in k+1)$ where $(u \in k+1) > (i \in k+1)$. We apply the resolution rule successively on each literal $(n+1 \in k+1), (n \in k+1), \ldots, (k' \in k+1)$ using the above clauses. If $\mathcal{C}$ is reduced, it contains at most one occurrence of each literal $(u \in k+1)$, thus this requires at most $n+1$ applications of the resolution rule.

Obviously, this replaces any occurrence of a literal $(u \in k+1)$ by $\alpha{:}((u \in k+1) \wedge \neg(i \in k+1))$. Let $\mathcal{C}'$ be the obtained †-formula.

Let $\mathcal{C}(u, v)$ a sub-†-formula of $\mathcal{C}$ s.t. $\mathcal{C}(u, v) \in \mathcal{A}(k, u, v)$. Let $\mathcal{C}'(u, v)$ be the corresponding sub-†-formula of $\mathcal{C}'$. We show, by induction on the pair $(-u, v)$, that we have *cut*$_{(i \in k+1)}(\mathcal{C}'(u, v)) \in \mathcal{A}(k+1, u, v)$, for any $v > 0$, $u \geq k'$.

- If $v = 0$ then we have $\mathcal{C}(u,v) = \top$ thus $\mathit{cut}_{(i \in k+1)}(\mathcal{C}'(u,v)) = \mathcal{C}'(u,v) = \top$ hence $\mathit{cut}_{(i \in k+1)}(\mathcal{C}'(u,v)) \in \mathcal{A}(k+1,u,v)$.
- If $v > (n-u+2)$ then we have $\mathcal{C}(u,v) = \bot$ hence $\mathit{cut}_{(i \in k+1)}(\mathcal{C}'(u,v)) = \bot$ and the proof is immediate.
- Otherwise, since $0 < v \leq (n-u+2)$, we have

  $$\mathcal{C}(u,v) = ((u \in [k+1..n]) \vee \mathcal{C}_1) \wedge \mathcal{C}_2$$

  where $\mathcal{C}_1 \in \mathcal{A}(k,u+1,v)$ and $\mathcal{C}_2 \in \mathcal{A}(k,u+1,v-1)$

  We have $\mathcal{C}'(u,v) = (\mathcal{C}'' \vee \mathcal{C}_1') \wedge \mathcal{C}_2'$, where $\mathcal{C}'', \mathcal{C}_1', \mathcal{C}_2'$ are obtained from $(u \in [k+1..n])$, $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively by replacing any occurrence of a literal $(u \in k+1)$ by $\alpha{:}((u \in k+1) \wedge \neg(i \in k+1))$.

  By the induction hypothesis we have $\mathcal{C}_1' \in \mathcal{A}(k+1,u+1,v)$ and $\mathcal{C}_2' \in \mathcal{A}(k+1,u+1,v-1)$ where $\mathcal{C}_j'' \overset{\text{def}}{=} \mathit{cut}_{(i \in k+1)}(\mathcal{C}_j')$ $(j = 1,2)$.

  $(u \in [k+1..n]) = (u \in k+1) \vee (u \in [k+1+1..n])$. We have $u \geq k'$. The literals $(u \in k+1)$ are replaced by $\alpha{:}((u \in k+1) \wedge \neg(i \in k+1))$. Thus $\mathcal{C}''$ is of the form $(\alpha{:}((u \in k+1) \wedge \neg(i \in k+1)) \vee (u \in [k+1+1..n])$, hence $\mathit{cut}_{(i \in k+1)}(\mathcal{C}'') = (u \in [k+1+1..n])$.

  Therefore, we have $\mathit{cut}_{(i \in k+1)}(\mathcal{C}'(u,v)) \in \mathcal{A}(k+1,u,v)$.

□

Let p be a literal. We write $\mathcal{C} \lhd p$ if S is of the form $\bigwedge_{i=1}^{n}(p_i \vee \mathcal{C}_i)$, where $p_i > p$. Clearly, this implies that $\mathcal{C}_{|\leq p} = \top$.

By definition, if $\mathcal{C} \in \mathcal{A}(k,k',k'')$ then $\mathcal{C} \lhd (k' \in k+1)$ (this follows immediately from the definition, using a straightforward induction on k', k'': indeed any clause in $\mathcal{S}(\mathcal{C})$ contains a literal greater or equal than $(k' \in k+1)$).

The next lemma expresses the fact that two formulae $\mathcal{D}_i$ $(i = 1,2)$ s.t. $\mathcal{D}_i \in \mathcal{A}(k,k',k'')$ occurring a given †-formula can be "merged" by using the merging rule (thus there is at most one †-formula $\mathcal{D}$ s.t. $\mathcal{D} \in \mathcal{A}(k,k',k'')$.

Lemma 5.4. Let $\mathcal{C} = (\mathcal{L}, \mathcal{M}, \alpha, \delta, \mu)$ be a reduced †-formula s.t. $\mathcal{C} \in \mathcal{A}(k,k',k'')$ for $k,k',k'' \in \mathbb{N}$. Let $\mathcal{D}_1, \mathcal{D}_2$ be two †-formulae occurring in $\mathcal{C}$ s.t. $\mathcal{D}_i \in \mathcal{A}(k,l',l'')$ for any $i = 1,2$ (where $l,l' \in \mathbb{N}$). If $l'' \in [1..n-l'+2]$ then $\mathcal{D}_1 = \mathcal{D}_2$.

Proof:
The proof is by induction on $(-l', l'')$.

Since $l'' \in [1..n-l'+2]$, $\mathcal{D}_i$ is of the form $[((l' \in [k+1..n]) \vee \mathcal{D}_i') \wedge \mathcal{D}_i'']$ for $i = 1,2$ where $\mathcal{D}_i' \in \mathcal{A}(k,l'+1,l'')$ and $\mathcal{D}_i'' \in \mathcal{A}(k,l'+1,l''-1)$. By the induction hypothesis, we have $\mathcal{D}_1'' = \mathcal{D}_2''$. and $\mathcal{D}_1' = \mathcal{D}_2'$. By irreducibility w.r.t. the sharing rule, the two occurrences of $(l' \in [k+1..n])$ must be identical. Then the merging rule applies and $\mathcal{D}_2$ (for instance) can be replaced by $\mathcal{D}_1$ (thus if $\mathcal{D}_1 \neq \mathcal{D}_2$ then $\mathcal{C}$ is not reduced). □

The next lemma is the heart of the proof. It shows how to construct $\mathcal{C}' \in \mathcal{A}(k+1,k',k''-1)$ from $\mathcal{C} \in \mathcal{A}(k,k',k'')$.

Lemma 5.5. Let $n \in \mathbb{N}$. If $\mathcal{C} \in \mathcal{A}(k, k', k'')$, where $k \in [0..n]$, $k' \in [1..n+1]$, $k'' \in [0..n]$ and $k'' \leq (n - k' + 2)$, then one can generate from $\mathcal{C} \wedge \mathit{PigHPb}_n$ in a number of steps polynomial w.r.t. n a †-formula $\mathcal{C}'$ s.t. $\mathcal{C}' \in \mathcal{A}(k+1, k', k''-1)$.

Proof:

We write $\mathcal{C} \in \mathcal{A}'(k, k', k'')$ iff $\mathcal{C}$ is of the form $\mathcal{C}' \wedge \mathcal{C}$" where $\mathcal{C}' \in \mathcal{A}(k, k', k'')$ and if $k \neq 0$ then $\mathcal{C}'' \lhd (n+1 \in k)$.

First, we remark that we have $\mathcal{C} \lhd (n+1 \in k+1)$. We prove that one can generate from $\mathcal{C}$ a †-formula $\mathcal{C}'$ s.t. $\mathcal{C}' \in \mathcal{A}'(k+1, k', k''-1)$.

The proof is by induction on $(-k', k'')$.

- If $k'' = 0$ then the proof is immediate (since $\mathcal{C}' = \top$, satisfies the desired conditions).
- If $k'' > 0$, then since $k'' \leq (n - k' + 2)$, $\mathcal{C}$ is of the form

  $$((k' \in [k+1..n]) \vee \mathcal{C}_1) \wedge \mathcal{C}_2$$

  where $\mathcal{C}_1 \in \mathcal{A}(k, k'+1, k'')$ and $\mathcal{C}_2 \in \mathcal{A}(k, k'+1, k''-1)$.

  By the induction hypothesis, since $(k''-1) \leq (n - (k'+1) + 2)$, we deduce from $\mathcal{C}_2$ a †-formula: $\mathcal{C}_2'$ s.t. $\mathcal{C}_2' \in \mathcal{A}'(k+1, k'+1, k''-2)$.

  If $k'' = (n - k' + 2)$ then we have $\mathcal{C}_1 = \bot$ (since $k'' > (n - (k'+1) + 2)$). Otherwise, by the induction hypothesis, we deduce from $\mathcal{C}_1$ a clause set: $\mathcal{C}_1'$ s.t. $\mathcal{C}_1' \in \mathcal{A}'(k+1, k'+1, k''-1)$. Thus we can deduce from $(k' \in [k+1..n]) \vee \mathcal{C}_1$ the clause set $((k' \in [k+1..n]) \vee \mathcal{C}_1') \wedge \mathcal{C}_2'$ where either $\mathcal{C}_1' = \bot$ (if $k'' = (n - k' + 2)$) or $\mathcal{C}_1' \in \mathcal{A}'(k+1, k'+1, k''-1)$.

  But $(k' \in [k+1..n]) = (k' \in k+1) \vee (k' \in [k+1+1..n])$.

  By Lemma 5.3, since $k' < k'+1$ we can construct from $\mathcal{C}_2$ a clause $\mathcal{C}_2''$ s.t. $\mathit{cut}_{(k' \in k+1)}(\mathcal{C}_2'') \in \mathcal{A}(k+1, k'+1, k''-1)$. Moreover, we have $\mathcal{C}_2' \lhd (k' \in k+1) < (n+1 \in k+1)$.

  By resolving this †-formula with $(k' \in k+1) \vee (k' \in [k+1+1..n]) \vee \mathcal{C}_1'$ we get:

  $$\mathit{cut}_{(k' \in k+1)}(\mathcal{C}_2'') \vee (k' \in [k+1+1..n]) \vee \mathcal{C}'_{1|\leq(k' \in k+1)}.$$

  Let $\mathcal{C}_1'' = \mathcal{C}'_{1|\leq(k' \in k+1)}$. Since $\mathcal{C}_1' \in \mathcal{A}'(k+1, k'+1, k''-1)$, $\mathcal{C}_1'$ must be of the form $\mathcal{T} \wedge \mathcal{T}'$ where $\mathcal{T} \in \mathcal{A}(k+1, k'+1, k''-1)$ and $\mathcal{T}' \lhd (n+1 \in k+1)$. Since $(k' \in k+1) < (n+1 \in k+1)$, we have $\mathcal{T}'_{|\leq(k' \in k+1)} = \top$. Moreover, $\mathcal{T}_{|\leq(k' \in k+1)} = \mathcal{T}$ (indeed, since $\mathcal{T} \in \mathcal{A}(k+1, k'+1, k''-1)$, $\mathcal{T}$ contains only atoms of the form $(\ldots \in v)$ for some $v \geq k+2$, thus all the atoms in $\mathcal{T}$ are lower than $(k' \in k+1)$).

  By Lemma 5.4 we have $\mathit{cut}_{(k' \in k+1)}(\mathcal{C}_2'') \equiv \mathcal{T}$.

  Hence after (at most) one merging step we get: $(k' \in [k+1+1..n]) \vee \mathcal{T}$.

  Hence the clause set $\mathcal{C}' = ((k' \in [k+1+1..n]) \vee \mathcal{T}) \vee \mathcal{C}_2'$ has been generated. By definition we have $\mathcal{C}' \in \mathcal{A}(k+1, k', k'')$.

  Clearly, a polynomial number of steps is required for each sub-†-formula $\mathcal{D} \in \mathcal{A}(k, k', k'')$. If the †-formulae are reduced, the number of distinct such †-formulae is bounded by n^2 (due to Lemma 5.4). Thus, the total number of steps is polynomial.

□

Corollary 5.1. Let $n \in \mathbb{N}$. If $\mathcal{C} \in \mathcal{K}(k)$ where $k \leq n$. We can construct from $S \cup \mathit{PigHPb}_n$ in a number of steps polynomial w.r.t. n a †-formula $\mathcal{D}$ s.t. $\mathcal{D} \in \mathcal{K}(k+1)$.

Proof:
This follows immediately from Lemma 5.5. □

Lemma 5.6. Let $n \in \mathbb{N}$. Let $\mathcal{C}$ be a †-formula s.t. $\mathcal{C} \in \mathcal{K}(n)$. $\mathcal{S}(\mathcal{C})$ contains $\Box$.

Proof:
By definition, we have $\mathcal{C}' \subseteq \mathcal{C}$ for some $\mathcal{C}' \in \mathcal{A}(n,1,1)$. We show that for any †-formula $\mathcal{C}'$ s.t. $\mathcal{C}' \in \mathcal{A}(n, n+2-i, 1)$, $\mathcal{S}(\mathcal{C}')$ contains $\Box$. The proof is by induction on i.

- If $i = 0$ then $1 > n-(n+2-i)+2$ hence the proof stems from the definition of $\mathcal{C}' \in \mathcal{A}(k,k',k'')$.
- Otherwise, by definition of $\mathcal{C}' \in \mathcal{A}(k,k',k'')$, $\mathcal{S}(\mathcal{C}')$ must contain the clauses in $\mathcal{S}((n+2-i \in [n+1..n]) \vee \mathcal{S}(\mathcal{D}))$ where $\mathcal{D} \in \mathcal{A}(n, n+3-i, 1)$. By definition, we have $(n+2-i \in [n+1..n]) = \Box$ for any $i \in [1..n+1]$. Moreover by induction hypothesis $\mathcal{S}(\mathcal{D})$ contains $\Box$. Thus $\mathcal{S}(\mathcal{C}')$ contains $\Box$.

□

Theorem 5.1. Let $n \in \mathbb{N}$. The †-formula $\bot$ can be obtained from PigHPb_n by †-resolution and †-factorization, in a number of steps that is polynomial w.r.t. n.

Proof:
Immediate by Lemma 5.2, Lemma 5.6 and Corollary 5.1. □

6. Related works

Not surprisingly, several authors already tried to improve the efficiency of the resolution calculus by introducing mechanisms for sharing information and avoiding redundant computations.

Our technique is obviously related to the use of *structure-sharing*, that is used by most existing provers in order to represent information in a convenient and efficient way (see for instance [5]). Sharing is ubiquitous in implementations of automated reasoning systems. However, rather than using it as a tool for storing terms and clauses, we use it at the *logical* level, and take it into account when defining the inference rules. Shared subclauses and goals can be merged, which yields shorter derivations as well as more compact representations of the search space.

The approach presented in [14] (called *paramodulation without duplication*) is similar to our work, though more concerned by the reduction of the search space than by the reduction of proof length. Clauses are represented by graphs and inferences are performed by adding new edges into the graph. The idea is similar to the one in this paper, but there are some differences. The method by [14] handles equality and has the advantage that terms may be shared as well as literals and clauses. To this aim, instead of explicitly applying the substitutions (i.e. the unifiers and renamings) generated during proof search, the method encodes them into the graph, as equational conditions attached to the edges. Moreover, graphs may be cyclic (in our context this would mean that inferences between a †-formula and the context in which it occurs would be allowed). A drawback is that this makes the detection of empty clauses much more difficult: it does not terminate in general thus must be interleaved with the inference steps. Moreover, additional information must be stored into the edges. Our method avoids this, though at the cost of some additional redundancy. In [14], building the graph is only part of the proof search: the other part is implicitely delegated to the algorithm for detecting empty clauses (in particular, for Horn clauses, the entire proof search is performed by this algorithm, since the building of the graph is trivial), whereas in our approach all the work is explicitly done by the inference rules. An advantage of our method is the use of the sharing and merging rules, that allow one to merge identical subgoals without having to care how they have been obtained. Notice that this feature (not shared by [14]) was essential for constructing a polynomial refutation of PigHPb_n. Moreover this also simplifies the writing of the inference rules (we make the †-formulae disjoint, and merge them afterward when possible). Moreover, our technique also has the advantage that inferences may be performed without having to explicitly compute the clause on which the rule is applied (only the literal on which the rule is applied is important). This is required in [14] because one need to know the vertices to which the “replacement edges” should be added.

It is not difficult to see that Tseitin’s extended resolution [20] polynomially simulates our technique, when restricted to the propositional case (as recalled before, first-order versions of Tseitin’s extension rule can also be considered). Indeed, extended resolution allows one to introduce arbitrary definitions in the clause set, using equivalences of the form $p \Leftrightarrow \phi$, where p is a variable and ϕ a formula. These definitions can be proceeded as usual, after transformation into clausal form. This feature can be used to encode †-formulae into standard clauses, simply by introducing additional propositional variables to “name” shared subformulae. Then our †-resolution rule can be simulated by repeated applications of the resolution rule. Rather than giving a formal justification and proof, we provide an illustrating example that should allow the reader to grasp the intuitive idea.

Example 6.1. The †-formula $(p \wedge q) \vee r$ can ge represented by the following clause set:

$$p \vee \neg r'$$
$$q \vee \neg r'$$
$$r \vee r'$$

where r' is introduced as a shortcoming for $q \wedge r$. Resolving the initial †-formula with $\neg p \vee p'$ yields:

$$(p \wedge q \wedge p') \vee r.$$

A clause set representing this †-formula can be obtained by resolving the clauses $p \vee r'$ and $\neg p \vee p'$.

But Tseitin’s extended resolution is not usable in practice due to the huge branching factor. Our method can be seen as a restriction of extended resolution which has the following advantages: first it strongly restricts the kind of definitions that can be proceeded. Only the ones allowing an immediate reduction of the size of the formula will be considered. Second the adding of the definitions is *dynamic* and does not need not to be performed explicitly.

Our method is also related to the multiresolution approach described in [6] in the context of propositional logic. The idea of multiresolution is to use binary decision diagrams (see for instance [15]) to represent sets of clauses. This yields a representation of propositional clause sets that is much more compact that the usual one. Moreover [6] shows how to adapt the usual resolution principle in order to operate on these representations. Due to the more compact representation and due to the sharing of information, one resolution step in the obtained calculus can correspond to *several* resolution steps in the usual sense. Building on these results, [6] proposed a breath-first search strategy similar to the Davis and Putnam procedure [8]. As our procedure, the multiresolution calculus refutes the pigeonhole problem in polynomial time. It is worth mentioning that [16] proposed a related approach, also based on breath-first search and also taking advantage of the expressive power of BDD's to reduce the search space, but using a very different and original way of representing the search space. Roughly speaking, the idea is to denote as a BDD the set of sets of "active" clauses, i.e. of clauses that remains to satisfy in the partial models constructed so far.

Though based on similar principles, our technique is essentially different from the one of [6] (and [16]) due to the very different way of representing information (†-formulae w.r.t. BDD). Moreover the procedures of [6, 16] are (as far as we are aware) restricted to propositional logic, whereas our method handles first-order logic as well.

In [17], a calculus has been proposed for reducing the length of resolution proofs, by factoring some literals. The idea is completely different from the one considered in the present paper, since in [17] only literals belonging to the same clause can be shared, whereas the principle of the †-resolution calculus is to share literals or disjunction of literals between distinct clauses. On the other hand, the calculus presented in [17] has the advantage that the common part of *distinct* literals can be shared. For instance the clause $p(a) \vee p(b)$ is expressed as a constrained clause: $p(x)$ if $x = a$ or $x = b$. The pattern $p(x)$ is shared between the two terms. Thus the two presented approaches are orthogonal. Their combination (and/or merging) could deserve to be considered.

7. Conclusion

We have presented a method for reducing proof length in resolution-based calculi, which tries to avoid, when possible, the duplication of information. This is done by an extensive use of sharing at the logical level. We designed inference and simplification rules, and we proved their soundness and refutational completeness. Using the well-known pigeonhole problem as an example, we have shown that our technique – when restricted to propositional calculus – can reduce the length of the proof by an exponential factor.

Future works include the implementation of a theorem-prover based on the proposed rules in order to estimate their practical performances. Moreover, the extension of our technique to equational theorem proving (using †-extensions of the paramodulation or superposition rules) deserves to be considered.

References

[1] Baader, F., Snyder, W.: Unification Theory, in: *Handbook of Automated Reasoning* (A. Robinson, A. Voronkov, Eds.), vol. I, chapter 8, Elsevier Science, 2001, 445–532.

[2] Baaz, M., Leitsch, A.: Complexity of resolution proofs and function introduction, *Annals of Pure and Applied Logic*, **20**, 1992, 181–215.

[3] Barendregt, H., van Eekelen, M., Glauert, J., Kenneway, R., Plasmeijer, M. J., Sleep, M.: Term Graph Rewriting, *PARLE'87*, Springer, LNCS 259, 1987, 141–158.

[4] Ben-Sasson, E., Galesi, N.: Space Complexity of Random Formulae in Resolution, *Colloquium on Computational Complexity*, 2001.

[5] Boyer, R. S., Moore, J. S.: The Sharing of Structure in Theorem Proving Programs, in: *Machine Intelligence 7* (B. Meltzer, D. Michie, Eds.), Edinburgh University Press, 1972, 101–116.

[6] Chatalic, P., Simon, L.: Multiresolution for sat checking, *Journal of Artificial Intelligence Tools*, **10**(4), 2001.

[7] Cook, S., Reckhow, R.: The Relative Efficience of Propositional Proof Systems, *The Journal of Symbolic Logic*, **44**(1), 1979, 36–50.

[8] Davis, M., Putnam, H.: A Computing Procedure for Quantification Theory, *Journal of the ACM*, **7**(3), July 1960, 201–215.

[9] Eder, E.: *Relative Complexities of First-Order Logic Calculi*, Vieweg, 1990.

[10] Egly, U.: On different concepts of function introduction, *Computational Logic and Proof Theory, KGC 93*, Springer, LNCS 713, 1993, 172–183.

[11] Fitting, M.: *First-Order Logic and Automated Theorem Proving*, Texts and Monographs in Computer Science, Springer-Verlag, 1990.

[12] Haken, A.: The Intractability of Resolution, *Theoretical Computer Science*, **39**, 1985, 297–308.

[13] Leitsch, A.: *The resolution calculus*, Springer. Texts in Theoretical Computer Science, 1997.

[14] Lynch, C.: Paramodulation without Duplication, *Proceedings of the Tenth Annual IEEE Symp. on Logic in Computer Science, LICS* (D. Kozen, Ed.), IEEE Computer Society Press, June 1995, 167–177.

[15] Meinel, C., Theobald, T.: *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Application*, Springer, 1998.

[16] Motter, D. B., Markov, I. L.: A Compressed Breadth-First Search for Satisfiability, *ALENEX*, 2002, 29–42.

[17] Peltier, N.: A Resolution Calculus for Shortening Proofs, *Logic Journal of the Interest Group in Pure and Applied Logics*, **13**, 2005, 307–333.

[18] Robinson, J. A.: A machine-oriented logic based on the resolution principle, *J. Assoc. Comput. Mach.*, **12**, 1965, 23–41.

[19] Statman, R.: Lower Bounds on Herbrand's Theorem, *Proc. AMS 75*, 1979, 104–107.

[20] Tseitin, G. S.: On the Complexity of Derivation in Propositional Calculus, in: *Studies in Constructive Mathematics and Mathematical Logics* (A. Slisenko, Ed.), 1968.