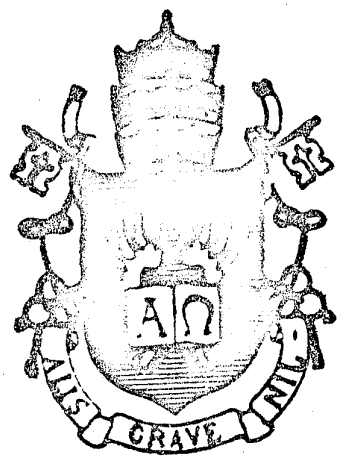

PUC

Series:Monografias em Ciência da Computação,
No. 21/91

LINEAR RESOLUTION, DEFINITIONS AND FUNCTIONS

Roberto Lins de Carvalho
Newton José Vieira

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Series: Monografias em Ciência da Computação, No. 21/91
Editor: Carlos J. P. Lucena November , 1991

LINEAR RESOLUTION, DEFINITIONS AND FUNCTIONS *

Roberto Lins de Carvalho **
Newton José Vieira ***

* This work has been sponsored by Secretaria de Ciência e Tecnologia of Presidência da República Federativa do Brasil and SERC under GR/G24644.

** PUC Rio and LNCC
*** Universidade Federal de Minas Gerais.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio - Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
Brasil

Tel.:(021)529-9386 Telex:31078 Fax:(021)511-5645
E-mail:rosane@inf.puc-rio.br

Abstract:

In this paper we analyse the use of definitions and functions in the context of proof procedures based on the resolution principle (particularly linear resolution). By leting the proof procedure "knowing" what formulas are definitions and what predicates denote functional relations, it can take advantage from this in order to improve its performance.

Linear Resolution, Definitions and Functions

Roberto Lins de Carvalho*
LNCC-CNPq

Newton José Vieira
Universidade Federal de Minas Gerais

October 31, 1991

Abstract

In this paper we analyse the use of definitions and functions in the context of proof procedures based on the resolution principle (particularly linear resolution). By leting the proof procedure *knowing* what formulas are definitions and what predicates denote functional relations, it can take advantage from this in order to improve its performance.

1 Introduction

In this paper we analyse the use of definitions and functions in the context of proof procedures based on the resolution principle (particularly linear resolution). By leting the proof procedure *knowing* what formulas are definitions and what predicates denote functional relations, it can take advantage from this in order to improve its performance.

Definitions, being non-creative axioms, are susceptible to special treatment. This has implications in the naturalness of deductions and in the efficiency of the inference engine.

We start, in this work, with a logic without equality. A restricted use of equality to deal with the functionality of some functional relations is then proposed.

In sections 2 and 3 we present the essencials about linear resolution and theory of definitions, respectively, in order to set the framework for the later sections.

In section 4 we analyse some problems that arise when the proof procedure don't recognize definitions as such, and suggest a special treatment for definitions.

*Supported by SERC under GR/G24644

In section 5 we present some approachs to the problem of functional relations and related treatment of the equality relation in special situations (such as the use of functional relations in the context of definitions).

In section 6 we present some conclusions, and in section 7 we list the references.

2 Linear Resolution

Linear resolution is a refinement of resolution [Robinson 65] that was proposed independently in [Loveland 68] and [Luckham 68]. In linear resolution any resolvent is obtained from the previous resolvent and an input clause or another resolvent, except the "first" resolvent, which is obtained from a *selected input clause* and another input clause.

Linear resolution produces *linear deductions*. It is customary to present a linear deduction as a sequence S of clauses

$$C_1, \ldots, C_n$$

where a sequence of input clauses constitutes a prefix of S, and every other clause C_i is obtained by resolving C_{i-1} against some clause C_j for $j < i$. We call this prefix the *basic prefix*, and the last clause of the basic prefix is the above mentioned *selected input clause*. A linear deduction of $\Box$ (the empty clause) is called a *linear refutation*.

Linear resolution is complete when the selected input clause C is such that $S - \{C\}$ is unsatisfiable.

One special kind of linear resolution is *linear input resolution*. In this case one of the premises in any application of resolution is always an input clause. Implementations of linear input resolution are usually more efficient than implementations of irrestricted linear resolution. However, linear input resolution is not complete, as can be seen by the following example.

Exemplo 2.1 *The set of clauses*

$P(x) \vee Q(x)$
$\neg P(a) \vee Q(b)$
$\neg Q(y) \vee R(y)$
$\neg R(z)$

is obviously unsatisfiable. But if we take $\neg R(z)$ as the selectec input clause (without this clause the set is satisfiable) we can not obtain a linear input refutation.

A linear refutation is:

1. $P(x) \vee Q(x)$

2. $\neg P(a) \vee Q(b)$

3. $\neg Q(y) \vee R(y)$

4. $\neg R(z)$ (selected input clause)

5. $\neg Q(z)$ (from 4 and 3)

6. $P(z)$ (from 5 and 1)

7. $Q(b)$ (from 6 and 2)

8. $\square$ (from 7 and 5)

In the context of *answer extraction* [Green 69] resolving two resolvents implies in general an answer with more than one disjunct. This is the case in the example: the answer to $\exists z R(z)$ is $R(a) \vee R(b)$.

In question answering systems where answers with more than one disjunct are not required, linear input resolution can be appropriate.

One of the more efficient technologies to implement linear input resolution is that based on the usual implementations of the PROLOG language. But the same technology (with adaptations) could in principle be used to implement a complete linear resolution based procedure: the model elimination procedure [Loveland 69]. Proof procedures similar to model elimination, but without the "ordered clauses" restriction imposed by model elimination and related procedures (SL-resolution, for example [Kowalski 71]) are proposed in [Vieira 87]. In this later work, it is given special attention to the problem of *answer explanation*. The possibility of efficient implementations, as well of natural explanations of deductions, which are essential in *knowledge processing systems*, justify our preference on using linear resolution in first place as a vehicle to expose our ideas.

3 Definitions

From a logic point of view, definitions must meet some requirements. Such requirements give some properties to the defined symbols that distinguish them from the other symbols (the primitive symbols). Few proof procedures take this into account. It is our purpose to call attention to a possible use of definitions to decrease the amount of work done in proving theorems.

One of the requirements that a definition must meet is that it be possible to eliminate the defined symbol from any expression in which it appears. This requirement, known as the *requirement of eliminability*, implies that definitions

must not be circular. More, it implies that whatever can be said with the help of defined symbols can be said without this help.

More formaly, a formula α introducing a new symbol of a theory satisfies the requiment of eliminability if and only if: whenever β is a formula in which the new symbol occurs, then there is a formula γ in which the new symbol does not occur, such that $(\alpha \Rightarrow (\beta \Leftrightarrow \gamma))$ is derivable from the axioms and preceding definitions of the theory.

Another requirement that a definition must meet is that any formulas not containing the defined symbol that can be proved, can be proved without the help of the definition. In other words, a definition must not increase what can be said without the help of the definition. This requirement is known as the *requirement of non-creativity*, and implies in particular that if a given theory is consistent, then if we add a definition to the theory it will remain consistent.

More formaly, a formula α introducing a new symbol of a theory satisfies the requiment of non-creativity if and only if: there is no formula β in which the new symbol does not occur such that $(\alpha \Rightarrow \beta)$ is derivable from the axioms and preceding definitions of the theory, but β is not so derivable.

Then, in a certain sense, definitions are *non-creative axioms* while other axioms of a theory are *creative axioms.*

In the following definitions we will show standard forms of making definitions according to the type of symbol to be defined (predicate or function symbol). Such standard forms were devised to meet the above two requirements.

Definition 1 *A formula α introducing a new n-ary predicate symbol P is a definition of P in a theory if α is of the form*

$$\forall x_1 \forall x_2 \ldots \forall x_n (P(x_1, x_2, \ldots, x_n) \Leftrightarrow \psi(x_1, x_2, \ldots, x_n))$$

and the following restrictions are satisfied:

(a) $x_1, x_2, \ldots, x_n$ are distinct variables;

(b) $\psi(x_1, x_2, \ldots, x_n)$ has no free variables other than $x_1, x_2, \ldots, x_n$;

(c) $\psi(x_1, x_2, \ldots, x_n)$ is a formula in which the only non-logical constants are primitive symbols and previously defined symbols of the theory.

In clause form the definition of a predicate symbol has the form:

$$D_+^P \cup D_-^P$$

where D_+^P (positive part of the definition) and D_-^P (negative part of the definition) of P are:

- $D_+^P = \neg P(x_1, x_2, \ldots, x_n) \vee S_{\psi(x_1, x_2, \ldots, x_n)}$
- $D_-^P = P(x_1, x_2, \ldots, x_n) \vee S_{\neg \psi(x_1, x_2, \ldots, x_n)}$

Definition 2 *A formula α introducing a new n-ary function symbol f is a definition of f in a theory if α is of the form*

$$\forall x_1 \forall x_2 \ldots \forall x_n \forall y (f(x_1, x_2, \ldots, x_n) = y \Leftrightarrow \psi(x_1, x_2, \ldots, x_n, y))$$

and the following restrictions are satisfied:

(a) $x_1, x_2, \ldots, x_n, y$ *are distinct variables;*

(b) $\psi(x_1, x_2, \ldots, x_n, y)$ *has no free variables other than* $x_1, x_2, \ldots, x_n, y$*;*

(c) $\psi(x_1, x_2, \ldots, x_n, y)$ *is a formula in which the only non-logical constants are primitive symbols and previously defined symbols of the theory.*

(d) $\forall x_1 \forall x_2 \ldots \forall x_n \exists! y (\psi(x_1, x_2, \ldots, x_n, y))$[1] *is a formula derivable from the axioms and previously defined symbols of the theory.*

4 Linear Resolution with Definitions

In this section we will analyse some problems that might occur when we don't recognize or use definitions in a special way. We will do so in the context of linear resolution by means of examples. For that purpose we will use in the examples the following definitions concerned with the elementary algebra of classes:

I	$\forall x \forall y (x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x))$	(definition of "$=$")
II	$\forall x \forall y (x \subseteq y \Leftrightarrow \forall z (z \in x \Rightarrow z \in y))$	(definition of "$\subseteq$")
III	$\forall x \forall y \forall z (z \in (x \cap y) \Leftrightarrow (z \in x \wedge z \in y))$	(definition of "$z \in (x \cap y)$")[2]
IV	$\forall x \forall y \forall z (z \in (x \cup y) \Leftrightarrow (z \in x \vee z \in y))$	(definition of "$z \in (x \cup y)$")

The corresponding clauses are shown bellow:

I.1	$\neg x = y \vee x \subseteq y$	**III.1**	$\neg z \in (x \cap y) \vee z \in x$
I.2	$\neg x = y \vee y \subseteq x$	**III.2**	$\neg z \in (x \cap y) \vee z \in y$
I.3	$x = y \vee \neg x \subseteq y \vee \neg y \subseteq x$	**III.3**	$z \in (x \cap y) \vee \neg z \in x \vee \neg z \in y$
II.1	$\neg x \subseteq y \vee \neg z \in x \vee z \in y$	**IV.1**	$\neg z \in (x \cup y) \vee z \in x \vee z \in y$
II.2	$x \subseteq y \vee f(x, y) \in x$	**IV.2**	$z \in (x \cup y) \vee \neg z \in x$
II.3	$x \subseteq y \vee \neg f(x, y) \in y$	**IV.3**	$z \in (x \cup y) \vee \neg z \in y$

The positive part of the definition of $=$ $D^{=}_{+}$ is given by the clauses **I.1** and **I.2** and the negative part $D^{=}_{-}$ by the clause I.3.

[1] "$\exists! x(\psi(x))$" is a notation for "$\exists x(\psi(x) \wedge \forall y(\psi(y) \Rightarrow y = x))$".

[2] This is not definition of "$\cap$", and the next definition is not definition of "$\cup$".

The concepts represented by the symbols $=$, $\subseteq$, $\cap$ and $\cup$ are redutible to the primitive concept $\in$, as can be seen by the *definitional structure* depicted in figure 1.

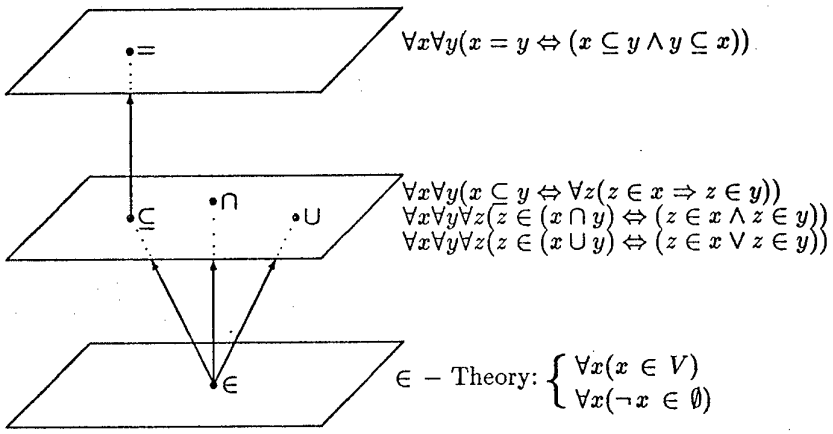

Figure 1: Definitional Structure

In the following example we show a proof of a simple theorem, using the above definitions as axioms.

Exemplo 4.1 *Using linear resolution we can prove the theorem*

$$\forall x (x \subseteq x) \tag{1}$$

One possible proof is:

1. $\neg A \subseteq A$ *(skolemizing the negation of (1))*

2. $f(A, A) \in A$ *(from 1 and II.2)*

3. $A \subseteq A$ *(from 2 and II.3)*

4. $\square$ *(from 3 and 1)*

When a proof procedure process example 4.1, the first "doubt" appears when it tries to choose a clause to resolve against clause 1 ($\neg A \subseteq A$). Here it has two possibilities:

(a) to choose clause I.1 or I.2; or

(b) to choose clause II.2 or II.3.

Any choice will produce a "boomerang efect" in terms of the definitional structure: the choice taken in example 4.1 (clause II.2) provokes a first walk one level down and then one level up (returning to the previous level), and as can be seen bellow, choosing clause I.1 will cause a first walk one level up and then one level down.

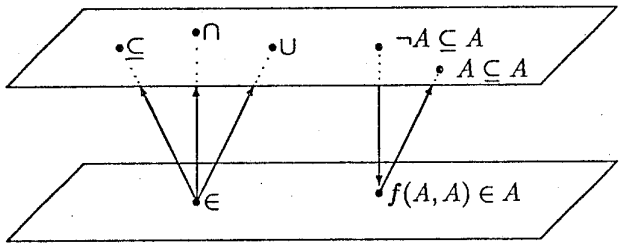

Figure 2: Boomerang effect

1. $\neg A \subseteq A$ (skolemizing the negation of (1))

2. $\neg A = A$ (from 1 and I.1)

3. $\neg A \subseteq A$ (from 2 and I.2)

This last sequence is obviously innocuous, as it must be: the definition of "$=$", as there is not any creative axiom involving "$=$", could not help to prove a formula not involving this symbol.

The "boomerang effect" can cause the possibility of having a lot of clauses as candidates to resolve against the last resolvent. In example 4.1 clause 2 could be resolved against II.1, II.3, III.3, IV.2 or IV.3, and any other clause from a definition that has "$\in$" in its body: the walk one level up could be made in the wrong direction. In other words, the symbols in a connected definitional structure are interwoven in such a way that all concepts involved are potentialy reachable. In terms of a resolution based proof procedure, this might imply that it can make a lot of fruitless deductions.

Another aspect to be considered in example 4.1 is the specific refutation encountered. From the human point of view, it is not *natural* (the boomerang behaviour is not natural). A more natural refutation would be:

1. $\neg A \subseteq A$ (skolemizing the negation of (1))

2. $f(A,A) \in A$ (from 1 and II.2)

3. $\neg f(A,A) \in A$ (from 1 and II.3)

4. $\square$ (from 3 and 2)

where clauses 2 and 3 are the result of *applying* the definition of $\subseteq$ in the context of clause 1. In this simple example, applying the definition produces contradiction immediately.

In fact the latter refutation could not be achieved by a linear resolution proof procedure.

The difficulties above can be better appreciated by considering the next example.

Exemplo 4.2 *Using linear resolution we can prove the theorem*

$$\forall x \forall y \forall z (x \subseteq y \wedge y \subseteq z \Rightarrow x \subseteq z) \tag{2}$$

One possible proof is:

1. $A \subseteq B$	*(skolemizing the negation of (2), 1st clause)*
2. $B \subseteq C$	*(skolemizing the negation of (2), 2nd clause)*
3. $\neg A \subseteq C$	*(skolemizing the negation of (2), 3rd clause)*
4. $f(A,C) \in A$	*(from 3 and II.2)*
5. $\neg A \subseteq y \vee f(A,C) \in y$	*(from 4 and II.1)*
6. $f(A,C) \in B$	*(from 5 and 1)*
7. $\neg B \subseteq y \vee f(A,C) \in y$	*(from 6 and III.1)*
8. $f(A,C) \in C$	*(from 7 and 2)*
9. $A \subseteq C$	*(from 8 and III.3)*
10. $\square$	*(from 9 and 3)*

From this (yet simple) example it is apparent that a proof procedure will have a good chance to get lost in a lot of fruitless deductions. And it can be seen that the cause of this is the failure to recognize that what matters is to apply the definition of $\subseteq$ to clauses 1–3. This is what was made (implicitly) in the refutation shown in the example, but in such a way that the application is not immediately apparent to a human being.

The natural manner to apply a definition is (remembering the eliminability requirement):

(a) to *translate* an instance of the defined concept in order to work in a inferior level of the definitional structure; or

(b) to derive an instance of the defined concept in order to work in a superior level of the definitional structure.

An alternative to (b) is to translate the other available concepts and to work in the actual level of the considered concept. Thus, the approach (a) is sufficient; in principle we could use the radical procedure of translating exaustively all available concepts and working in the primitive level.

In a proof procedure it is important that the above mentioned translation be made in a controlled way (by *necessity*). If this is the case, we will have solved our two problems:

- increased efficiency; and
- more natural (inteligible) deduction.

In [Carvalho 74] a translation system based on the theory of definitions is proposed. Deductions are essentialy linear deductions with provision for translation of defined symbols that appear on the theorem to be proved. The process of translation is exaustive in the sense that the system works (by linear resolution) on the primitive level.

A translation system for our fragment of set theory is shown bellow:

T.1 $\phi(t_1 = t_2) = \phi(t_1 \subseteq t_2) \cup \phi(t_2 \subseteq t_1)$

T.2 $\phi(\neg t_1 = t_2) = \phi(\neg t_1 \subseteq t_2) \vee \phi(\neg t_2 \subseteq t_1)$

T.3 $\phi(t_1 \subseteq t_2) = \phi(\neg z \in t_1) \vee \phi(z \in t_2)$

T.4 $\phi(\neg t_1 \subseteq t_2) = \phi(f(t_1, t_2) \in t_1) \cup \phi(\neg f(t_1, t_2) \in t_2)$

T.5 $\phi(t_3 \in (t_1 \cap t_2) = \phi(t_3 \in t_1) \cup \phi(t_3 \in t_2)$

T.6 $\phi(\neg t_3 \in (t_1 \cap t_2) = \phi(\neg t_3 \in t_1) \vee \phi(\neg t_3 \in t_2)$

T.7 $\phi(t_3 \in (t_1 \cap t_2) = \phi(t_3 \in t_1) \vee \phi(t_3 \in t_2)$

T.8 $\phi(\neg t_3 \in (t_1 \cup t_2) = \phi(\neg t_3 \in t_1) \cup \phi(\neg t_3 \in t_2)$

The process of translation is exaustive in the sense that the complete translation of the original query is obtained in terms of the primitive predicates ($\in$, in the example), the inferential system works (by linear resolution) on the primitive level. As an example we show bellow a possible proof of the formula(2) of example 1.2:

1. $A \subseteq B$	(skolemizing the negation of (eq2, 1st clause)
2. $B \subseteq C$	(skolemizing the negation of (eq2), 2nd clause)
3. $\neg A \subseteq C$	(skolemizing the negation of (eq2), 3rd clause)
4. $\neg z \in A \vee z \in B$	(translation of 1)

5. $\neg z \in B \vee z \in C$ (translation of 2)

6. $f(A,C) \in A$ (translation of 3, 1st clause)

7. $\neg f(A,C) \in C$ (translation of 3, 2nd clause)

8. $\neg f(A,C) \in B$ (from 7 and 5)

9. $\neg f(A,C) \in A$ (from 8 and 4)

10. $\square$ (from 9 and 6)

This example shows that even if we simply translate all defined symbols to the primitive level, efficience can be increased, as only the definitions effectively involved are used; other definitions are kept idle in the data base.

Possible extensions to this approach can be visualized as we consider the possibility of using the definitional structure to guide the translations in more complex situations. The definitional structure is not enough in most situations, and the complete translation may generate sets of clauses of non manageable size. For example if the sentence to be proven includes = the we may have an combinatorial explosion, as shown in the following example:

Example 4.1 *Let (a) be se sentence:* $\forall x \forall y \forall z (x \cap (y \cup z) = (x \cap y) \cup (x \cap z))$

Skolemizing the negation of (a):

$$\neg A \cap (B \cup C) = (A \cap B) \cup (A \cap C))$$

which can be translated to:

$$\neg A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)) \vee \neg (A \cap B) \cup (A \cap C)) \subseteq A \cap (B \cup C)$$

which can be translated to the following set of clauses:

2.1 $f(I,II) \in A \vee f(II,I) \in A$

2.2 $f(I,II) \in A \vee f(II,I) \in A \vee f(II,I) \in C$

2.3 $f(I,II) \in A \vee f(II,I) \in B \vee f(II,I) \in A$

2.4 $f(I,II) \in A \vee f(II,I) \in B \vee f(II,I) \in C$

2.5 $f(I,II) \in A \vee \neg f(II,I) \in A \neg \vee f(II,I) \in B$

2.6 $f(I,II) \in A \vee \neg f(II,I) \in A \neg \vee f(II,I) \in C$

2.7 $\neg f(I,II) \in A \neg f(I,II) \in B \vee f(II,I) \in A$

2.8 $\neg f(I,II) \in A \neg f(I,II) \in B \vee f(II,I) \in A \vee f(II,I) \in C$

2.9 $\neg f(I,II) \in A \neg f(I,II) \in B \vee f(II,I) \in B \vee f(II,I) \in A$

2.10 $\neg f(I,II) \in A \neg f(I,II) \in B \vee f(II,I) \in B \vee f(II,I) \in C$

2.11 $\neg f(I,II) \in A \neg f(I,II) \in B \vee \neg f(II,I) \in A \neg \vee f(II,I) \in B$

2.12 $\neg f(I,II) \in A \neg f(I,II) \in B \vee \neg f(II,I) \in A \neg \vee f(II,I) \in C$

2.13 $\neg f(I,II) \in A \neg f(I,II) \in C \vee f(II,I) \in A$

2.14 $\neg f(I,II) \in A \neg f(I,II) \in C \vee f(II,I) \in A \vee f(II,I) \in C$

2.15 $\neg f(I,II) \in A \neg f(I,II) \in C \vee f(II,I) \in B \vee f(II,I) \in A$

2.16 $\neg f(I,II) \in A \neg f(I,II) \in C \vee f(II,I) \in B \vee f(II,I) \in C$

2.17 $\neg f(I,II) \in A \neg f(I,II) \in C \vee \neg f(II,I) \in A \neg \vee f(II,I) \in B$

2.18 $\neg f(I,II) \in A \neg f(I,II) \in C \vee \neg f(II,I) \in A \neg \vee f(II,I) \in C$

2.19 $f(I,II) \in B \vee f(I,II) \in C \vee f(II,I) \in A$

2.20 $f(I,II) \in B \vee f(I,II) \in C \vee f(II,I) \in A \vee f(II,I) \in C$

2.21 $f(I,II) \in B \vee f(I,II) \in C \vee f(II,I) \in B \vee f(II,I) \in A$

2.22 $f(I,II) \in B \vee f(I,II) \in C \vee f(II,I) \in B \vee f(II,I) \in C$

2.23 $f(I,II) \in B \vee f(I,II) \in C \vee \neg f(II,I) \in A \neg \vee f(II,I) \in B$

2.24 $f(I,II) \in B \vee f(I,II) \in C \vee f(II,I) \in A \neg \vee f(II,I) \in C$

where

$$I = A \cap (B \cup C)$$
$$II = (A \cap B) \cup (A \cap C))$$

this example shows that the use of definitions is not enough, we have to control the combinatorial explosion. One way to do this by the use of Communication Predicates [Carvalho 74]. The use of communication predicates allows us to split a unsatisfiable set of clauses which is a disjunction of two sets of clauses with variables in common, in such way that we cut down the number of clauses generated by resolution. The result bellow [Carvalho 74], justify the introduction of communication predicates:

Teorema 4.1 *Let* $S = S_1 \vee S_2$ *be a set of clauses,* $x_1, x_2, \cdots, x_n$ *be the set of variables common to* S_1 *and* S_2*, and* P *a new n-ary predicate symbol.* S *is unsatisfiable if and only if* $S_1 \vee P(x_1, \cdots, x_n) \cup \neg P(x_1, \cdots, x_n) \vee S_2$ *is unsatisfiable.* □

Notice that if S_0 is a satisfiable set of clauses, for instance the proper axioms of a theory or definitions represented by D_+ and D_- we can split S_0 or the set of clauses obtained by the negation of the theorem to be proven and the above result prevails. In the proof procedure we have to be more careful in order to maintain refutation - completeness. In [Carvalho 74] a refutaiton-complete strategy called P-refutation was presented. The next example illustrates the use of this strategy.

4.1 Example

The translation system **T.1**,$\cdots$,**T.8** can me modified by substituting:

CT.1 $\phi(\neg t_1 = t_2) = \phi(\neg t_1 \subseteq t_2) \vee P(t_1, t_2)$

CT.2 $\phi(P(t_1, t_2) = \phi(\neg t_2 \subseteq t_1)$

by **T.2**, obtaining a different translation system. Notice that by the translation rule **CT.1** the translation stops in the predicate P.

Deductions, now, consist of translations followed by linear deductions up to the point where a clause containning only occurrences of the communication predicate P is found, and a new translation followed by linear deduction. So the last example can be worked out as follows:

Step I: Partial translation of the predicate $\subseteq$ to the level of the primitive $\in$:

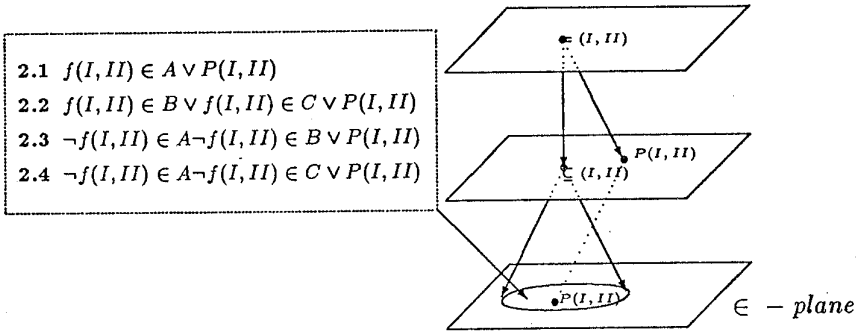

Figure 3: Partial reduction to the primitive level

Notice in the picture that the predicate P is originally in the same level or plane as $\subseteq$ and the dashed line means that the ground literal $P(I,II)$ was not translated to that plane but transfered without modifications.

Step II: Linear deduction

2.5 $\neg f(I,II) \in B \vee P(I,II)$ (from 2.1 and 2.3)

2.6 $f(I,II) \in C \vee P(I,II)$ (from 2.5 and 2.2)

2.7 $\neg f(I,II) \in A \vee P(I,II)$ (from 2.6 and 2.4)

2.8 $P(I,II)$ (from 2.7 and 2.1)

Step III: Translation of $P(I,II)$

2.1 $f(II,I) \in A$

2.2 $f(II,I) \in A \vee f(II,I) \in C$

2.3 $f(II,I) \in B \vee f(II,I) \in A$

2.4 $f(II,I) \in B \vee f(II,I) \in C$

2.5 $\neg f(II,I) \in A \neg \vee f(II,I) \in B$

2.6 $\neg f(II,I) \in A \neg \vee f(II,I) \in C$

Step IV: Linear deduction

2.7 $\neg f(II, I) \in B$ (from 2.1 and 2.5)

2.8 $f(II, I) \in C$ (from 2.7 and 2.4)

2.9 $\neg f(II, I) \in A$ (from 2.8 and 2.6)

2.10 $\square$ (from 2.9 and 2.1)

Note: The result obtained in **Step III** is, in general, a disjunction of positive literals in the communication predicate P. After each translation of one of these literals and subsequent linear deduction, we obtain a shorter disjunction obtained from the previous one by substitution. $\square$

The use of communication predicates in a definitional theory, like the one shown in this example, changes the structure of the original theory. This modification, in most situations [Carvalho 74], improves the overall efficiency of a deductive system.

5 Functions and Synonymy

In this section we will propose a way to treat functions in certain special situations. Again we will use examples to expose our ideas.

The examples will be entirely based on the following definition of the relation "brother":

$$\forall x \forall y (Brother(x, y) \Leftrightarrow \exists (z = parent(x) \wedge z = parent(y)) \tag{3}$$

In clause form:

I $\neg Brother(x, y) \vee f(x, y) = parent(x)$

II $\neg Brother(x, y) \vee f(x, y) = parent(y)$

III $Brother(x, y) \vee \neg z = parent(x) \vee \neg z = parent(y)$

Let us begin with a simple example.

Exemplo 5.1 *The formula that assures the symmetry of brother:*

$$\forall x \forall y (Brother(x, y) \Rightarrow Brother(y, x)) \tag{4}$$

can be proved by means of the following refutation:

1. $Brother(A, B)$ *(skolemizing the negation of(4), 1st clause)*

2. $\neg Brother(B, A)$ *(skolemizing the negation of (4), 2nd clause)*

3. $f(A, B) = parent(A)$ *(translation of 1, 1st clause)*

4. $f(A, B) = parent(B)$ *(translation of 1, 2nd clause)*

5. $\neg z = parent(A) \vee \neg z = parent(B)$ *(translation of 2)*

6. $\neg f(A, B) = parent(B)$ *(from 5 and 3)*

7. $\square$ *(from 6 and 4)*

As we can see, the use of the logical "=" in example 5.1 is *redundant* in the sense that the *functionaty condition* implicit in functions is not required to process formula (4).

Let us write (3) as

$$\forall x \forall y (Brother(x, y) \Leftrightarrow \exists z (Parent(z, x) \wedge Parent(z, y)) \tag{5}$$

and let us exhibit the functionality condition of Parent:

$$\forall x \exists z (Parent(z, x) \wedge \forall y (Parent(y, x) \Rightarrow y = z)) \tag{6}$$

Writing formula (5) in clause form:

I $\neg Brother(x, y) \vee Parent(f(x, y), x)$

II $\neg Brother(x, y) \vee Parent(f(x, y), y)$

III $Brother(x, y) \vee \neg Parent(z, x) \vee \neg Parent(z, y)$

Then we have the following proof of (4):

1. $Brother(A, B)$ (skolemizing the negation of (4), 1st clause)

2. $\neg Brother(B, A)$ (skolemizing the negation of (4), 2nd clause)

3. $Parent(f(A, B), A)$ (translation of 1, 1st clause)

4. $Parent(f(A, B), B)$ (translation of 1, 2nd clause)

5. $\neg Parent(z, A) \vee \neg Parent(z, B)$ (translation of 2)

6. $\neg Parent(f(A, B), B)$ (from 5 and 3)

7. $\square$ (from 6 and 4)

And the functionality condition was not necessary for this specific refutation.

How could we manage functional relations such that the functionality condition is applied only when required? This is the question we will approach next. We emphasize that we will not try to preserve completeness, but we will respect completeness as long as we can (provided that efficiency be considered). Our intent is to control the combinatorics involved when using equality in the context ilustrated above.

The next example involves the use of the functionality condition of Parent. The clauses for this condition (formula (6)) are:

(f1) $Parent(g(x), x)$

(f2) $\neg Parent(y, x) \vee y = g(x)$

Exemplo 5.2 *The formula that assures the transitivity of brother:*

$$\forall x \forall y \forall z (Brother(x, y) \wedge Brother(y, z) \Rightarrow Brother(x, z)) \qquad (7)$$

can be proved by means of the following refutation:

1. $Brother(A, B)$ *(skolemizing the negation of (7), 1st clause)*

2. $Brother(B, C)$ *(skolemizing the negation of (7), 2nd clause)*

3. $\neg Brother(A, C)$ *(skolemizing the negation of (7), 3rd clause)*

4. $Parent(f(A, B), A)$ *(translation of 1, 1st clause)*

5. $Parent(f(A, B), B)$ *(translation of 1, 2nd clause)*

6. $Parent(f(B, C), B)$ *(translation of 2, 1st clause)*

7. $Parent(f(B, C), C)$ *(translation of 2, 2nd clause)*

8. $\neg Parent(z, A) \vee \neg Parent(z, C)$ *(translation of 3)*

9. $\neg Parent(f(A, B), C)$ *(from 8 and 4)*

10. $\neg Parent(g(x), C) \vee \neg Parent(f(A, B), x)$ *(from 9 and (f2))*

11. $\neg Parent(g(B), C)$ *(from 10 and 5)*

12. $\neg Parent(y, C) \vee \neg Parent(y, B)$ *(from 11 and (f2))*

13. $\neg Parent(f(B, C), B)$ *(from 12 and 7)*

14. $\square$ *(from 13 and 6)*

Clauses 10 and 12 were derived by applying the *rule of paramodulation* [Robinson 69], a rule whose applications are difficult to control.

Analysing example 5.2 we note by looking at clauses 7 and 9 that to establish the theorem it is sufficient to prove that

$$f(B,C) = f(A,B).$$

But we know that Parent is functional in the first argument, and so clauses 5 and 6 (with the functionality condition) must do the work. Then we could produce, instead of clause 10:

10'. $\neg f(B,C) = f(A,B)$ (from 9 and 7)

We know that clause (f2) is the clause that must be used to establish the equality. But things become complicated by the fact that the equality predicate in clause (f2) makes reference to another Skolem function. What we can do is to produce:

11'. $\neg Parent(F(B,C),x) \vee \neg g(x) = f(A,B)$ (from 10' and (f2))

using the rule of paramodulation. But the above equality can be established only by using clause (f2):

12'. $\neg Parent(F(B,C),x) \vee \neg Parent(F(A,B),x)$ (from 11' and (f2))

13'. $\neg Parent(f(A,B),B)$ (from 12' and 6)

14'. $\square$ (from 13' and 5)

One interesting thing to note is that the later deduction don't depends on axiom (f1) (the *existence condition*) but only on axiom (f2) (the *unicity condition*). But the Skolem function denoted by g was introduced just to cope with the existence condition. And, as g appears also in the axiom that expresses the unicity condition, to use this axiom, we will have to deal with g. This is accomplished by clause 11'. This clause in a certain sense is a "transition clause" that says us that the equality of f(B,C) and f(A,B) depends on the equality of these and g(x) for some x (in the example, B is used).

A more efficient thing to do in such circunstances is to use the following formula logicaly equivalente to formula (6), that separates completely the unicity condition from the existence condition:

$$\forall x \exists z(Parent(z,x)) \wedge \forall x \forall y \forall z(Parent(z,x) \wedge Parent(y,x) \Rightarrow z = y) \quad (8)$$

In clause form we have:

(ec) $Parent(g(x),x)$

(uc) $\neg Parent(z,x) \vee \neg Parent(y,x) \vee z = y$

And so, a proof procedure will obtain one less clause and will be much more restricted in terms of choices (we preserve the numbering above, but now with double apostrophes):

10". $\neg f(B,C) = f(A,B)$ (from 9 and 7)

12". $\neg Parent(f(B,C),x) \vee \neg Parent(f(A,B),x)$ (from 10" and (uc))

13". $\neg Parent(f(A,B),B)$ (from 12" and 6)

14". $\square$ (from 13" and 5)

Then, if we wish to avoid the use of (logical) equality, but we have to use functional relations in special situations like that shown above, a good thing to do is to let the proof procedure *know* the functionality and use the unicity condition as exemplified. Such proof procedure will not worry about interpreting the equality symbol appearing, for instance, in clause 10", as "real equality". That clause would be derived in a controlled behaviour, because the proof procedure will know about the functionality of Parent and know about a unicity clause to deal with it. From that behaviour, we could regard the process of derivation of clauses 10" and 12" as a single pass.

Another improvement can be obtained by an automatic management of synonyms. Clauses 4 to 7 in example 5.2:

4. $Parent(f(A,B),A)$ (translation of 1, 1st clause)

5. $Parent(f(A,B),B)$ (translation of 1, 2nd clause)

6. $Parent(f(B,C),B)$ (translation of 2, 1st clause)

7. $Parent(f(B,C),C)$ (translation of 2, 2nd clause)

imply, with the help of the unicity condition associated to *Parent*, that all its first arguments are identical. In a functional notation we would have:

$$\begin{aligned} parent(A) &= f(A,B) \\ parent(B) &= f(A,B) \\ parent(B) &= f(B,C) \\ parent(C) &= f(B,C) \end{aligned}$$

Using "$parent(c_1)$" as a canonical representative for c_2 in "$Parent(c_2,c1)$"we could collect all synonyms of $parent(c_1)$ in a list, and use this list to solve the negative literals referencing the predicate symbol *Parent*. Figure 4 shows how this works for our example. In the first column are presented the clauses in order of generation, and in the second column is presented the evolution of the synonymy.

The next clause generated in example 5.2 is clause 8:

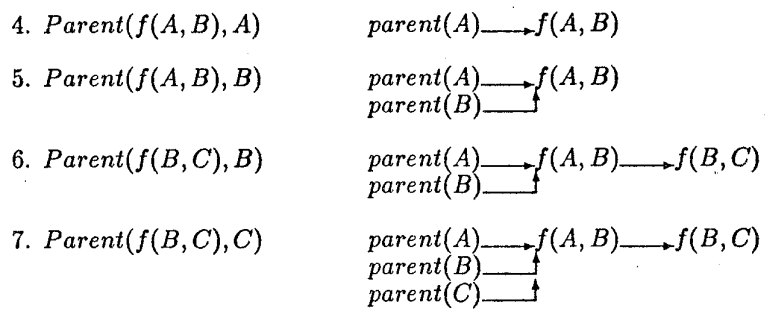

Figure 4: Lists of synonyms.

8. $\neg Parent(z, A) \vee \neg Parent(z, C)$ (translation of 3)

and then z can be substituted by $parent(A)$, producing:

9. $\neg Parent(parent(A), C)$ (from 8)

an the empty clause is imediatety produced by consulting the synonymy.

In general, if a clause $Q(t_1, \ldots, t_n, u)$ is entered in the data base (as an input clause or as a resolvent), where Q is functional in the last argument and $t_1, \ldots, t_n, u$ are terms, then u and $q(t_1, \ldots, t_n)$ are put in the same list (variables in these terms are standardized apart from other terms). And, if a clause $\neg Q(t_1, \ldots, t_n, u) \vee \psi$ is generated, then if u is a variable we apply the substitution $\{q(t_1, \ldots, t_n)/u\}$ to ψ to obtain the next clause, and if u is not a variable then the next clause will be $\psi\ \sigma$, where σ is the unifier of $q(t_1, \ldots, t_n)$ and u with terms in the same list (obviously, if ψ is empty the result is the empty clause).

A more sophisticated approach remains to be worked when we consider non-unit clauses with a (positive) literal of the form $Q(t_1, \ldots, t_n, u)$.

6 Conclusions

In this paper we were concerned with two subjects which are very important from the standpoint of knowledge processing systems: efficiency and naturality of deductions.

It is interesting to note that the special treatement of definitions suggested in section 4 and the special treatment of functional relations suggested in section 5 mimics, in a certain sense, what a human being would probably do in similar situations. And, in these cases, naturality contributes to efficiency. This is

very important, particularly, because proofs by resolution are not very clear and linear resolution is not very efficient in many situations.

In section 4 we have shown that even if we translate all defined symbols to the primitive level, efficiency can be increased. It remains to be worked the use of the definitional structure to guide the translations in more complex situations, particularly when (possibly many) creative axioms are stated with the help of defined symbols.

In section 5 we have seen how to construct lists of synonyms so as the proof procedure can "remember" of identical terms in the process of constructing a proof. It remais to be worked the processing of non-unit clauses involving positive literals about functional relations.

7 References

[**Carvalho 74** Carvalho, R.L. *Some Results in Automatic Theorem-Proving with Applications in Elementar Set Theory and Topology* Ph.D. Thesis Dept. of Computer Sciences University of Toronto, 1974.

[**Green 69** Green, C.C. *The Application of Theorem Proving in Question-Answering Systems* Ph.D. Thesis, Stanford University, Stanford California, USA 1969.

[**Kowalsky 71** Kowalski, R. Kuehner, D. Linear Resolution with Selection Functions *Artificial Intelligence 2* 1971 227–260.

[**Loveland 67** Loveland, D.W. Linear Format for Resolution. *Proc. of the IRIA Symposium of Automatic Demonstration.* Versailles France, 1968, 147–162.

[**Loveland 69** Loveland, D.W. Simplified Format for the Model-Elimination Theorem-Proving Procedure *Journal of The ACM, 16,* 1969, 349–363.

[**Luckham 68** Luckham, D. Refinement Theorem in Resolution Theory. *Proc. of the IRIA Symposium on Automatic Demonstration* Versailles, France, 1968 163–190.

[**Robinson 65** Robinson, J.A. A Machine-Oriented Logic Based on the Resolution Principle *Journal of the ACM. 12*(1), 1965 23–41.

[**Vieira 87**] Vieira, N.J. *Máquinas de Inferência para Sistemas Baseados em Conhecimento*, Tese de Doutorado, PUC/RJ, Rio de Janeiro, 1987.