University of Ostrava

Institute for Research and Applications of Fuzzy Modeling

Non-clausal resolution theorem prover

background, implementation techniques and applications

Hashim Habiballa

Research report No. 64

2005

Submitted/to appear:

Empirically Successful Automated Reasoning Workshop (ESCAR'2005) (CADE), Tallinn, Estonia

Supported by:

Research grant of Czech Ministry of Education - MSM 6198898701

University of Ostrava
Institute for Research and Applications of Fuzzy Modeling
30. dubna 22, 701 03 Ostrava 1, Czech Republic

tel.: +420-69-6160234 fax: +420-69-6120 478
e-mail: Hashim.Habiballa@osu.cz

Non-clausal resolution theorem prover
background, implementation techniques and applications

Hashim Habiballa *
Institute for Research and Applications of Fuzzy Modeling
and Department of Computer Science
University of Ostrava
30. Dubna 22
Ostrava, Czech Republic
Hashim.Habiballa@osu.cz
http://www.volny.cz/habiballa/index.htm

Abstract

The article deals with an automated theorem prover designed for an inference based software tool that performs non-clausal resolution. The prover is implemented in the form of an experimental application - GERDS (GEneralized Resolution Deductive System), which is able to handle first-order logic formulas i.e. it generates proofs of predicate logic theorems from sets of axioms. The reduction to clausal form and skolemization is avoided and therefore in special cases it brings a new view to some problem sets, which are currently difficult. Especially it avoids destruction of formula structure that may contain useful contextual information. Although the notion is well-known for a long time, its possibilities have been overlooked. Basic definitions and knowledge obtained within the creation of the application are discussed. Comparison through examples from another existing CNF based automated prover SPASS is given. The essential knowledge of classical logic and principles of deduction and resolution is supposed. The main intent of the article is not theoretical, but should bring view to data structures and algorithms used. From theoretical point of view it gives look into prospective application of non-clausal resolution in fuzzy logic and description logic (DL) with their integration into universal theorem prover for fuzzy DL.

keywords: Automated theorem proving, non-clausal resolution, general resolution, fuzzy logic, description logic.

1 Introduction

The first-order logic (FOL) is well known notion among mathematicians and computer scientists. It is a suitable formal representation for expressing knowledge and performing reasoning. Although it is very popular and clear way, the problem of automated theorem proving is not so simple. Utilization of FOL in knowledge representation and deduction often requires a lot of transformations, before one can use any deductive system. These transformations destroy the meaning of the formula and may produce high amount of new transformed formulas (e.g. clausal resolution principle). Such a destruction cause loss of contextual information, which could be

*This work was supported by research grant of Czech Ministry of Education - MSM 6198898701

needful for successful inference process. It has also decisive advantages such as the efficiency of the proof, well-developed strategies and high-speed techniques.

FOL covers many deductive systems and methods. Probably the most implemented and practiced is principle based on resolution [St96]. This originally simple rule has been intensively studied and has been extended to numerous theories leading to efficient implementations [Ba01]. The conventional view of resolution is closely related with clausal normal form of a formula and with unification in predicate logic. It is not very difficult to obtain equivalent clausal normal form of any formula (particularly for non-quantified formulas). Such a formula disintegrates into several clauses (it may be a high amount of formulas, see the example 7), that are not holding original rich information alone, but must be treated as a subset of closely-related formulas. The main problem of this construction lies in the fact that classical tools e.g. Prolog doesn't reflect this situation.

When looking for some suitable solution, we may use relatively old and well-known, but practically unused notion of general resolution rule [Ba97], which is capable of performing resolution on general formulas of FOL. There are thoroughly discussed various resolution techniques and strategies in the cited paper, but it is not the focus of this paper to utilize these techniques. We will try to present general resolution as a possible technique for non-clausal resolution on universal formulas of FOL [Mu82] (without their deformations) and also we will present simple (but efficient) strategy for reduction of search space during refutational resolution process (empirically successful). The main contribution should be the implementation (experimental computer application) - presenting data structures and algorithms in detail. In order to not confuse the reader it must be stated that the application is not fully comparable with existing professionally developed tools like SPASS, Gandalf etc. It serves only for demonstration of the methods and could be directly used for competition with other provers. Nevertheless the significance of the non-clausal resolution through GERDS is demonstrated on modern approaches to knowledge representation - Description Logic (DL) Theorem Proving and Fuzzy Logic Theorem Proving. Even these two approaches could be naturally merged into an automated theorem prover for Fuzzy DL.

Although we have a lot of modern approaches to automated theorem proving (ATP) like orderings [Ba01], chain resolution [Ta03] etc. at present, non-clausal resolution may give good results for special problems (equivalence, contextual knowledge on semantic web).

It has been mentioned a computer application, which is called GEneralized Resolution Deductive System (GERDS), which performs searching for a proof of a query. This application was implemented on Windows 32 platform and its usage is simple and transparent, since it allows natural notation of FOL. It is rather a demonstrational tool than a theorem proving engine, nevertheless it fulfils its role - presentation of possibilities of FOL, DL and Fuzzy FOL and DL. It will be shown in examples to understand how to use it. The application can be obtained from the http location [Ha99].

2 Background

2.1 General resolution - brief overview

There is a comprehensible chapter in Handbook [Ba01] concerning basics of generalized theory of resolution written by Harald Ganzinger and Leo Bachmair (it is also reachable as a research report [Ba97]). They present elegant deduction rules for general formulas and also resolution methods. Let's review these rules.

Definition 1 *General resolution*

$$\frac{F[G] \quad F'[G]}{F[G/\bot] \vee F'[G/\top]}$$

where F and F' are formulas - premises of first-order logic and G represents an occurrence of a subformula (we may consider atoms only) of F and F'. The expression below the line means the resolvent of premises on G. Every occurrence of G is replaced by false in the first formula and by true in the second one. It is also called F the positive, F' the negative premise, and G the resolved subformula.

The proof of the soundness of the rule is similar to clausal resolution rule proof. Suppose the Interpretation I in which both premises are valid. In I, G is either true or false. If G ($\neg G$) is true in I, so is $F[G/\top]$ ($F[G/\bot]$). From this point of view, it shows, that the resolution rule is nothing more that assertion of the type: If we have two formulas holding simultaneously and they contain the same formula, then we can deduce that either the common subformula is true in this interpretation then the truthfulness is assured by the first formula or the second formula in the opposite case. Now we can have a look to the question, how these facts influence the view of clausal resolution. Consider following table showing various cases of resolution on the similar clauses.

Premise1	Premise2	Resolvent	Simplified	Comments
$a \vee b$	$b \vee c$	$(a \vee \bot) \vee (\top \vee c)$	$\top$	no sense
$a \vee \neg b$	$b \vee c$	$(a \vee \top) \vee (\top \vee c)$	$\top$	redundant
$a \vee b$	$\neg b \vee c$	$(a \vee \bot) \vee (\bot \vee c)$	$a \vee c$	right resolution
$a \vee \neg b$	$\neg b \vee c$	$(a \vee \top) \vee (\bot \vee c)$	$\top$	no sense

Table 1: Clausal resolution in the frame of general resolution

When trying to make a reasonable resolvent it should be considered, which formula has to be taken as positive premise. In the clausal case, it is trivial question; it is the positive atom (unsigned). Let's have a look into an example of a non-clausal refutation.

Example 1 *General resolution with equivalence*

1. $a \wedge c \leftrightarrow b \wedge d$ (axiom)
2. $a \wedge c$ (axiom)
3. $\neg[b \wedge d]$ (axiom) - negated goal
4. $[a \wedge \bot] \vee [a \wedge \top]$ (resolvent from (2), (2) on c) $\Rightarrow$ a
5. $[a \wedge \bot] \vee [a \wedge \top \leftrightarrow b \wedge d]$ ((2), (1) on c) $\Rightarrow a \leftrightarrow b \wedge d$
6. $\bot \vee [\top \leftrightarrow b \wedge d]$ ((4), (5) on a) $\Rightarrow b \wedge d$
7. $\bot \wedge d \vee \top \wedge d$ ((6), (6) on b) $\Rightarrow$ d
8. $b \wedge \bot \vee b \wedge \top$ ((6), (6) on d) $\Rightarrow b$
9. $\bot \vee \neg[\top \wedge d]$ ((8), (3) on b) $\Rightarrow \neg d$
10. $\bot \vee \neg\top$ ((7), (9) on d) $\Rightarrow \bot$ (refutation)

In the above example, it can be observed how simply it is to handle general formulas. Some of the used manipulations were not discussed (how to select formula order to not produce redundant resolvents). Simplification used above is also not an essential need, but it was performed only

for clarity. It is possible to sustain the resolvents unsimplified until it is completely empty of atoms and then to determine logical value of the resolvent (it may simplify the work with tree-based structures as described in following text). There is an important case of resolution called self-resolution describing resolution on one formula.

Definition 2 *General self-resolution*

$$\frac{F[G]}{F[G/\bot] \vee F[G/\top]}$$

This type of rule allows us to perform unusual, but in some cases efficient, way of refutation the set of formulas as a whole formula. Again, consider the set from previous example.

Example 2 *Refutation by self-resolution*

$a \wedge c \leftrightarrow b \wedge d$ (axiom)
$a \wedge c$ (axiom)
$\neg[b \wedge d]$ (axiom) - negated goal

Now we translate it to one formula:

1. $[a \wedge c \leftrightarrow b \wedge d] \wedge [a \wedge c] \wedge \neg[b \wedge d]$
2. $[\bot \wedge c \leftrightarrow b \wedge d] \wedge [\bot \wedge c] \wedge \neg[b \wedge d] \vee [\top \wedge c \leftrightarrow b \wedge d] \wedge [\top \wedge c] \wedge \neg[b \wedge d]$
(resolving on a) $\Rightarrow [c \leftrightarrow b \wedge d] \wedge c \wedge \neg[b \wedge d]$
3. $[\bot \leftrightarrow b \wedge d] \wedge \bot \wedge \neg[b \wedge d] \vee [\top \leftrightarrow b \wedge d] \wedge \top \wedge \neg[b \wedge d]$ (resolving on c) $\Rightarrow [b \wedge d] \wedge \neg[b \wedge d]$
4. $[\bot \wedge d] \wedge \neg[\bot \wedge d] \vee [\top \wedge d] \wedge \neg[\top \wedge d]$ (resolving on b) $\Rightarrow d \wedge \neg d$
5. $\bot \wedge \neg\bot \vee \top \wedge \neg\top$(resolving on d) $\Rightarrow \bot$ (refutation)

This type of resolution has two advantages as you saw in the example. It leads to the refutation quickly and without the need of deciding, if the resolvent will be redundant or not. Unfortunately, the self-resolution is not suitable for non-propositional instances.

So it was a short overview of the basics of the notions from the cited handbook. Although these rules seem trivial, they can be simply introduced into a computer application, which uses them with combination of high-level data structures and algorithms. It could lead to efficient reasoning in potential application in practice.

2.2 Selection of premises and simplification

As it was noticed above, it is important to decide which of the two premises to be taken as positive. It has been developed a simple way to decide it. It is an extended notion of polarity, which is rather intuitive item. It is clear, that every subformula has some polarity and if some of its superior connectives are equivalence, then it is both positive and negative. This anomalous property causes problems in conjunction with an existential variable within unification of variables and its solution is described below, within the lifting of inferences section. Following theorem gives an algorithm for the determination of polarity.

Theorem 1 *Extended polarity criteria*

1. *F is a positive subformula of F.*

2. *If $\neg G$ is a positive (resp. negative) subformula of F, then G is a negative (resp. positive) subformula of F.*

3. *If $G \vee H$ is a positive (resp. negative) subformula of F, then G and H are both positive (resp. negative) subformulas of F.*

4. *If $G \wedge H$ is a positive (resp. negative) subformula of F, then G and H are both positive (resp. negative) subformulas of F.*

5. *If $G \rightarrow H$ is a positive (resp. negative) subformula of F, then G is a negative (resp. positive) subformula and H is a positive (resp. negative) subformula of F.*

6. *If $G \leftrightarrow H$ is a subformula of F, then every subformula of G and H is positive and negative subformula of F.*

Then it is possible to set the formula with positive polarity as the positive premise. This solves the problem of wrong order of premises i.e. it avoids redundant resolvents.

Example 3 *Positive and negative premise*

Source formulas (axioms) :
F0 : $a \vee b$, F1 : $b \vee c$, F2 (query) : $\neg[a \vee c]$
R0 [F2&F2] : $\neg c$. R1 [F2&F1] : $\neg b$.
R2 [F2&F0] : b. R3 [F1&F2] : $\neg b$.
R4 [F1&F0] : $c \vee a$. (c as negative premise)
R5 [F0&F2] : b.
R6 [F0&F1] : $a \vee c$. (a as positive premise)
R7 [R5&R3] : YES.

R4 was created as resolvent of F1 and F0 where F1 was treated as a negative premise: $(\neg\top \vee c) \vee (a \vee \bot)$.

2.3 Resolution strategies and redundant resolvents

In this paragraph we will discuss the method which is the main point of the work on any automated prover. There is a lot of strategies which makes proofs more efficient when we use refutational proving. We consider well-known strategies - orderings, filtration strategy, set of support etc. One of the most effective strategies is the elimination of consequent formulas. It means the check if a resolvent is not a logical consequence of a formula in set of axioms or a previous resolvent. If such a condition holds it is reasonable to not include the resolvent into the set of resolvents, because if the refutation can be deduced from it, then so it can be deduced from the original resolvent, which it implies of. The clausal case is trivial. Consider these two clauses $p(X, a)$ and $p(b, a) \vee r(Y)$. It is clear that the second clause implies from the first (if we suppose that set of constant objects is closed - only b as the first argument in p predicate is considered).

It has been devised an algorithm for the non-clausal case based on self-resolution rule. This algorithm highly reduces the proof search time, as it should be seen from examples. Before it was introduced into the inference core of GERDS, proofs were useless, due to their time and space complexity. The method is complex in the relation with time complexity; nevertheless the gain could be higher. The idea is quite simple and can be expressed in the following definition and theorem.

Definition 3 *Consequent formula*
Formula F is a consequent formula in a refutational proof if there is a formula G in the set of resolvents or axioms, where $G \to F$ holds.

Theorem 2 *Detection of consequent formulas (DCF)*
Formula F is a consequent formula of G if it could be proved by self-resolution on the formula $\neg[G \to F]$ that the formula is contradictory.

PROOF: We can refer to a proof of completeness of the refutational resolution. When we have an inconsistent set of formulas, it is assured by the completeness that it lead to refutation. And since one formula forms a set and the self-resolution is a special case of general resolution, we can say that if $\neg[G \to F]$ is inconsistent then we prove it by self-resolution i.e. we prove that $G \to F$ is a tautology. We may accept for exceptional instances that a consequent formula will be added to the set of resolvents rather than the possibility to end in an infinite loop. This is better alternative, so we can limit the theorem iterations by time or certain number. Then we assure that the theorem procedure is finite and in the worst case the redundant resolvent is added. $\diamond$

Example 4 *DCF theorem application 1*

Consider the formula $[a \vee b] \wedge [\neg b \vee c]$ and we prove that $[a \vee c]$ is a consequence of it.

1. $\neg[[a \vee b] \wedge [\neg b \vee c] \to a \vee c]$
2. $\neg[[\bot \vee b] \wedge [\neg b \vee c] \to \bot \vee b] \vee \neg[[\top \vee b] \wedge [\neg b \vee c] \to \top \vee b] \Rightarrow \bot$

The important aspect of the theorem DCF lies in its simple implementation into an automated theorem prover based on general resolution. The prover handles formulas in the form of syntactical tree. It is programmed a procedure performing general resolution with two formulas on an atom. This procedure is also used for the implementation of the theorem. A "virtual tree" is created from candidate and former resolvent (axiom) connected by negated implication. Then it remains to perform self-resolution on such formula until a logical value is obtained. Let's compare the efficiency of standard strategies and the above-defined one. Consider following examples. It was also used modified notion of partially performed general resolution i.e. not every atom in a premise was replaced by a logical value. This modification may shorten proofs in some cases. Rn means n-th resolvent and the expression in brackets represents premises of it.

Example 5 *DCF usage 1*

F0 :$a \leftrightarrow b \wedge g$. F1 : $b \wedge g \leftrightarrow c$. F2 (¬query) : $\neg[a \leftrightarrow c]$.
R0 [F2&F1] : $[\neg a \vee \neg[b \wedge g]] \wedge [a \vee b \wedge g]$. R1 [R0&F2] : $\neg[b \wedge g] \vee c$.
R2 [R1&F0] : $\neg g \vee c \vee \neg a$. R3 [R2&F2] : $\neg g \vee \neg a$.
R4 [R3&F2] : $\neg g \vee c$. R5 [R4&F0] : $c \vee \neg a$.
R6 [R5&F2] : $\neg a$. R7 [R6&F2] : c.
R8 [R7&F1] : $b \wedge g$. R9 [R8&F1] : $g \leftrightarrow c$.
R10 [R9&F2] : $[g \vee a] \wedge [\neg g \vee \neg a]$. R11 [R10&F1] : $a \vee [b \leftrightarrow c]$.
R12 [R11&R6] : $b \leftrightarrow c$. R13 [R12&F2] : $[b \vee a] \wedge [\neg b \vee \neg a]$.
R14 [R13&F0] : $a \vee [a \leftrightarrow g]$. R15 [R14&F2] : $\neg g \vee \neg c$.
R16 [R15&F1] : $\neg c$. R17 [R16&F2] : a.
R18 [R17&R6] : YES.
Solving time : 0.22 s.

None of standard above-mentioned strategies was able to limit proof but DCF was. Their proof sequence was cancelled after several seconds when it contained more than 300 resolvents. Most of them were the same formulas or its variations.

2.4 Lifting of inferences

Before we start with explanation it should be stated that the following unification process doesn't allow an occurrence of the equivalence connective. It is needed to remove them by the following rewrite rule: $A \leftrightarrow B \Rightarrow [A \rightarrow B] \wedge [B \rightarrow A]$. The general resolution presented was based on the propositional calculus. The lifting of resolution inferences to formulas with indiscriminated variables into universal and existential ones follows below.

Definition 4 *Lifting of general resolution*

$$\frac{F[G] \quad F'[G]}{F[G/\bot] \vee F'[G/\top]}$$

is lifted to

$$\frac{F[G_1, ..., G_k] \quad F'[G'_1, ..., G'_n]}{F\sigma[G/\bot] \vee F'\sigma[G/\top]}$$

where σ is the most general unifier (mgu) of the atoms $G_1, ..., G_k, G'_1, ..., G'_n, G = G_1\sigma$, and in contrast with [Ba97] it is not supposed renaming of the variables for the purposes of the application.

We can suppose, that every variable occurring in a quantifier has its own identifier (for example memory address), which is assigned to variable occurrence. For this open problem it has been devised the extension of most general unifier. At first it is needed to introduce some supporting notions. The following discrimination of existential and universal variables is needed for the Variable Unification Restriction (VUR) definition. When we speak about existential and universal variables, they are related to its notion with respect to the scope of the whole formula e.g. in $[\exists X \forall Y p(X, Y)] \rightarrow a(Z)$ Y variable to be treated as an existential variable, because the p(X,Y) subformula has negative extended polarity. It means, if we translate the formula into clausal form, Y would transform into existential variable. We define the discrimination as follows.

Definition 5 *Discrimination of variables*
Variable quantified by existential (resp. universal) quantifier is said to be globally existential (resp. globally universal), if the extended polarity of the subformula, which owns the quantifier (the quantifier prefixes it), is positive and it is said to be globally universal (resp. globally existential), if the the extended polarity is negative.

If the polarity is both negative and positive, the variable is both globally existential and universal, but it only occurs if there is an equivalence connective in the formula. Since we required to remove this connectives by rewriting, there is not any problem for further definitions. If we speak about variables over scope of another variable, it means that the quantifiers of variables are located in superior nodes of the syntactical tree of the formula (in prenex form they prefixes the quantifier of mentioned variable).

Definition 6 *Variable Unification Restriction*
Variable Unification Restriction holds if one of the conditions 1. or 2. is satisfied:

1. *One of the terms is only a globally universal variable and the second one is not a globally existential variable. (non-existential case)*

2. *One of the terms is a globally universal variable, the second one is a globally existential variable, and every globally universal variable over the scope of the existential one has assigned some term.*

This restriction performs the same job as skolem constants in clausal form. The unifiability of a gl. existential variable into a gl. universal variable is possible only if every gl. universal variable, which the existential variable depends on, has been substituted by a term already. Remark: The above definition determines, that two globally existential variables can't be unified, that is clear. Basically, the mgu idea is following:

Definition 7 *Most general unifier*

1. *When both the terms, which have to be unified, are of the type string, number or functor without parameters then they are unifiable iff its type is the same (e.g. string and string and so on) and their identifiers match.*

2. *When one of the terms is a variable then it is unifiable with the second one iff VUR holds. Then it is supposed that mgu substitutes the first variable to the second one and we do not require renaming, as it is obvious. If both the variables are globally universal then it is not significant, which is selected as the first one. If one of them is existential, we select it as the first.*

3. *If both the terms are of the type functor with arguments, then they are unifiable iff all the arguments are unifiable by the same procedure from the point 1. (The order of unification to be mentioned with respect to unification of variables, so it tries to unify the atom until it expands the mgu from the first ununified term.)*

4. *There is no other possibility to unify two terms, except that two object of unification are the same physically (e.g. during the self-resolution the occurrence of two variables points to the same variable).*

Let's have a look in an example of well-known facts: It doesn't hold $\forall X \exists Y p(X,Y) \models \exists Y \forall X p(X,Y)$ and it holds $\exists Y \forall X p(X,Y) \models \forall X \exists Y p(X,Y)$. (General Y for all X can't be deduced from Y specific for X but contrary it holds.)

Example 6

Source formulas (axioms) :
F0 : $\forall X \exists Y p(X,Y)$. F1 ($\neg$query) : $\forall Y \exists X \neg p(X,Y)$.
R [F1&F1] : $\bot \vee \top$.
R [F0&F0] : $\bot \vee \top$.

In this sample F0 and F1 can't resolve, since $\forall X \exists Y p(X,Y)$ and $\forall Y \exists X \neg p(X,Y)$ have no unifier. It is impossible to substitute universal X from F0 with X from F1, because X from F1 is existential and its superior variable Y is not assigned with a value. Counter-example with variable Y is the same instance and it is not allowed to substitute anything into an existential

variable. However, in the next example a unifier exists. Source formulas (axioms) :
F0 : $\exists Y \forall X p(X, Y)$. F1 ($\neg$query) : $\exists X \forall Y \neg p(X, Y)$.
R [F1&F0] : YES.
R [F0&F1] : YES.

Hereinabove the universal variable X from F0 could be assigned with existential Y because Y in F1 has no superior variable. Then existential Y from F0 can substitute the universal Y from F1 for the same reason.

3 Implementation

The computer realization of the ideas described above is relatively simple, but not trivial. The main obstacles, which occurred during the construction of the prover, all ensued from the conception of variables. The prover was realized on Windows platform and the application has standard control set. The user interface of GERDS is quite simple. It is a MDI (Multiple document interface) application, which means that the user has possibility to open more than one set of source formulas at once. The design is based on Object Oriented Programming with Object Pascal Language. Visual components were used from Borland Delphi Visual Component Library providing also usual efficient container objects. Every independent frame has two parts - Editor for source formulas (axioms) and Output for results of inference. The figure 1 shows GERDS's GUI.

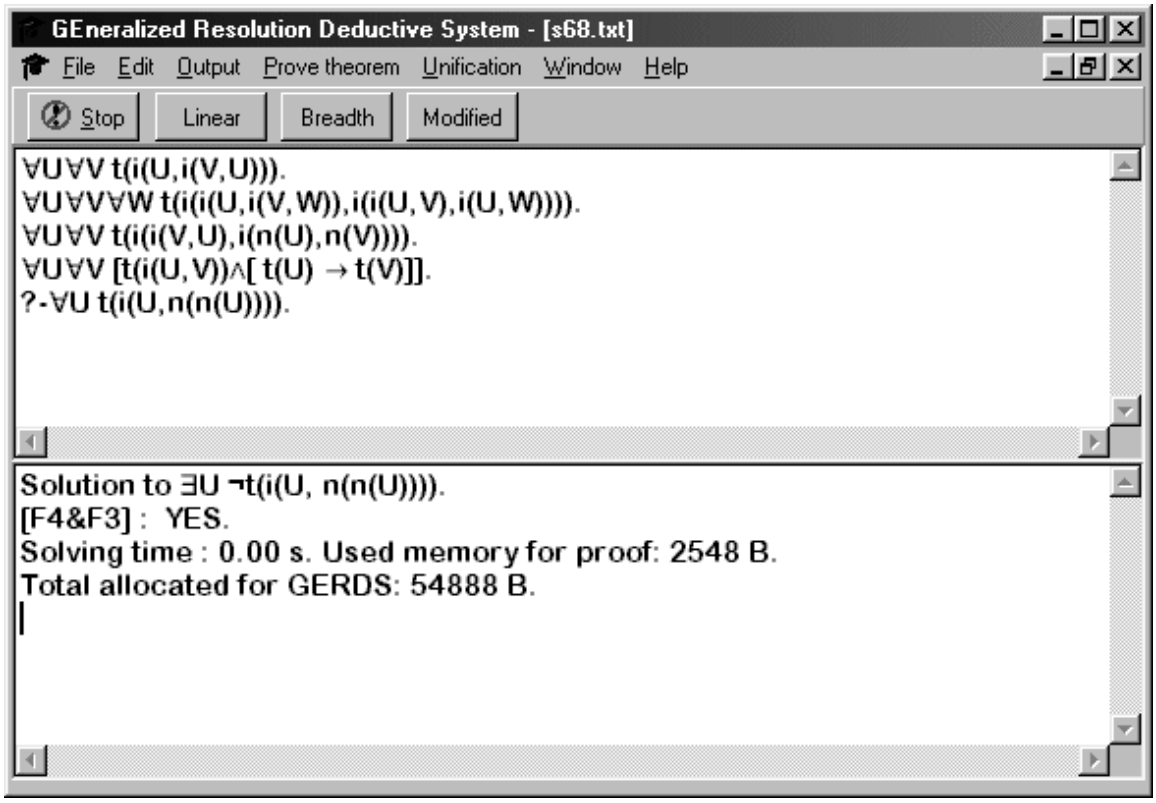

Figure 1: Frame of GERDS

Axioms are written in common mathematical notion and results of inference provide proof sequence with marked premises, which particular resolvent was derived from. It can present

the terms used for unification in goal's variables (PROLOG like). It also offer several types of resolution strategies as visible on the upper panel and several unification and resolution restrictions as well as statistics of inference. The main programmer's issues involved are now summarized:

1. Object model - decomposition of the formula structure into particular object hierarchy e.g. TSub object represents abstract conception of subformulas and TCon is its descendant, which represents conjunction of two subformulas.

2. Compiler - formula parser and generator of a syntactical tree; variables are firstly created in a quantifier related to a node of subformula and their occurrences are not bound to specific quantified variable.

3. Postprocessing - essential modifications of a tree especially assignment of variable occurrences to quantifiers encapsulated in subformulas; it causes problems under removal of unused variables, because the tree is then "hyper" connected by various relations, but this connections are essential for elegant operations on the tree.

4. Inference process - control of refutational proving by resolution strategies.

5. General resolution rule - universal procedure creating disjunction of premises as stated in the definition.

6. Most general unifier - function returning mgu (if exists); in particular the check if Variable unification restriction holds (also there are problems with "hyper" connection of the formula tree).

7. Detection of consequent resolvents - it's fairly elegant, requires only creation of "virtual" negated implication as described in DCF theorem and performing self-resolution using universal resolution procedure.

8. Simplification - removal of logical constants and evaluation of arithmetic expressions.

3.1 Object hierarchy and compiler

In contrast with clausal resolution prover the implementation of non-clausal prover requires more **complex pointer-based data structures** for internal formula representation. At the first sight it may be observed as a significant disadvantage, since simple data structures of CNF representation are easy to create, store and handle and therefore their algorithms should have reasonable time and space complexity.

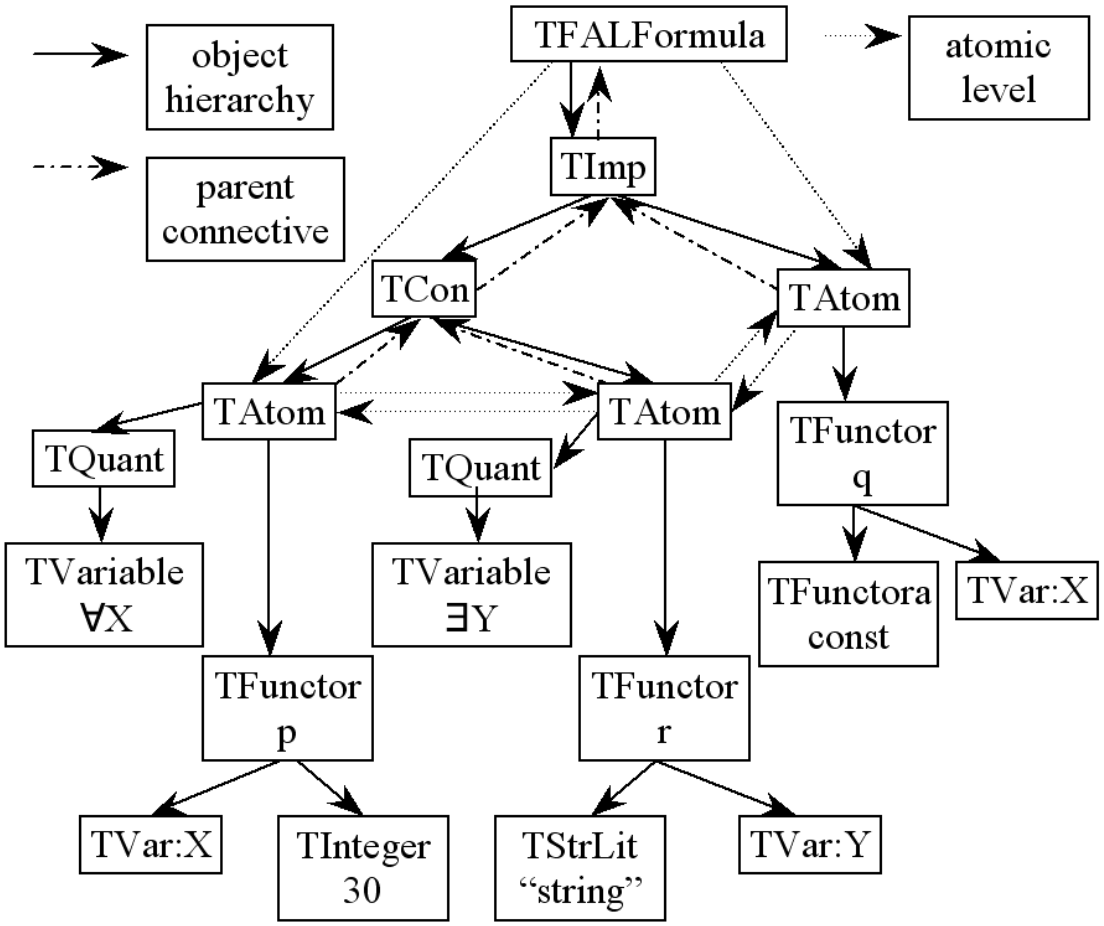

Figure 2: Example of formula data structure (before postprocessing) for:

$$\forall \mathbf{Xp}(\mathbf{X}, \mathbf{30}) \wedge \exists \mathbf{Yr}(\text{"}\mathbf{string}\text{"}, \mathbf{Y}) \rightarrow \mathbf{q}(\mathbf{const}, \mathbf{X})$$

It could be observed from the figure 2, how the formula data structure is constructed. After compilation the syntactical tree is constructed without variable occurrences links to specific quantifier. The hierarchy of syntactical elements could be specified in Backus-Naur form (for details see [Ha00]). Abstract class TSub represents general subformula of FOL, which has its specifications according to specific type of logical connective (e.g. TCon - conjunction, TImp - implication etc.).

```
TSub = class
    Neg : boolean;  { Flag of logical negation. }
    Ev : shortint;  { Indicator of the subformula logical value. }
    Pol : shortint; { Stores the polarity of the node. }
    L : TSub;
    R : TSub;         { Left and right subtree. }
    Ac : TObject;    { Parent object (logical connective). }
    Q : TQuant;     { Quantifier containing variables at this level. }
    ...
end;
```

Constructed tree is built and linked in three basic levels:

1. Object hierarchy - provides standard linkage of parent-child relations.
2. Parent connective - enables to quickly evaluate and simplify the tree.
3. Atomic level - provides access to atoms (literals) in linear time for simpler general resolution rule application.

Parser is based on LL(1) grammar for FOL and it is constructed using recursive descent parsing technique (RDP). It produces a procedure for every non-terminal symbol according to the string that is contained in production for particular syntactical structure. Advantage of RDP lies mainly in the possibility to perform compilation phase simultaneously with parsing. The algorithm in each non-terminal related procedure performs not only check of syntactical correctness but also stack based creation of the tree (both on logical and term level). In [Ha00] it can be seen which procedures create the tree (or directly in source codes [Ha99]).

Once the original tree is built the postprocessing phase may continue. Its main aim is to establish links (pointers) to appropriate quantifiers, to evaluate polarities of subformulas and to evaluate infix operators in the formula. This implementation method provides as with **structure containing all necessary information built simultaneously in these two phases**.

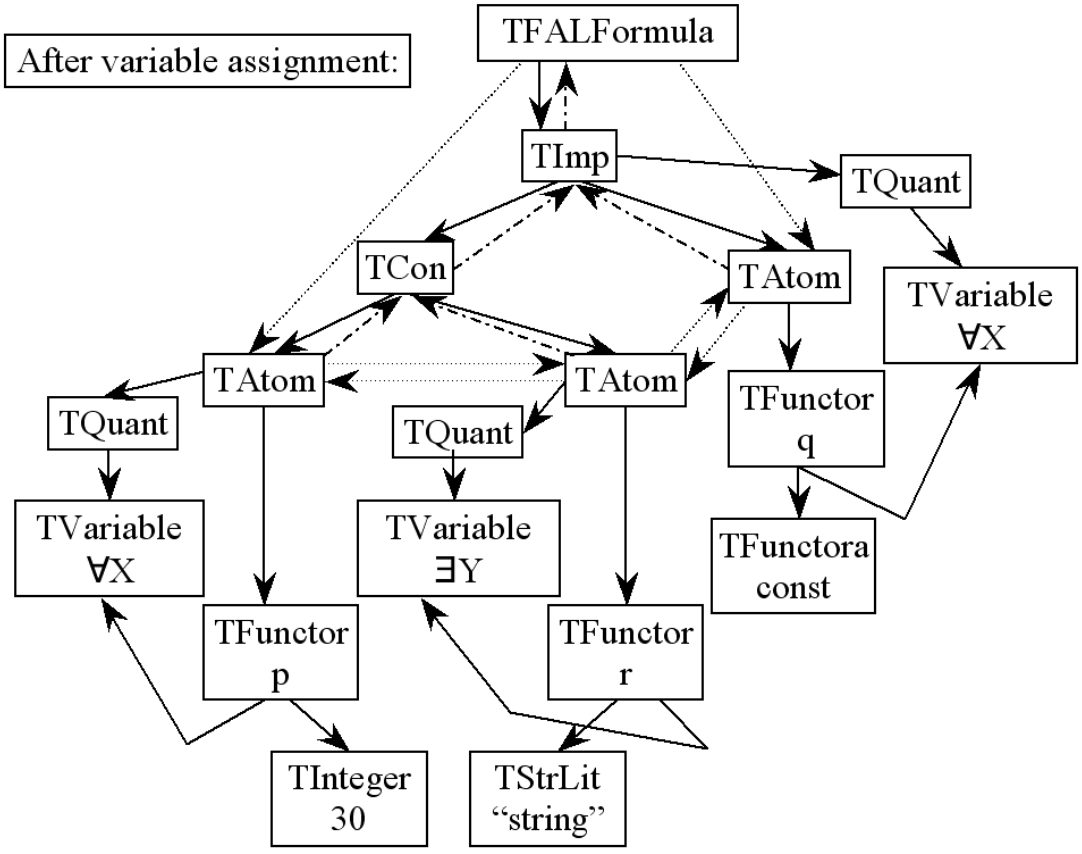

Figure 3: Example of formula data structure (postprocessed)

3.2 Inference engine

Inference engine is based on general resolution rule procedure for production of resolvent on two formulas of FOL. The base procedure controlling the process tries to resolve upon all possible

premises (it contains also filtrating mechanism of resolution strategies). It checks the consistency of the axiom set with negated query formula. The core procedure performs the general resolution rule. Creation of a resolvent is relatively simple. It requires only creation of premise copies and unification procedure. When then unification exists it is possible to **link copies of premises by disjunction** (TDis object) and to **substitute terms** generated by unification (there is **no** need to **replace all occurrences** but only quantifier object in one memory cell). DCF theorem may use existing general resolution procedure since self-resolution is its special case.

3.3 Examples and comparison of GERDS with the clausal resolution through SPASS

The Max-Planck-Institut für Informatik in Germany has developed effective software tool for theorem proving [Wi01]. Since it is a full first-order logic theorem prover with equality and GERDS currently does not support equality all examples will be based on non-equality problems. Comparing problems are taken from the SPASS [Wi99] and they are specially selected to demonstrate advantages of non-clausal prover (GERDS). It is clear that SPASS is more practical prover than GERDS, nevertheless the GERDS could solve some special classes of formulas more efficently.

Example 7 *Pelletier's Problem No. 71*

$? - p1 \leftrightarrow [p2 \leftrightarrow [p3 \leftrightarrow [p4 \leftrightarrow [p5 \leftrightarrow [p1 \leftrightarrow [p2 \leftrightarrow [p3 \leftrightarrow [p4 \leftrightarrow p5]]]]]]]].$
This simple sequence of equivalencies seems to be valid, but how to prove it? If we utilize clausal prover (SPASS) 32 clauses are generated from the goal. Then it kept 494 clauses during the proving process. In contrast the GERDS produced a short proof by general resolution (including SR_check of the resolvent):
Solution to $\neg[p1 \leftrightarrow p2 \leftrightarrow p3 \leftrightarrow p4 \leftrightarrow p5 \leftrightarrow p1 \leftrightarrow p2 \leftrightarrow p3 \leftrightarrow p4 \leftrightarrow p5]$. SR_check:$\neg[p3 \leftrightarrow p4 \leftrightarrow p5 \leftrightarrow p3 \leftrightarrow p4 \leftrightarrow p5] \vee [p3 \leftrightarrow p4 \leftrightarrow p5 \leftrightarrow \neg[p3 \leftrightarrow p4 \leftrightarrow p5]] \vee [p3 \leftrightarrow p4 \leftrightarrow p5 \leftrightarrow \neg[p3 \leftrightarrow p4 \leftrightarrow p5]] \vee \neg[p3 \leftrightarrow p4 \leftrightarrow p5 \leftrightarrow p3 \leftrightarrow p4 \leftrightarrow p5]$.
SR_check:$\neg[p4 \leftrightarrow p5 \leftrightarrow p4 \leftrightarrow p5] \vee [p4 \leftrightarrow p5 \leftrightarrow \neg[p4 \leftrightarrow p5]] \vee [p4 \leftrightarrow p5 \leftrightarrow \neg[p4 \leftrightarrow p5]] \vee \neg[p4 \leftrightarrow p5 \leftrightarrow p4 \leftrightarrow p5] \vee [p4 \leftrightarrow p5 \leftrightarrow \neg[p4 \leftrightarrow p5]] \vee \neg[p4 \leftrightarrow p5 \leftrightarrow p4 \leftrightarrow p5] \vee \neg[p4 \leftrightarrow p5 \leftrightarrow p4 \leftrightarrow p5] \vee [p4 \leftrightarrow p5 \leftrightarrow \neg[p4 \leftrightarrow p5]]$.
SR_check:$\neg[p5 \leftrightarrow p5] \vee [p5 \leftrightarrow \neg p5] \vee [p5 \leftrightarrow \neg p5] \vee \neg[p5 \leftrightarrow p5] \vee [p5 \leftrightarrow \neg p5] \vee \neg[p5 \leftrightarrow p5] \vee \neg[p5 \leftrightarrow p5] \vee [p5 \leftrightarrow \neg p5] \vee [p5 \leftrightarrow \neg p5] \vee \neg[p5 \leftrightarrow p5] \vee \neg[p5 \leftrightarrow p5] \vee [p5 \leftrightarrow \neg p5] \vee \neg[p5 \leftrightarrow p5] \vee [p5 \leftrightarrow \neg p5] \vee [p5 \leftrightarrow \neg p5] \vee \neg[p5 \leftrightarrow p5]$.
SR_check: $\perp$.
R [F0&F0] : YES.
Solving time : 0.00 s. Used memory for proof: 920 B.

Let's have a look to a non-propositional example.

Example 8 *Pelletier's Problem No. 68*

It consist of these axioms: $\forall U \forall V t(i(U, i(V, U)))$.
$\forall U \forall V \forall W t(i(i(U, i(V, W)), i(i(U, V), i(U, W))))$.
$\forall U \forall V t(i(i(V, U), i(n(U), n(V))))$.
$\forall U \forall V [t(i(U, V)) \wedge [t(U) \rightarrow t(V)]]$. and this goal:
?-$\forall U t(i(U, n(n(U))))$.

The SPASS had a short proof here, it kept 3 only clauses, but GERDS is also efficient:
Solution to $\exists U \neg t(i(U, n(n(U))))$.
R [F4&F4] : $\bot \vee \top$.
R [F4&F3] : $\bot \vee [\bot \wedge [t(U) \rightarrow t(n(n(U)))]]$.
R [F4&F3] : YES.
Solving time : 0.00 s. Used memory for proof: 2592 B.

4 Further research and applications

The article presented theory and practice of non-clausal resolution. It tried to present this deduction principle not only as an obsolete and uninteresting technique, but as powerful formalization, which could be used for the following main objectives (current and perspective applications).

1. **Fuzzy general resolution:**
   The notion of general resolution may be in simple form transformed into fuzzy logic (clausal resolution principle was studied in [Le95]. Since the problem of theorem proving in fuzzy logic is much more complicated (normal forms are very hard to obtain), the possibility to perform resolution on general formulas of fuzzy predicate logic seems to be very interesting. Fuzzy general resolution was formulated in [Ha02]. For the purposes of fuzzy extension the Modus ponens rule was considered as an inspiration. We will suppose that set of truth values is Łukasiewicz algebra. Therefore we will assume standard notions of conjunction, disjunction etc. to be bound with Łukasiewicz operators.

   We will assume Łukasewicz algebra to be

   $$\mathcal{L}_{\text{Ł}} = \langle [0,1], \wedge, \vee, \otimes, \rightarrow, 0, 1 \rangle$$

   where [0, 1] is the interval of reals between 0 and 1, which are the smallest and greatest elements respectively. Basic and additional operations are defined as follows:

   $$\begin{aligned} a \otimes b &= 0 \vee (a + b - 1) && (1)\\ a \rightarrow b &= 1 \wedge (1 - a + b) && (2)\\ a \oplus b &= 1 \wedge (a + b) && (3)\\ \neg a &= 1 - a && (4) \end{aligned}$$

   The biresiduation operation $\leftrightarrow$ could be defined $a \leftrightarrow b =_{df} (a \rightarrow b) \wedge (b \rightarrow a)$, where $\wedge$ is infimum operation.

   Syntax and semantics of propositional fuzzy logic is following:

   - propositional variables, logical constants – $\{\mathfrak{a} \,|a \in [0,1]\}$. Instead of 0 we write $\bot$ and instead of 1 we write $\top$, connectives - **&** (Łukasiewicz conjunction), $\wedge$ (conjunction), ∇ (Łukasiewicz disjunction), $\vee$ (disjunction), $\Rightarrow$ (implication), $\Leftrightarrow$ (equivalence), $\neg$ (negation) and furthermore by F_J we denote set of all formulas of fuzzy logic in language J
   - connectives have the following semantic interpretations: $\mathcal{D}(\mathfrak{a}) = a$ for $a \in [0,1]$, $\mathcal{D}(A \,\&\, B) = \mathcal{D}(A) \otimes \mathcal{D}(B)$, $\mathcal{D}(A \wedge B) = \mathcal{D}(A) \wedge \mathcal{D}(B)$, $\mathcal{D}(A \nabla B) = \mathcal{D}(A) \oplus \mathcal{D}(B)$, $\mathcal{D}(A \vee B) = \mathcal{D}(A) \vee \mathcal{D}(B)$, $\mathcal{D}(A \Rightarrow B) = \mathcal{D}(A) \rightarrow \mathcal{D}(B)$, $\mathcal{D}(A \Leftrightarrow B) = \mathcal{D}(A) \leftrightarrow \mathcal{D}(B)$, $\mathcal{D}(\neg A) = \neg \mathcal{D}(A)$

Graded fuzzy propositional calculus assigns grade to every axiom, in which the formula is valid. It will be written as

$$a/A$$

where A is a formula and a is a syntactic evaluation. More detailed overview of fuzzy logic properties could be found in [Hj97] or [No99].

Definition 8 *General fuzzy resolution*

$$r_R : \frac{a/F_1[G],\ b/F_2[G]}{a \otimes b/F_1[G/\bot]\nabla F_2[G/\top]} \tag{5}$$

where F_1 holding in the degree a and F_2 holding in the degree b are formulas – premises of fuzzy logic and G is an occurrence of a subformula in F_1 and F_2. The expression below the line means the resolvent of premises on G holding in the degree $a \otimes b$. Every occurrence of G is replaced by false in the first formula and by true in the second one.

Clearly also the implementation of this rule for should be very natural (at least for interval $[0, 1]$ of truth degrees). It remains only to extend the object encapsulating formula with data structure representing truth degree. We are unable to show whole theoretical background of this technique in this article, but we can observe the behaviour of the rule on simple propositonal example based on fuzzy propositional calculus with graded formal proof.

Example 9 *Proof of d by r_R*

Xa. steps represent application of simplification rules for $\bot$ and $\top$.

1. $1/[a \Rightarrow b]$	(axiom with grade 1)
2. $0.9/[(b \,\&\, c) \Rightarrow d]$	(axiom with grade 0.9)
3. $0.8/c$	(axiom with grade 0.8)
4. $0.95/a$	(axiom with grade 0.95)
5. $0.95 \otimes 1/[\bot\nabla(\top \Rightarrow b)]$	(r_R on 4, 1)
5a. $0.95/b$	(simplification for $\Rightarrow$, ∇)
6. $0.95 \otimes 0.9/[\bot\nabla((\top \,\&\, c) \Rightarrow d)]$	(r_R on 5a, 2)
6a. $0.85/[c \Rightarrow d]$	
7. $0.8 \otimes 0.85/[\bot\nabla(\top \Rightarrow d)]$	(r_R on 3, 6a)
7a. $0.65/d$	

We have found a proof of d with the grade 0.65.

Of course, many questions arise in the frame of usage of the rule in fuzzy logic, e.g. algorithm for finding the best provability degree or refutation-based proof search by fuzzy general resolution.

2. **Deduction principle for Description Logic:**
Description logics (DL) for semantic web are widely studied nowadays [Bd03]. Its expression power together with some decidability properties gives DL significant advantage in practical application - knowledge representation and deduction. There are several deduction principles for DL, e.g. semantic tableau [Lu05]. The non-clausal resolution should

be natural notion of deduction for DL since it enables to be applied without any transformation of original formula. The work on utilization the general resolution rule is in formalization at present, so we present it only on an example processed by GERDS. Note that we will express the example in the standard notation of FOL (for clarity). We use ALC-style DL (but the power of general resolution rule enables as to use it in every DL based on FOL without any transformations or modifications.

Example 10 *Non-clausal resolution in Description Logic*

Suppose we have the following concepts:
person, male, female and *happy_child*
and *role is_child_of* representing the role of being X child of Y.
We would like to express the knowledge that:
If person is child of at least one male and one female, then it is a happy child (since has both mother and father).
It can be expressed in ALC-like DL after rewriting the implication into disjunction like:
$\neg(person \sqcap \exists is\_child\_of.female \sqcap \exists is\_child\_of.male) \sqcup happy\_child$
Then we will suppose the following instances:
is_child_of(johana, hashim). is_child_of(johana, lucie). male(hashim). female(lucie). person(johana).
GERDS will generate the following proof (rewritten to FOL!). Note that we used detailed output with unsimplified logical constants - GERDS may produce several types of outputs.
Source formulas (axioms) :

$F0:$ $\forall X[person(X) \wedge \exists Y[is\_child\_of(X,Y) \wedge female(Y)]$
$\wedge \exists Y[is\_child\_of(X,Y) \wedge male(Y)] \rightarrow happy\_child(X)]$.
$F1:$ $is\_child\_of(johana, hashim)$.
$F2:$ $is\_child\_of(johana, lucie)$.
$F3:$ $male(hashim)$.
$F4:$ $female(lucie)$.
$F5:$ $person(johana)$.
$F6(\neg query):$ $\neg happy\_child(johana)$.
$[F6\&F6]:$ $\bot \vee \top$.
$[F6\&F0]:$ $\bot \vee [person(johana) \wedge \exists Y[is\_child\_of(johana,Y) \wedge female(Y)]$
$\wedge \exists Y[is\_child\_of(johana,Y) \wedge male(Y)] \rightarrow \bot]$.
$R0[F6\&F0]:$ $\neg[person(johana) \wedge \exists Y[is\_child\_of(johana,Y) \wedge female(Y)]$
$\wedge \exists Y[is\_child\_of(johana,Y) \wedge male(Y)]]$.
$[R0\&F5]:$ $\neg[\top \wedge \exists Y[is\_child\_of(johana,Y) \wedge female(Y)]$
$\wedge \exists Y[is\_child\_of(johana,Y) \wedge male(Y)]] \vee \bot$.
$R1[R0\&F5]:$ $\neg[\exists Y[is\_child\_of(johana,Y) \wedge female(Y)]$
$\wedge \exists Y[is\_child\_of(johana,Y) \wedge male(Y)]]$.
$[R1\&F4]:$ $\neg[[is\_child\_of(johana, lucie) \wedge \top]$
$\wedge \exists Y[is\_child\_of(johana,Y) \wedge male(Y)]] \vee \bot$.
$R2[R1\&F4]:$ $\neg[is\_child\_of(johana, lucie)$
$\wedge \exists Y[is\_child\_of(johana,Y) \wedge male(Y)]]$.
$[R2\&F3]:$ $\neg[is\_child\_of(johana, lucie)$
$\wedge [is\_child\_of(johana, hashim) \wedge \top]] \vee \bot$.
$R3[R2\&F3]:$ $\neg[is\_child\_of(johana, lucie) \wedge is\_child\_of(johana, hashim)]$.
$[R3\&F2]:$ $\neg[\top \wedge is\_child\_of(johana, hashim)] \vee \bot$.
$R4[R3\&F2]:$ $\neg is\_child\_of(johana, hashim)$.
$[R4\&F1]:$ $\bot \vee \bot$.
$[R4\&F1]:$ YES.

Inference statistics : Resolvents : 5, Removed tautologies : 1, Unification : 774, Simplification : 234
Solving time : 0.03 s. Used memory for proof: 4628 B.
Total allocated for GERDS: 48616 B.

3. **Framework for deduction in Fuzzy Description Logic:**
Integration of fuzzy general resolution and non-clausal resolution should lead to simple, but general theorem proving for fuzzy description logic [Sr01]. Fuzzy DL is already formalized and there is a recent paper concerning Fuzzy DL [Hj05]. This paper presents necessary notions, algorithms and propeties e.g. satisfiability, validity. Our future work will be focused mainly into mentioned integration. It requires both implementation and theoretical formalization of general fuzzy description logic resolution principle.

References

[Ba97] Bachmair L., Ganzinger H. A theory of resolution. Technical report: Max-Planck-Institut für Informatik, 1997.

[Ba01] Bachmair, L., Ganzinger, H. Resolution theorem proving. in Handbook of Automated Reasoning, MIT Press, 2001.

[Bd03] Baader et col. (eds.). The Description Logic Handbook - Theory, Interpretation and Applications. Cambridge Univ. Press, 2003.

[Ha99] Habiballa, H. GERDS for Windows.
http://www.volny.cz/habiballa/files/gerds.zip

[Ha00] Habiballa, H. Non-clausal resolution - theory and practice. Research report: University of Ostrava, 2000, http://www.volny.cz/habiballa/files/gerds.pdf

[Ha02] Habiballa, H., Novák, V. Fuzzy general resolution. Research report: Institute for research and applications of fuzzy modeling, University of Ostrava, 2002, http://ac030.osu.cz/irafm/ps/rep47.ps

[Hj97] Hájek, P. Metamathematics of fuzzy logic, Kluwer Academic Publishers - Dordrecht, 2000

[Hj05] Hájek, P. Making fuzzy description logic more general. Research report: Institute of Computer Science, Czech Academy of Sciences, 2005.

[Le95] Lehmke, S. On resolution-based theorem proving in propositional fuzzy logic with 'bold' connectives. Diploma thesis:University of Dortmund, 1995

[Lu05] Lukasová, A. Reasoning in Description Logic with semantic tableua binary trees. Resarch report: Institute for Research and Applications of Fuzzy Modeling, University of Ostrava, 2005.

[Mu82] Murray, N. Completely non-clausal theorem proving. Artificial Intelligence, 18, 1982, 6785.

[Sr01] Straccia, U. Reasoning within Fuzzy Description Logics. Journal of Articial Intelligence Research 14 (2001), pp. 137-166.

[No99] Novák, V., Perfilieva, I., Močkoř, J. Mathematical principles of fuzzy logic, Kluwer Academic Publishers, 1999

[St96] Stachniak, Z. Resolution Proof Systems: An Algebraic Theory, Kluwer Academic Publishers, 1996.

[Ta03] Tammet, T. Extending Classical Theorem Proving for the Semantic Web. CEUR Workshop Proceedings, Technical University of Aachen (RWTH), 2003.

[Wi99] Wiedenbach, Ch. SPASS for Windows. http://spass.mpi-sb.mpg.de.

[Wi01] Weidenbach, Ch. SPASS: Combining Superposition, Sorts and Splitting. In A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning, Elsevier