ICOT Technical Report: TR-437

TR-437

WEIGHTED-GRAPHS— Tool for Studying the Halting Problem and Time Complexity in Term Rewriting Systems and Logic Programming

by
P. Devienne

November, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

WEIGHTED GRAPHS
a Tool for Studying the Halting Problem and Time Complexity in Term Rewriting Systems and Logic Programming

Philippe DEVIENNE*
Institute for New Generation Computer Technology, Tokyo, Japan
Universite de Lille - LIFL , Lille, France†

November 21, 1988

Abstract

This study is based on the halting and complexity problems for a simple class of logic programs in PROLOG-like languages. Any Prolog program can be expressed in the form of an overlap of some simpler programs whose structures are basic and can be studied formally. The simplest recursive rules are studied here and the weighted graph is introduced to characterise their behaviour.

This new syntactic object, the *weighted graph*, generalises the directed graph. Unfoldings of directed graphs generate infinite regular trees that I generalise by weighting the arrows and putting periods on the variables. The weights along a branch are added during unfolding and the result (modulo of the period) indexes variables. Hence, their interpretations are non-regular trees because of the infinity of variables. This paper presents some of the formal properties of these graphs, finite and infinite interpretation and unification.

Although they have a consistency apart from all possible applications, weighted graphs characterise the behaviour of recursive rules in the form $L :- R$. They express the most general fixpoint of these rules and range across a finite sequence of recursive rewritings. Within global rewriting systems, narrowing and logic programming, the halting problem and the existence of solutions are proved to be decidable for this simple recursive rule with linear goals and facts, and the complexity is shown to be at most linear.

Although these problems are undecidable for slightly more complex schemes, it is hoped that from the weighted graphs of each recursive sub-structure of a Prolog program, the whole behaviour of the program will be understandable. Then, the weighted graphs would be the nucleus of an efficient and methodological logic programming, which could be called, *Structured Logic Programming*.

1 Introduction

Estimating the termination and complexity of a program from its structure is not an original idea. Although [Böhm and Jacopini 66] proved that all programming can be done with at most one while loop, usually the structure of a well-written imperative program gives good behavioural properties.

Within logic programming, this structural approach has not given comparable results; however, this approach is more coherent because the language nucleus is based on one and only

*Supported by INRIA grant
†On leave during 1988

one operation, called inference or rewriting, but it also seems to be more complex because this operation is very powerful [Dauchet 1987].

The question of termination has been studied in different contexts, namely, term rewriting systems, narrowing and Horn clauses with or without function symbols; they share the same basic operation, rewriting. However, within Horn clauses, the term *global rewriting* must be used because the whole term, not a part of it, may be rewritten.

Because termination is in general an undecidable property [Huet and Lankford 78], many works have been devoted to introducing methods for proving that particular systems or programs are terminating or non-terminating, but even in this case, in spite of their simple appearance, they are complex and show the expressive power of rewriting. Let us look at some of them:

- Term rewriting systems, more general than our context: A set of rules, R, terminates iff, for any ground term, T, no infinite derivations are possible. [Lipton and Snyder 77] assert that three rules suffice for undecidability. [Dershowitz 85 and 87] give the same result for two rules. Finally, [Dauchet 87] proves that it is possible with only one rule to simulate any Turing machine. Thus, the uniform termination of one rule is undecidable, too.

- Horn clauses without function symbols, more particular than our context: The structural approach has been studied; for example, if it is possible to eliminate recursion from a program, then it is said to be bounded. This property is undecidable even for linear programs (that is, each rule contains at most one occurrence of a recursive predicate) [Gaifman and Mairson 87]. This property is decidable for linear programs with a single rule if the intensional predicate is binary and its complexity is shown to be NP-complete [Vardi 88].

- Horn clauses, context studied in this paper: This termination problem is often asked about a set of rules, R, and one term, T; R and T terminate iff no infinite derivations of T are possible.

However, even in the case of a single rule, there is no good tool for checking termination and for understanding the basic recursivity. If good intuition is possible about simple rules such as the following :

1. friend(X,Y) :- friend(Y,X). (infinite)
   X is a friend of Y if Y is a friend of X

2. put(milk) :- put(coffee). (finite)
   I prefer white coffee

3. integer(succ(X)) :- integer(X). (depending on goals)
   succ(X) is an integer if X is an integer

unfortunately, the non-linearity of the terms, the existence of some variables on one side of the clause, and the permutation of variables during rewriting generally make intuitive comprehension of behaviour impossible.

Weighted graphs have been introduced [Devienne and Lebegue 86] as an attempt to give a first answer element. They generalise the notion of infinite trees [Courcelle 83].

Directed graphs are well known: their unfoldings generate infinite rational trees. Informally, a weighted graph is a graph with a top, nodes and arrows, but the arrows are weighted by relative integers, and the variables may have a period. During the unfolding, the weights along a branch are added and their sum (modulo of the period) indexes the variable. These unfolded trees are non-rational because of their infinity of variables; their formal properties are studied. The most important properties, unfolding in a counter range and unification with occur-check, are presented here. The whole formal presentation is a generalisation of definition, interpretation and the unification algorithm of directed graphs.

Although they have a consistency within the algebraic theory apart from the halting and complexity problem, weighted graphs express as clearly as possible the behaviour of rules, denoted $L \rightarrow R$ within term rewriting systems or L :- R in logic programming.

Any loop-generating rule, L :- R, has a computable weighted graph whose interpretation is its most general fixpoint and whose finite interpretation is a range of the most general sequence of finite inferences using this rule. This means that all the information about its behaviour is concentrated in its weighted graph.

2 Why Yet Another Syntactic Object?

The basic syntactic objects used in term rewriting systems and in logic programming are trees, directed acyclic graphs (dags) and directed (or oriented) graphs.

In Prolog, any literal of a rule is a finite tree, that is, the basic object in a Prolog program. In a tree, each different node of the root has one and only one father node:

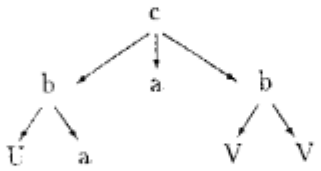


Fig. 1

In term rewriting systems, the term which has to be rewritten is *ground*, that is, without variables. The tree (fig. 1) is not ground because of variables U and V. A tree is said to be *linear* if there is only one occurrence of its variables. The tree (fig. 1) is also not linear because of two occurrences of variable V.

Directed acyclic graphs (dags) were introduced for improving the memory size and the unification algorithm. In this new structure, a node may be shared by several father nodes, and it is possible to suppose that there are not two different nodes labelled by the same variable. Hence, we need not search all the occurrences of a variable to apply a substitution during the unification. Thus, the form is a graph whose arrows are directed, but this graph has no cycle. By unfolding, they characterise the finite trees.

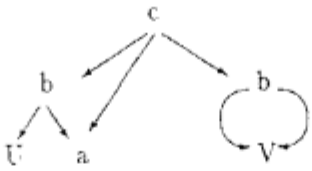


Fig. 2

Another generalisation exists in the form of a directed graph which may contain cycles [Courcelle 83] and [Fages]. These unfolded graphs are regular trees, that is, solutions of a finite system of equations, or composed of a finite set of sub-trees. The interest is to unify without occur-check (as in Prolog II). For example, the unifier of A and f(A) does not exist with occur-check. However, if occur-check is omitted, then A V f(A) is represented by the directed graph:

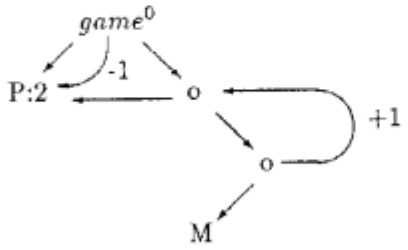
$\xrightarrow{\text{Unfolding}}$ $f(f(f(\cdots$ (infinite tree)

Unfortunately, these syntactic objects are too poor to study the behaviour of a recursive rule. Using them, the results are only partial and their proofs are complex. Their structures are not adapted to allow a good comprehension of the basic recursivity. This is why we are obliged to generalise them and to introduce weighted graphs.

2.1 What is a weighted graph: Informal presentation

A *weighted graph* is a directed graph where:

1. the root is weighted by a relative integer
2. the arrows are weighted by relative integers
3. the variables may be periodic.



This looks like a directed graph:
- there are six nodes labelled by
function symbols (game, o) or variables (P, M),
the arrows are directed, and this graph contains loops;
- however, the root (game) is weighted by 0,
two arrows are weighted by -1 and +1,
and variable P has period 2.

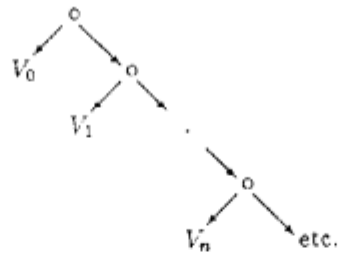
Let us look at two examples to understand the meaning of these new notions, named weight and period.

2.1.1 Weight

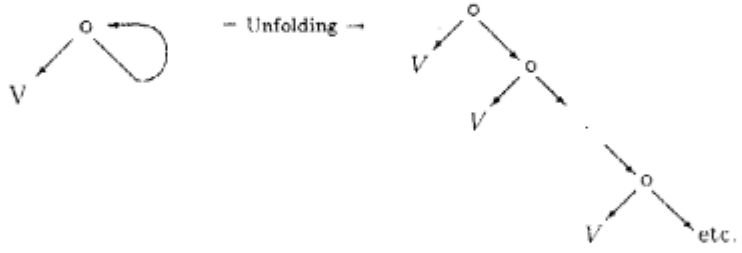
The most general infinite list can be expressed in the following form:

$$[\, V_0 \,,\, V_1 \,,\, \ldots \,,\, V_n \,,\, \ldots \quad \text{where } V_i \text{ are variables}$$

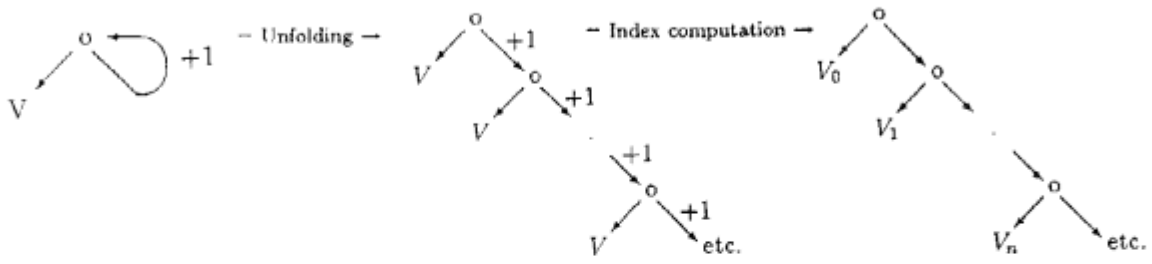
or in tree form:



Let us consider rule r: [V, U] → U. This list is its most general fixpoint.
The directed graph cannot be used to express the list because of the infinite number of variables:



However, if a weight, 1, is put on the looping arrow, the index, i, of the variables, V_i, can be computed as the sum of the weights along the branch from the root to the leaf:

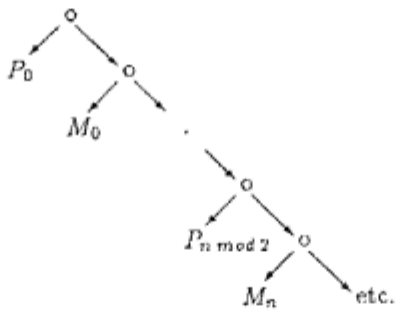


2.1.2 Period

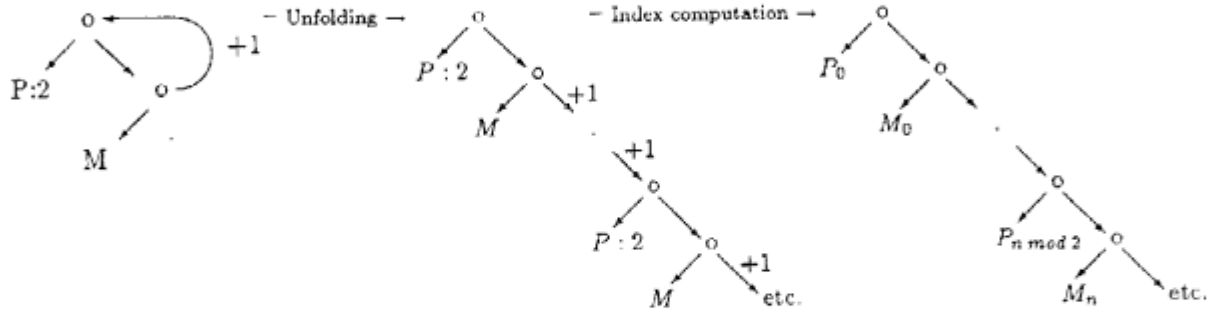
Let us consider a chess game between two players, P_0 and P_1, who play any move, M_i, in turn. The chess game can be expressed in list form:

$$[\, P_0\,,\ M_0\,,\ P_1\,,\ M_1\,,\ P_0\,,\ M_2\,,\ \ldots\,,\ P_{n \bmod 2}\,,\ M_n\,,\ \ldots$$

Player P_0 plays move M_0, then P_1 plays M_1, and P_0 plays M_2, and so on. Move M_n is, therefore, played by player $P_{n \bmod 2}$.
In equivalent tree form, that is:



For describing the periodicity of variable P, a period is put on this variable. In this way, the index is the sum of the weights modulo the period:

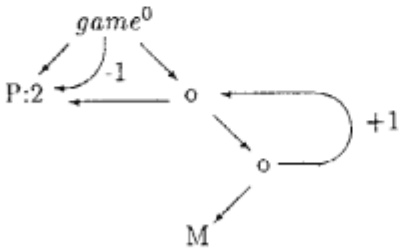


2.2 Definition of a weighted graph

Definition 1 *A weighted graph is a graph in the form*

$$wg \;=\; (X, Lab, Succ, \underline{Period}) \,/\, (Root, \underline{w_R})$$

where - *X is a set of nodes*
- *Lab is the label function from X to F ∪ Var*
F is the alphabet and Var is the set of variables
- *Succ is the successor function from X × N to X × Z*
Succ(x, i) = (x_i, w_i) means that x_i is the i^th successor of x
and this arrow is weighted by w_i.
- *Period is a function from Var to N*
Period(V) = p means that p is the period of variable V:
∀ k ∈ E (interval of interpretation), V_k = V_{k mod p}.
- *(Root, W_R) is an element of X × Z, that is, the weighted root.*



- $X = \{x_1, x_2, x_3, x_4, x_5\}$
- $Lab(x_1) = game$, $Lab(x_2) = P$, $Lab(x_3) = o$
  $Lab(x_4) = o$, $Lab(x_5) = M$
- $Succ(x_1, 1) = (x_2, 0)$, $Succ(x_1, 2) = (x_2, -1)$
  $Succ(x_1, 3) = (x_3, 0)$, $Succ(x_3, 1) = (x_2, 0) \ldots$
- $Period(P) = 2$
- $Root = x_1$ and $W_R = 0$

Remark 1 *Directed graph and weighted graph*

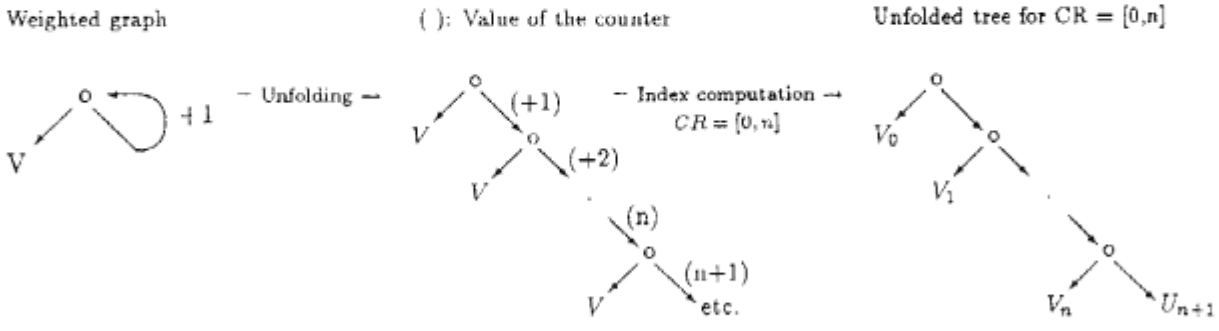
1. *The directed graph definition is the same, but the underlined parts have been added.*
2. *Another definition is possible: periods on the nodes [Devienne 87]. The advantage is unification without occur-check, but the major disadvantage is that interpretation is much more complex.*

2.3 Interpretation of a weighted graph

In the intuitive presentation, the weighted graph was unfolded without control, but for a good and powerful interpretation, two new notions must be introduced.

2.3.1 Counter range (CR)

For computing the index of variables, the sum of the weights along the branch is calculated using a counter. During unfolding, any weight found on the branch is added to the counter. Using the counter, a control can be defined. While the value of the counter belongs to the interval, CR, the unfolding is applied. As soon as this value becomes invalid, the unfolding of this branch stops. This is why a special variable must be associated with each node.
Let us look at an example with CR = [0,n]:

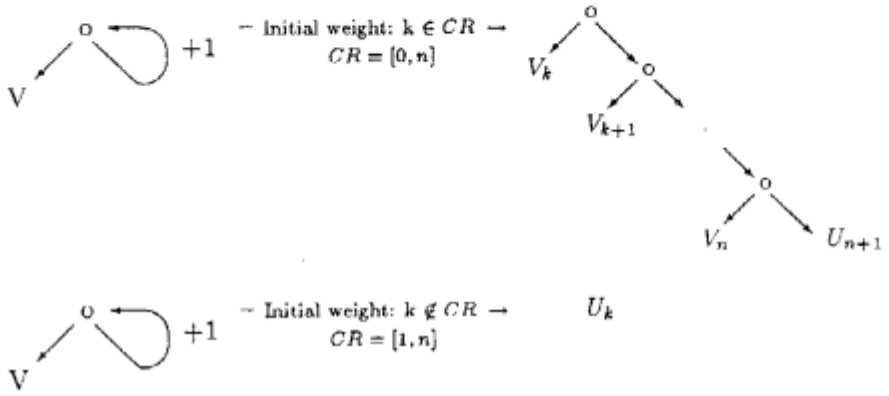


A special variable, U, is associated with node o. After n unfoldings of the weighted loop, the value of the counter becomes (n+1), so unfolding stops and the leaf of this branch is labelled by U_{n+1}, that is, the variable of node o, indexed by the value of the counter. The obtained tree expresses the most general list composed of more than n elements.

If $CR = Z$, unfolding is applied without control, and the interpretation is the most general infinite list.

2.3.2 Initial weights (IW)

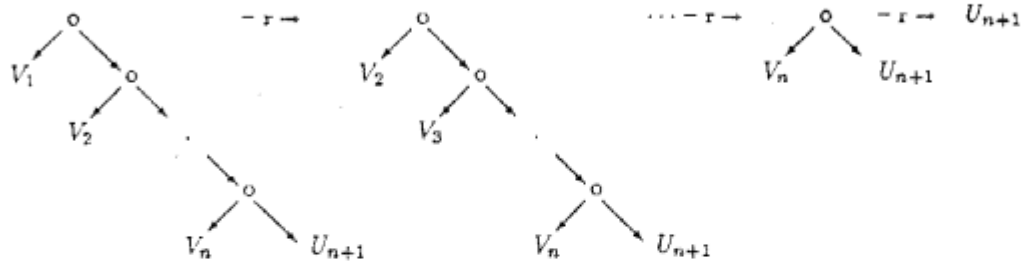
In addition to the weights of the root and the arrows, an initial weight is used. This is equivalent to considering that the counter contains an initial value.



In this way, the most general sequence of n rewrites using rule r: $[V, U] \rightarrow U$:

$$t_1 - r \rightarrow t_2 - r \rightarrow \cdots \rightarrow t_{n+1}$$

can be expressed from the previous weighted graph. The i^{th} term of this sequence is the interpretation of wg with the initial weight, i, and CR = [1,n]:



Definition 2 *The paths in a weighted graph are defined through the following recursive function, called Descendant. Let m be a path in the form* $i_1.i_2 \ldots i_k \in N^k$:

$Desc_{CR}((x, w), \varepsilon) = (x, w)$ (ε: *empty path*)

$Desc_{CR}((x, w), i_1.i_2 \ldots i_k) = Desc_{CR}((x_{i1}, w_{i1} + w), i_2 \ldots i_k)$ *if* $w \in CR$ *and* $Succ(x, i_1) = (x_{i1}, w_{i1})$

Definition 3 *Unfolding of a weighted graph wg for a counter range, CR, and an input weight, k.*

Let $(Root, w_R)$ *be the weighted root of* wg*, then the unfolded graph denoted* $U^k_{CR}(wg)$ *is:*
$\forall m$ *such that* $Desc((Root, w_R + k), m) = (x, w)$

$$\begin{array}{lll} U^k_{CR}(wg)(m) & = f & \text{if } w \in CR \text{ and } Lab(x) = f \in F \text{ (function symbol)} \\ & = V_{w \bmod period(V)} & \text{if } w \in CR \text{ and } Lab(x) = V \in Var \\ & = Vx_w & \text{if } w \notin CR \text{ and } Vx \text{ is the special variable of } x. \end{array}$$

Notes: The definition of the *modulo* function and the *highest common factor* is extended to Z to simplify this presentation:

1. The *modulo* function from Z to N
   $\forall w \in Z, w \bmod 0 = w$ means that a variable has at least period 0.
   The modulo function is an equivalence relation, where any element of an equivalence class can be chosen as a canonic element. This canonic element will be taken in the interval, CR, for example, the smallest positive element if there is one, otherwise the greatest negative one.
2. The *highest common factor* function from $Z \times Z$ to N
   $\forall w, w' \in Z,\ hcf(w, w') = hcf(\mid w \mid, \mid w' \mid)$ *and* $hcf(w, 0) = hcf(0, w) = \mid w \mid$.

Theorem 1 *The weighted graph is a generalisation of the directed graphs and it can express a subset of non-regular trees:*

$$Finite\ trees \equiv Dags \subset Directed\ graphs \subset Weighted\ graphs \subset Trees$$

Proof Any directed graph is a weighted graph whose weights and periods are null, or a weighted graph with any weights and whose period of all the variables is 1.

The weighted graph can express some non-regular trees because of the infinity of variables. Therefore, they cannot express the non-regular ground trees, for example, the infinite list of natural integers.

Definition 4 *Interpretation of a weighted graph for an input weight interval*
The interpretation of a weighted graph for a counter interval, CR, and an input weight interval, IW, is a set of pairs in the form (input weight, unfolded from this input weight):

$$I^{IW}_{CR}(wg) \ = \ \{ (k\ ,\ U^k_{CR}(wg)) \ / \forall k \in IW \}$$

Example $I^{[1,n+1]}_{[1,n]}(wg)$ expresses the most general sequence of n rewrites using $[V, U] \rightarrow U$ (cf 2.3.2).

2.3.3 Path and loop in a weighted graph

Definition 5 *Loop and basic loop*
A loop is a path from a node to itself using the descendant function. This loop is said to be basic if all the nodes, except for the first and the last, are different.

Generally, there is an infinity of loops, but always a finity of basic loops.

Definition 6 *Weight and sign of a path*

1. *The weight of a path is the sum of the arrow weights along it.*
2. *The path is said to be positive (resp. negative, null) if its weight is positive (resp. negative, null).*

Proposition 1 *A weighted graph contains no null finite loop iff all the basic loops from the same node have the same sign and are not null.*

Proof It is obvious that these conditions are necessary. Let us show that they are sufficient.
For any node, y, appearing in a loop from node x, there exists a basic loop from x through y. This basic loop is also a basic loop from y through x.
Hence, all the basic loops from a node appearing in any loop from x have the same sign. Moreover, the weight of a loop is the sum of some overlapped basic loop weights which are either all greater than zero or all less than zero. This means that any loop weight is never null.

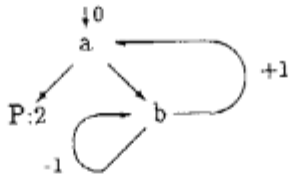
Corollary 1 *If a weighted graph contains no null finite loops, any looping node can be said to be either positive or negative because of the same sign of its loops.*

2.3.4 Finite weighted graphs

Definition 7 *A weighted graph is said to be finite if it contains no null finite loops, that is, iff it contains no null basic loops and there are no positive and negative basic loops from the same node.*

Therefore, it is easy to define an algorithm for checking whether a weighted graph is finite because of the finite number of basic loops. This property corresponds to the occur-check in unification.

The weighted graph in Fig. 1 *is finite, but the following weighted graph is not finite because of node b:*



Theorem 2 *The unfolding is finite in all finite counter ranges iff the weighted graph is finite:*

$$\forall k\ ,\ \forall \text{ finite } CR,\ U^k_{CR}(wg) \text{ is a finite tree} \quad \Leftrightarrow \quad wg \text{ is a finite weighted graph.}$$

The depth of the unfolding of a finite weighted graph is bounded by a linear function of the size of the counter range.

Lemma 1 *For a finite interval, CR, the unfolded result of a finite weighted graph is a finite tree whose height is bounded by a linear function of the size of CR.*

Proof Let k be the input weight and w_R the root weight. A path in the weighted graph is a path from the root to a function node in the unfolded graph iff any left subpath of this path has a weight, w_p, such that $k + w_R + w_p \in CR$.
However, CR, is a finite interval and w_p belongs to a finite interval.
Moreover, any path in the weighted graph can be expressed in the form:

$$m_1.loop(x_1).m_2.loop(x_2)\ldots m_k.loop(x_k).m_{k+1}$$

where the paths, m_i, contain no loops and the nodes, x_i, are all different.
All the parts of this path have a limited length:

1. $k \leq Card(X)$, that is, the number of nodes in the weighted graph
2. $\| m_i \| \leq Card(X)$
3. The length of each $loop(x_i)$ is limited because all the basic loop weights have the same sign. Therefore, the length is less than $Card(X) \times Card(CR)$ because the length of any basic loop is less than the number of nodes of the weighted graph and the weight of any basic loop is at least 1 or -1. This means that the length of each loop is bounded by a linear function of the size of CR.

Lemma 2 *If a weighted graph, wg, is not finite, there exists a constant, M, which is computable such that:*

$$Card(CR) \geq M \quad \Rightarrow \quad \exists k \in CR,\ U^k_{CR}(wg) \text{ is an infinite tree.}$$

Proof If the weighted graph is not finite, there exists a finite path, m, containing a loop whose weight is null:

$$\exists m \text{ path such that } m = m_1.loop(x) \text{ and } w_{loop(x)} = 0.$$

Let us define $[w_{min}, w_{max}]$ as the smallest interval which contains zero and the weights of every left subpath of one path, m. It is then easy to see that all the following paths have the same weight:

$$\forall n \in N\ ,\ \ m_1.loop(x)^n$$

and the weights of their left subpaths belong to $[w_{min}, w_{max}]$.
Thus, if interval CR contains more than $Card(w_{min}, w_{max}) + \mid w_R \mid$ elements, from the input weight, $(inf(CR) - w_{min})$ (if $w_R \geq 0$), or, $(sup(CR) - w_{max})$ (if $w_R \leq 0$), the unfolding is infinite because of the paths, $m_1.loop(x)^n$.

2.4 Partial order

Notation 1 *Let σ be a substitution.*

1. *$Domain(\sigma)$ is the set of variables which are substituted by σ.*
2. *$Range(\sigma)$ is the set of variables which appears in the trees associated with the variables of $domain(\sigma)$.*

$$domain(\sigma) = \{V_i \ / \ \exists (V_i \leftarrow t_i) \in \sigma \ \} \quad and \quad range(\sigma) = \bigcup_{(V_i \leftarrow t_i \ \in \sigma)} range(t_i)$$

3. *A substitution can be applied to a term by simultaneously replacing all occurrences of each variable of the domain. The term obtained is denoted $\sigma(t)$.*

Definition 8 *Let wg and wg' be two weighted graphs, then the following partial order is defined by:*

$$I^{IW}_{CR}(wg) \ \leq \ I^{IW'}_{CR'}(wg') \ \ if \ \ \exists \sigma, \text{ a substitution such that } \ \sigma(\ I^{IW}_{CR}(wg)\ ) \ \subset \ I^{IW'}_{CR'}(wg')$$

That is, if:

1. $IW \ \subset \ IW'$

2. *there exists a substitution making the unfolded tree of wg equal to the unfolded tree of wg' for every input weight of IW.*

Proposition 2 *Let WG be a weighted graph structure, that is, $WG = (X, Lab, Succ, Period)$ and let us define IW, IW', CR and CR' as four intervals of Z such that $IW \subset IW'$ and $CR \subset CR'$.*
There is a substitution, σ, which verifies:

$$\forall (Root, w_R) \in X \times Z,\ wg \text{ denotes } WG/(Root, w_R)\ ,\ \sigma(\ I^{IW}_{CR}(wg)\ )\ \subset\ I^{IW'}_{CR'}(wg)$$

Proof For any value of the counter belonging to CR'-CR, the interpretation of the nodes is different. Let x be a node and Vx its special variable; for a value, c, of the counter, this node will be interpreted during the unfolding as:

1. Vx_c from CR
2. $U^c_{CR}(WG/(x,0))$ from CR'

This is different, iff value c belongs to CR', but is not an element of CR. Therefore, this substitution can be expressed in the form of an indexed substitution:

$$\sigma = \{\ Vx_c = U^c_{CR}(WG/(x,0))\ /\ \forall x \in X\ ,\ \forall c \in CR' - CR\}$$

This substitution does not depend on the weighted root and verifies $\sigma(\ I^{IW}_{CR}(wg)\ )\ \subset\ I^{IW'}_{CR'}(wg)$.

Corollary 2 *Let wg be a weighted graph and IW, IW', CR and CR' be four intervals of Z:*

$$IW \subset IW' \text{ and } CR \subset CR' \ \Rightarrow\ I^{IW}_{CR}(wg)\ \leq\ I^{IW'}_{CR'}(wg)$$

The greater the counter range, the more precise and deep the interpretation is.

2.5 Unification of finite weighted graphs

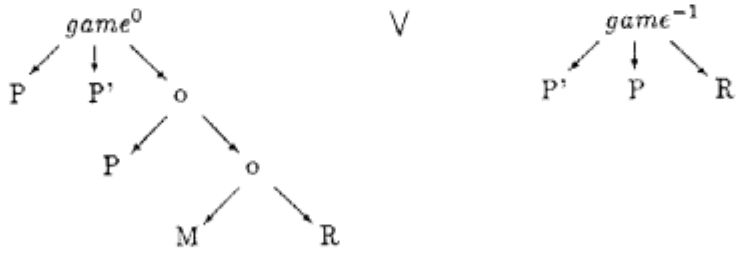
Definition 9 *Two weighted graphs are said to be unifiable from IW and in CR if the following system is solvable:*

$$\{\ \mathcal{U}^k_{CR}(wg)\ =\ \mathcal{U}^k_{CR}(wg')\ /\ \forall k \in IW\ \}.$$

Let σ be the most general unifier, then the result of the unification is denoted:

$$I^{IW}_{CR}(wg)\ \vee\ I^{IW}_{CR}(wg')\ =\ \{(k, \sigma(\ U^k_{CR}(wg)\ )\ /\ \forall k \in IW\}$$

Example Unification on CR = [0,n] and IW = [1,n]



Does there exist a substitution making the trees equal for all k $\in IW$?

Remark 2 *It is well known that unification is a fundamental concept and a crucial property in algebraic theory. If IW is infinite, the weighted graph unification is equivalent to solving an infinite system of equations.*

Unfortunately, the result of the unification of two weighted graphs, wg and wg', is generally not a weighted graph. However, it is possible to compute an approximate weighted graph, wg$\vee$wg', which does not depend on the IW and CR intervals and which ranges across the result of the unification:

1. *By decreasing CR, the unfolded trees of $wg \vee wg'$ are smaller than the unification of the unfolded trees of wg and wg'.*
2. *By increasing CR, the unfolded trees of $wg \vee wg'$ are greater than the unification of the unfolded trees of wg and wg'.*

This weighted graph, written $wg \vee wg'$, is a good approximation of the unification of wg and wg'. That is the meaning of the following theorem, and the next pages will prove that. The definition and the interpretation of weighted graphs are a generalisation of the directed graphs. The proof of the unification algorithm is also a generalisation of the directed graph algorithm.

Theorem 3 *Let wg and wg' be two unifiable finite weighted graphs, then there exist two constants, a and b, such that the range of unification of wg and wg' from IW in CR is obtained by the interpretation of $wg \vee wg'$ from IW in $(IW \cap CR)_{(a \rightarrow \leftarrow b)}$ and in $(IW \cup CR)_{(a \leftarrow\rightarrow b)}$:*

$$\mathcal{I}^{IW}_{CR_{sub}}(wg \vee wg') \leq S \leq \mathcal{I}^{IW}_{CR_{sup}}(wg \vee wg')$$

where $S = \mathcal{I}^{IW}_{CR}(wg) \bigvee \mathcal{I}^{IW}_{CR}(wg')$

$$CR_{sub} = (IW \cap CR)_{(a \rightarrow \leftarrow b)}$$

$$CR_{sup} = (IW \cup CR)_{(a \leftarrow \rightarrow b)}.$$

This means that the side effects of the unification have a constant size equal to a for the left-hand side effects and b for the right-hand side effects.

This is one of the most important properties, because after removing side effects, the approximate solution does not depend on the interpretation intervals. Within logic programming, this will give important consequences for characterising a finite sequence of recursive rewritings.

Corollary 3 *Let IW and CR be two intervals sharing at least (a+b) elements, then wg and wg' are unifiable from IW in CR iff they are unifiable from Z in Z.*

From proposition 1, it is obvious that if two weighted graphs are unifiable from Z in Z, they are unifiable from and in any intervals of Z. This corollary gives the reverse implication, that is, two weighted graphs are unifiable from Z and in Z if there exist two finite intervals, IW and CR, sharing sd elements where they are unifiable.

A consequence will be the decidability of the uniform termination of one global rewriting rule.

Corollary 4 *If IW and CR are equal to Z, the weighted graph unification is internal, that is, the result is another weighted graph:*

$$I_Z^Z(wg)\ )\ \vee\ I_Z^Z(wg')\ =\ I_Z^Z(wg \vee wg')$$

A consequence will be that the most general fixpoints of global rewriting rules are weighted graphs.

Lemma 3 *The weighted graphs can be supposed to share the same weighted structure. This means that they are similar, except for the weighted roots.*

Proof The semantics of the period obliges the weighted graphs to share the same period function. Moreover, it is easy to join their sets of nodes and the associated functions, named Lab and Succ.
So, let us define:

$wg = WG/(Root, w_R)$ and $wg' = WG/(Root', w'_R)$, where $WG = (X, Lab, Succ, Period)$.

Lemma 4 *Let wg and wg' be two weighted graphs and IW, IW', CR and CR' be four intervals:*

$$IW \subset IW' \text{ and } CR \subset CR' \ \Rightarrow\ I_{CR}^{IW}(wg)\ \vee\ I_{CR}^{IW}(wg')\ \leq\ I_{CR'}^{IW'}(wg)\ \vee\ I_{CR'}^{IW'}(wg')$$

Proof Directly from proposition 5

Notation 2 *Let [I,S] be an interval of Z and a, b two positive integers.*

$$[I+a, S-b] \text{ is denoted by } [I,S]_{(a\longrightarrow -b)} \text{ and } [I-a, S+b] \text{ by } [I,S]_{(a\longleftarrow\longrightarrow b)}$$

that is, the interval whose size has been added or reduced by constants on both sides.
In the same way, this operation will be applied also to infinite intervals, for example:

$$]-\infty, S]_{(a\longrightarrow -b)}\ =\ ]-\infty, S-b] \qquad or \qquad Z_{(a\longrightarrow -b)} = Z_{(a\longleftarrow\longrightarrow b)} = Z$$

These intervals are said to be a restriction or an extension of an interval, and are denoted $I \prec\!\sim I'$ (I is a restriction of I') or $I \sim\!\succ I'$ (I is an extension of I').

Lemma 5 *Weighted graphs wg and wg' share the same root node (Root = Root'), but the root weights are different ($w_R \neq w'_R$):*

1. *wg must be an acyclic weighted graph, otherwise the occur-check will not be verified for some finite IW and CR intervals.*

2. *Let h be the weighted graph, wg, all of whose periods are replaced by the Highest Common Factor of this period and* $(w_R - w'_R)$*:*

$$I^{IW}_{CR_{sub}}(h) \quad \leq \quad I^{IW}_{CR}(wg) \ \vee \ I^{IW}_{CR}(wg') \quad \leq \quad I^{IW}_{CR_{sup}}(h)$$

where $CR_{sub} \prec\!\sim (IW \cap CR)$ *and* $CR_{sup} \sim\!\succ (IW \cup CR)$.

Sketch of proof:

1. wg is a cyclic weighted graph.
   The weighted graph is supposed to be finite. For a well chosen finite interval CR, there is an input weight, k, such that:

$$\exists m = m_1.loop(x).m_2 \quad and \quad U^k_{CR}(wg)(m) = U^k_{CR}(wg')(m_1.m_2) = V_i.$$

   A loop whose weight is a multiple of $w'_R - w_R$ is enough. That is, the occur-check is not verified and unification is impossible. So now the weighted graph, wg, will be assumed to be acyclic.

2. To decrease the CR interval: CR_{sub}
   There are two constants, a_1 and b_1, (depending on WG, w_R and w'_R) such that:

$$\forall k \in IW_{sub} \ = \ (IW \cap CR)_{(a_1 \dashrightarrow b_1)}, \qquad \text{the unfolding of wg and wg' is complete,}$$

   that is, the unfolded trees of wg and wg' are similar, except for the indices of their variables. It is possible to show that there exist a_2 and b_2 (depending on WG, w_R and w'_R) such that the mgu of $I^{IW}_{CR}(wg) \ \vee \ I^{IW}_{CR}(wg')$ is greater than the following substitution:

$$\sigma = \{V_{i \bmod period(V)} = V_{i \bmod hcf(Period(V), w'_R - w_R)} \ / \ \forall i \in IW_{(a_2 \dashrightarrow b_2)}\}$$

   The only condition is that $(IW \cap CR)_{(a_2 \dashrightarrow b_2)}$ contains at least $period(V)$ elements.
   Let us define $WG = (X, Lab, Succ, Period)$, $wg' = (X, Lab, Succ, Period')$ and

$$\forall V \in range(WG) \ , \qquad Period'(V) = hcf(Period(V), w_R - w_{R'}).$$

   Hence, for $CR_{sub} = (IW \cap CR)_{(a_2 \dashrightarrow b_2)}$

$$I^{IW}_{CR}(wg) \ \vee \ I^{IW}_{CR}(wg') \ \geq \ I^{IW}_{CR_{sub}}(wg) \ \vee \ I^{IW}_{CR_{sub}}(wg') \ \geq \ \sigma(\ I^{IW}_{CR_{sub}}(wg)\ ) \ \vee \ \sigma(\ I^{IW}_{CR_{sub}}(wg')\ ),$$

$$\sigma(\ I^{IW}_{CR_{sub}}(wg)\ ) \ = \ I^{IW}_{CR_{sub}}(h),$$

   Therefore, $\qquad I^{IW}_{CR}(wg) \ \vee \ I^{IW}_{CR}(wg') \quad \geq \quad I^{IW}_{CR_{sub}}(h).$

3. To increase the CR interval: CR_{sup}
   For well chosen constants a_3 and b_3, the unfolding of wg and wg' is complete on the counter interval, $CR_{sup} = (IW \cup CR)_{(a_3 \longleftrightarrow b_3)}$. The mgu of the unification is less than:

$$\sigma = \{V_{i \bmod period(V)} = V_{i \bmod hcf(Period(V),(w_R - w'_R))} \ / \ \forall i \in CR_{sup} \}$$

$$I^{IW}_{CR}(wg) \vee I^{IW}_{CR}(wg') \leq I^{IW}_{CR_{sup}}(wg) \vee I^{IW}_{CR_{sup}}(wg') \leq \sigma(\ I^{IW}_{CR_{sup}}(wg)\ ) \vee \sigma(\ I^{IW}_{CR_{sup}}(wg')\ ).$$

   That is: $\quad I^{IW}_{CR}(wg) \vee I^{IW}_{CR}(wg') \leq I^{IW}_{CR_{sup}}(h).$

Example Let V^1 and V^{-1} be two weighted graphs,

$$I^{IW}_{CR_{sub}}(V^1 : 2) \quad \leq \quad I^{IW}_{CR}(V^1) \bigvee I^{IW}_{CR}(V^{-1}) \quad \leq \quad I^{IW}_{CR_{sup}}(V^1 : 2)$$

where $CR_{sub} = (IW \cap CR)_{(1 \rightarrow \leftarrow 1)}$ and $CR_{sup} = (IW \cup CR)_{(1 \longleftrightarrow 1)}$.

Lemma 6 *The weighted graphs, wg and wg', have different root nodes. Let wg' be the weighted structure, WG, whose Succ function has been modified as follows:*

$\forall x \in X,\ Succ'(x,i) = Succ(x,i) = (x', w)$ *if* $w \neq Root'$
$= (Root, w + w_R - w'_R)$ *if* $Succ(x,i) = (Root', w)$

That is, the node, Root', has been replaced by the node, Root, with a correction of the arrow weights ($w_R - w'_R$). Let us define h and h' as the weighted graphs obtained this way:

$$h = WG'/(Root, w_R) \quad \text{and} \quad h' = WG''/(Root', w'_R).$$

Then, the following inequation is verified:

$$I^{IW}_{CR_{sub}}(h) \bigvee I^{IW}_{CR_{sub}}(h') \quad \leq \quad I^{IW}_{CR}(wg) \bigvee I^{IW}_{CR}(wg') \quad \leq \quad I^{IW_{sup}}_{CR}(h) \bigvee I^{IW_{sup}}_{CR}(h')$$

where $CR_{sub} \prec\!\sim (IW \cap CR)$ *and* $IW_{sup} \sim\!\succ (IW \cup CR)$.

Proof To obtain a good property for unfolding, suppose that all the counter values of Root' (root node of wg') belong to $IW_{(|w'_R| \rightarrow \leftarrow |w'_R|)}$.
In this case, the fundamental property of the directed graph can be generalised.
For any unifier, σ, of wg and wg':

$$\sigma(\ I^{IW}_{CR}(wg)\ ) = \sigma(\ I^{IW}_{CR}(h)\ ) = \sigma(\ I^{IW}_{CR}(wg')\ ) = \sigma(\ I^{IW}_{CR}(h')\ )$$

This is proved by induction on the length of the paths in weighted graphs wg, wg', h and h'.
Let m be a path and k be an input weight of k.
Suppose, first, that $\| m \| = 0$. Then the result is obvious because the roots are unchanged.
Next, suppose that the results hold for $\| m \| \leq n - 1$. Consider a path, $m = i_1.i_2 \ldots i_n$.

1. There exists no i_k such that $Desc_{CR}((Root, w_R + k), i_1..i_k) = (Root', w)$.
   Then the descendant function is unchanged on the path:

$$\sigma(\ U^k_{CR}(wg)\ )(m) = \sigma(\ U^k_{CR}(h)\ )(m).$$

2. There exists i_k such that $Desc_{CR}((Root, w_R + k), i_1..i_k) = (Root', w)$.
   Then the Descendant function has been changed in this path:

$$Desc'_{CR}((Root, w_R + k), m) = Desc'_{CR}((Root, w + w_R - w'_R), i_{k+1}..i_n)$$

   (a) $\sigma(\ U^k_{CR}(wg)\ )(m) = \sigma(\ U^{w-w'_R}_{CR}(wg')\ )(i_{k+1}..i_n)$

   (b) $\sigma(\ U^k_{CR}(h)\ )(m) = \sigma(\ U^{w-w'_R}_{CR}(h)\ )(i_{k+1}..i_n)$

   However, the induction hypothesis can be applied because $w - w'_R \in IW$:

$$\sigma(\ U^{w-w'_R}_{CR}(wg')\ )(i_{k+1}..in) = \sigma(\ U^{w-w'_R}_{CR}(h)\ )(i_{k+1}..i_n)$$

   Using equations (a) and (b) and the induction hypothesis,
   $\sigma(\ U^k_{CR}(wg)\ )(m) = \sigma(\ U^k_{CR}(h)\ )(m)$.

Similarly, for wg' and h': $\sigma(\ U^k_{CR}(wg')\ )(m) = \sigma(\ U^k_{CR}(h')\ )(m)$.
However, the hypothesis was that all the counter values of Root' (root node of wg') belong to $IW_{(|w'_R| \rightarrow \leftarrow |w'_R|)}$.
Hence, it is possible to increase IW or to reduce CR:

$$CR_{sub} \prec\sim (IW \cap CR) \quad \text{and} \quad IW_{sup} \sim\succ (IW \cup CR)$$

Lemma 7 *Weighted graphs, wg and wg', have different root nodes and the node of wg' is labelled by a variable, V:*

1. *If V is periodic, weighted graph wg must be acyclic, otherwise the unifier does not exist.*
2. *Let h be the weighted graph, wg, whose every period is replaced by the Highest Common Factor of this period and $(w_R - w'_R)$:*

$$I^{IW}_{CR_{sub}}(h) \quad \leq \quad I^{IW}_{CR}(wg) \ \vee\ I^{IW}_{CR}(wg') \quad \leq \quad I^{IW}_{CR_{sup}}(h)$$

   where $CR_{sub} \prec\sim (IW \cap CR)$ and $CR_{sup} \sim\succ (IW \cup CR)$.

Proof If variable V is periodic, lemma 3 is applicable because there exists $IW_{sub} \prec\sim (IW \cap CR)$ such that:

$$I^{IW_{sub}}_{CR}(wg) \ \vee\ I^{IW_{sub}}_{CR}(wg_1) \ \vee\ I^{IW}_{CR}(wg) \ \vee\ I^{IW}_{CR}(wg') \quad \leq \quad I^{IW}_{CR}(wg) \ \vee\ I^{IW}_{CR}(wg')$$

and

$$I^{IW}_{CR}(wg) \ \vee\ I^{IW}_{CR}(wg') \quad \leq \quad I^{IW}_{CR}(wg) \ \vee\ I^{IW}_{CR}(wg_1) \ \vee\ I^{IW}_{CR}(wg')$$

where wg_1 has the same root as wg, but the root weight is $w_R + Period(V)$.
Hence, applying lemma 3, there exist CR_{sub} and CR_{sup} such that: let h be the weighted graph, wg, all of whose periods are replaced by the Highest Common Factor of this period and $(w_R - w'_R)$:

$$I^{IW}_{CR_{sub}}(h) \ \vee\ I^{IW}_{CR_{sub}}(wg') \quad \leq \quad I^{IW}_{CR}(wg) \ \vee\ I^{IW}_{CR}(wg')$$

$$I^{IW}_{CR}(wg) \ \vee\ I^{IW}_{CR}(wg') \quad \leq \quad I^{IW}_{CR_{sup}}(h) \ \vee\ I^{IW}_{CR_{sup}}(wg').$$

Moreover, weighted graph wg can be assumed to have no occurrences of variable V (Lemma 4). The unfolding of h is complete on CR_{sup} and the periodicity of variable V is verified in the unfolding of h:

$$I^{IW}_{CR_{sup}}(h) \ \vee \ I^{IW}_{CR_{sup}}(wg') \quad = \quad I^{IW}_{CR_{sup}}(h).$$

This means that this inequation is verified:

$$I^{IW}_{CR_{sub}}(h) \quad \leq \quad I^{IW}_{CR}(wg) \ \vee \ I^{IW}_{CR}(wg') \quad \leq \quad I^{IW}_{CR_{sup}}(h)$$

where $CR_{sub} \prec\sim (IW \cap CR)$ and $CR_{sup} \sim\succ (IW \cup CR)$

Lemma 8 *Weighted graphs wg and wg' have different root nodes, but these nodes are labelled by function symbols.*

1. *The label must be the same, otherwise the unifier does not exist.*
2. *The unification of wg and wg' can cover the range of the unification of their sub-graphs in well chosen intervals of interpretation.*

Proof For a well chosen IW_{sub} interval, the unification of wg and wg' is equivalent to the unification of all the couples of their sub-graphs. It is the same for CR_{sup}. Let us choose $a = b = \mid w_R \mid + \mid w'_R \mid$

$$IW_{sub} = (IW \cap CR)_{(a \rightarrow \leftarrow b)} \qquad \text{and} \qquad CR_{sup} = (IW \cup CR)_{(a \longleftrightarrow b)}.$$

For these intervals, IW_{sub}, CR or IW, CR_{sup}, finding the mgu of wg and wg' is equivalent to finding the mgu of $(wg_i, wg'_i)_{(0 \leq i \leq arity)}$.

Using these lemmas, a unification algorithm can be presented in the form of a generalisation of the directed graph algorithm of [Fages]. Let us denote the highest common factor as hcf. This algorithm is composed of two steps; the first is the application of the unification procedure, and the second, easy to define, is the finite weighted graph-check, that is, occur-check:

Procedure **Unification** $((Root, w_R), (Root', w'_R))$;
Begin
If (Root = Root')
  Then
    If $w_R \neq w'_R$
      Then % wg must be acyclic %
        $\forall V \in wg, Period(V) := hcf(Period(V), \mid w_R - w'_R \mid)$
  Else
    If (Root' is labelled by the variable V')
      Then % To replace the arrows going to Root' %
        $\forall x$, if $Succ(x,i) = (Root', w')$ then $Succ(x,i) := (Root, w' + w_R - w'_R)$
        If $Period(V') \neq 0$
          Then % wg must be acyclic %
            $\forall V \in wg, Period(V) := hcf(Period(V), Period(V'))$
      Else
        If (Root is labelled by the variable V)
          Then % To replace the arrows going to Root %
            $\forall x$, if $Succ(x,i) = (Root, w)$ then $Succ(x,i) := (Root', w + w'_R - w_R)$
            If $Period(V) \neq 0$
              Then % wg' must be acyclic %
                $\forall V' \in wg', Period(V') := hcf(Period(V), Period(V'))$
          Else
            If (Root and Root' are labelled by the same function)
              Then % To replace the arrows going to Root' %
                $\forall x$, if $Succ(x,i) = (Root', w')$ then $Succ(x,i) := (Root, w' + w_R - w'_R)$
                For i:= 1 A n Do
                  % $Succ(Root, i) = (x_i, w_i)$ and $Succ(Root', i) = (x'_i, w'_i)$%
                  Unification $((x_i, w_i + w_R), (x'_i, w'_i + w'_R))$;
                Done
              Else Fail – The unifier does not exist.
End

Proof of theorem 1 The unification algorithm computes a weighted graph, h, such that:

$$I_{CR_{sub}}^{IW_{sub}}(h) \quad \leq \quad I_{CR}^{IW}(wg) \ \vee \ I_{CR}^{IW}(wg') \quad \leq \quad I_{CR_{sup}}^{IW_{sup}}(h)$$

where $IW_{sub}, CR_{sub} \prec\sim (IW \cap CR)$ and $IW_{sup}, CR_{sup} \sim\succ (IW \cap CR)$.
Moreover, it is easy to understand that:

if $I^{IW}_{CR}(wg) \vee I^{IW}_{CR}(wg') \leq I^{IW_{sup}}_{CR_{sup}}(h)$
then $I^{IW}_{CR}(wg) \vee I^{IW}_{CR}(wg') \leq I^{IW}_{CR_{sup}}(h)$.

In a similar way, there exists CR'_{sub} such that:
if $I^{IW}_{CR}(wg) \vee I^{IW}_{CR}(wg') \geq I^{IW_{sub}}_{CR_{sub}}(h)$ then $I^{IW}_{CR}(wg) \vee I^{IW}_{CR}(wg') \geq I^{IW}_{CR'_{sub}}(h)$.
Let us define (Root",w") as the weighted root of h and for simplicity let us suppose that $IW_{sub} = [a, b]$.
If all the indices of the special variable of Root" belong to [a+w",b+w"], then the interpretation of h on IW and CR_{sub} can be split up into three independent parts:

$$I^{IW}_{CR'_{sub}}(h) = I^{IW_l}_{CR'_{sub}}(h) \cup I^{IW_{sub}}_{CR'_{sub}}(h) \cup I^{IW_r}_{CR'_{sub}}(h)$$

where $IW - IW_{sub} = IW_l \cup IW_r$ and using the following remarks:

1. $I^{IW_{sub}}_{CR'_{sub}}(h) \leq I^{IW_{sub}}_{CR_{sub}}(h)$ because $CR'_{sub} \subset CR_{sub}$.
2. $I^{IW_l}_{CR'_{sub}}(h)$ and $I^{IW_r}_{CR'_{sub}}(h)$ are lists of distincts variables.

Hence, the conclusion is that this algorithm computes a weighted graph, $wg \vee wg'$, such that:

$$I^{IW}_{CR_{sub}}(wg \vee wg') \leq I^{IW}_{CR}(wg) \vee I^{IW}_{CR}(wg') \leq I^{IW}_{CR_{sup}}(wg \vee wg')$$

where $CR_{sub} \prec\sim (IW \cap CR)$ and $CR_{sup} \prec\sim (IW \cap CR)$.

The occur-check must be verified. However, some unfolded results of wg $\vee$ wg' from CR_{sub} or CR_{sup} are infinite iff wg $\vee$ wg' is not a finite weighted graph. The unification is possible iff the algorithm computes a finite weighted graph.

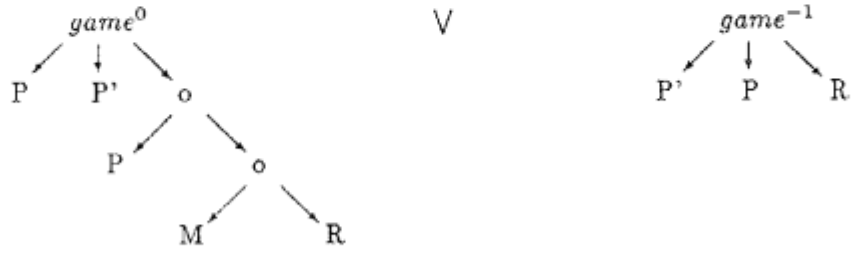
Remark 3 *Weighted graph and directed graph algorithms.*
The weighted graph unification algorithm is the same as the directed graph one, except for the computation of weights and periods which has been added. The termination of this unification algorithm is verified as in the directed graph algorithm, that is, at each procedure call one node disappears.

Corollary 5 *Let wg and wg' be two weighted graphs and $wg \vee wg'$ be their unifier, and let us define g, g' and d(wg $\vee$ wg') as the directed graphs obtained from wg, wg' and wg $\vee$ wg' by removing the weights and periods:*

$$wg \vee wg' \text{ exists} \implies g \vee g' \text{ exists and is equal to } d(wg \vee wg')$$

Proof Weighted graph unification is a generalisation of directed graph unification. Moreover, if the weights and the periods are removed in weighted graphs wg, wg' and $wg\vee$ wg', then the result corresponds to directed graph unification.

Example Let us compute the following unification:



This unification will characterise the behaviour of the following rule:

$$game(P\ ,\ P'\ ,\ P\ \circ\ M\ \circ\ R) \quad \rightarrow \quad game(P'\ ,\ P\ ,\ R)$$

which expresses a chess game between players P and P':

1. The first argument is the name of the player who has to play.
2. The second argument is the name of the player who will have to play at the next turn.
3. The third argument is the list of the name of the players and their moves:

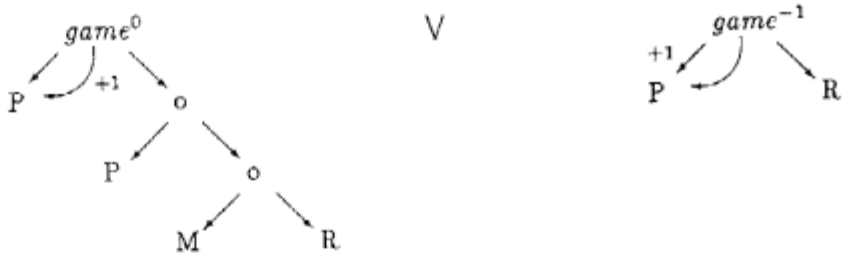
$$P \circ M_0 \circ P' \circ M_1 \circ P \circ M_2 \circ ...$$

The first weighted graph is the left term of this rule with a null root weight, and the second one is the right term of this rule with a root weight of -1.
Let us apply the unification algorithm:

1. The root nodes are labelled by the same function symbol and the unification of these weighted graphs is obtained by the unification of their sub-graphs:
2. Unification of the first weighted graphs: P^0 and P'^{-1}
   Node P' is replaced by node P with a correction of +1 :

$$P'^{-1} \xrightarrow{+1} P^0$$

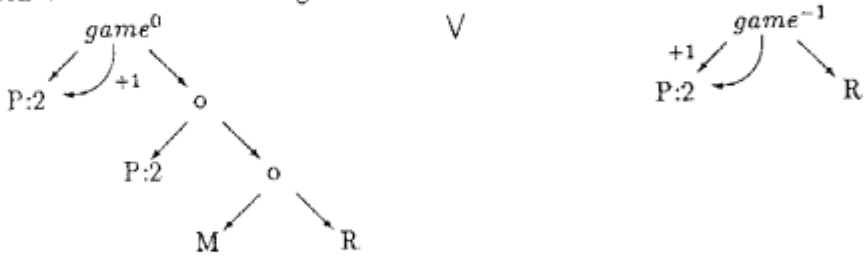
   The unification becomes the following:



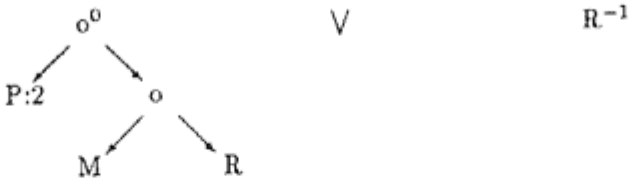
3. Unification of the second sub-graphs: P^1 and P^{-1}
   These weighted graphs share the same root node. Period 2 is put on variable P:

$$P^1 \vee P'^{-1} = P^1 : 2 = P'^{-1} : 2$$

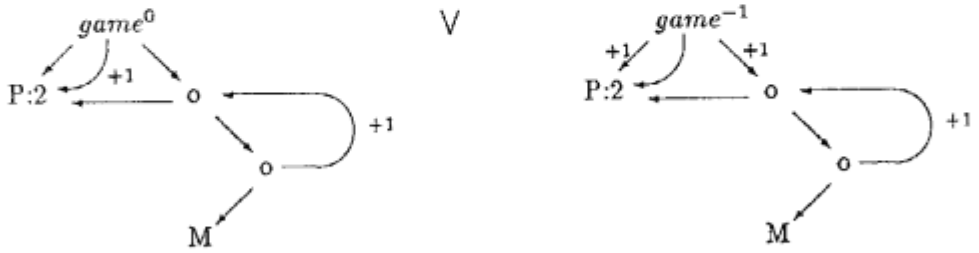
   The unification becomes the following:



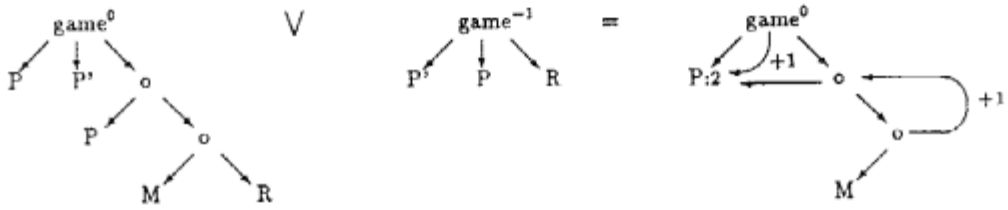
4. Unification of the third sub-graphs:



Node R is replaced by node o with a correction of +1. The unification becomes the following:



The unification is now finished and one of these weighted graphs can be chosen as the approximate weighted graph, for instance, the first one:



It is possible to verify that $CR_{sub} = (IW \cap CR)_{(1 \rightarrow \leftarrow 0)}$ and $CR_{sup} = (IW \cup CR)_{(1 \leftarrow\rightarrow 0)}$.

This weighted graph characterises the behaviour of the rule:

$$game(P\ ,\ P'\ ,\ P\ \circ\ M\ \circ\ R) \quad \rightarrow \quad game(P'\ ,\ P\ ,\ R)$$

The periodicity of the first and second arguments is expressed by the period of P and the third argument is characterised by the third sub-graph, already used for introducing the period notion (cf 2.1.2).

Because the definition, interpretation and unification algorithm are generalisations of the directed graphs, the equation is still true within the directed graphs if the weights and the periods are removed in the unification equation.

3 Systems of Equations, Algorithms and Properties.

3.1 Introduction

Generally, the idempotent most general unifier is used for expressing the constraints of unification [Eder 85]. A more general syntactic object is the *system of equations*, that is, a set of equations of the form $t = t'$ where t and t' are finite trees.

However, the advantages of the systems of equations are, first, the memory size; and second, the efficiency of the unification algorithm, and also that within the context of termination and complexity, the behaviour of resolution can be understood better through the constraints between the variables than from the final susbtitution of these variables.

The reduced systems of equations introduced by [Colmerauer 84] express this feeling which is strongly linked with the representative notion introduced by [Huet 1976]. The directed acyclic graph and the directed graph are based on the same idea: common information is shared [Fages 83]. Similarly, within intelligent backtracking, an important aspect is to express the dependency graph of variables.

The goal of this section is to show that the shortest expression of a reduced system can be used to express as clearly as possible the constraints generated by a system of equations; however, it can also be used as a complexity measure to understand the convergent, invariant or periodic phenomena for some recursive Prolog programs.

3.2 Reduced systems of equations

A system of equations is said to be solvable if there exists a grounding substitution, σ, which makes t and t' ground and equal for all equations, $t = t'$, of the system. Two systems, E1 and E2, are said to be equivalent if they have the same grounding substitutions.

Definition 10 *An endless system is a system of equations in which every term which occurs as the right-hand side of an equation also occurs as the left-hand side of an equation [Colmerauer 84].*

Definition 11 *A system of equations is said to be reduced if the left-hand sides of its equations are distinct variables, and it contains no endless sub-system [Colmerauer 84] .*

Definition 12 *A system is said to be acircular iff it has no sub-system where every variable which occurs as the right-hand side of an equation also occurs as the left-hand side of that equation.*

Proposition 3 *A reduced system of equations verifies the occur-check iff it is acircular.*

Proof There is no cyclic sequence of equalities between variables because the reduced system contains no endless sub-system.

Notation 3 *Let t be a tree, E be a system of equations and rs be a reduced system. Then we will denote:*

- $depth(x) = 1$ *when* x *is a variable or a constant*
  $depth(f(t_1, \cdots, t_n)) = 1 + max(depth(t_i))$
- $size(x) = 1$ *when* x *is a variable or a constant*
  $size(f(t_1, \cdots, t_n)) = \sum size(t_i))$
- *domain(rs) is the set of variables occurring on the left-hand side of rs.*

Reduction algorithm
The reduction algorithm is a non-deterministic series of six types of transformations:

(1) $f(t_1,\cdots,t_n) = f(t'_1,\cdots,t'_n)$	replace by the equations $t_1 = t'_1,\cdots,t_n = t'_n$
(2) $f(t_1,\cdots,t_n) = g(t'_1,\cdots,t'_n)$	halt with failure
(3) $U = U$	delete the equation
(4) $t = U$ $(t \notin Var)$	replace by the equation $U = t$
(5) $U = V$ $(V \in Var,\ V \neq U)$	if there are other occurrences of U, replace U by V in every other equation.
(6) $U = t$, $U = t'$ $(t,\ t' \notin Var)$	replace (U = t') by (t = t') if $depth(t) \leq depth(t')$

Within the finite trees, the occur-check must be verified, that is, the system is acircular:

$$\nexists\ (U_1 = t_1),\cdots,(U_k = t_k) \ \textit{where}\ \forall\ 2 \leq i \leq k,\ U_i \in var(t_{i-1})\ \textit{and}\ U_1 \in var(t_k)$$

Remark 4 *In the [Lassez, Maher, Marriott] algorithm which defines the most general unifier of a system, the last two transformations are replaced by the following one:*

$U = t$ *where t is different from U*	*if U appears in t, then halt with failure (occur-check), otherwise, replace U by t in every other equation.*

Hence, in the usual algorithm, no variables may occur on both the right and left hand sides of the solving system of equations. In this case, the size of the solved system is much more important, and contains many redundancies.

Theorem 4 *The reduction algorithm applied to a set of equations, E, will return an equivalent set of equations in a reduced form if and only if E is solvable. It will return failure otherwise.*

Proof Let depth be the following function from the set of systems of equations to N, the set of natural integers:
- $depth(t = t') = max(depth(t), depth(t'))$
- $depth(E) = \sum_{(t=t')\in E} depth(t = t')$

Using this function, the termination of the algorithm can be checked:

- (1) and (3) :	strictly diminish the depth of the system.
- (2) :	halts the reduction algorithm.
- (4), (5) and (6):	do not change the depth of the system, but can be applied successively at most a finite number of times (= number of equations).

It is easy to verify the correctness. None of the transformations affect the grounding solutions of the system and if transformation (2) generates a failure, this means that there is an equation (t=t') where t and t' are not unifiable, that is, the system is not solvable.

If the system of equations obtained after the first stage without failure is circular, the occur-check is not verified. Transformations (3) and (5) have deleted all the cyclic equalities of variables.

Thus, if the algorithm, with or without occur-check, terminates without failure, then the system is solvable within the finite or rational trees, and one of its idempotent most general unifiers can be easily computed.

Theorem 5 *The idempotent most general unifier of a reduced system can be defined through a function, called representative:*

$$
\begin{array}{lll}
repr(\ f(t_1,\cdots,t_n)\ ) & = & f(repr(t_1),\cdots,repr(t_n)) \\
repr(U) & = & repr(t) \qquad\qquad \textit{if } \exists (U = t) \in rs \\
repr(U) & = & U \qquad\qquad \textit{otherwise.}
\end{array}
$$

A most general unifier of the reduced system, rs, is: $\{U \leftarrow repr(U) \ / \ \forall\ U \in domain(rs)\ \}$.

3.3 Congruent systems

Definition 13 *A congruent system is a congruence,* $\Re$*, and a mapping function from* $Var/\Re$ *to* $(M(F,Var) - Var)/\Re$*, that is, the set of finite trees* ($\notin Var$) *built from F (set of functions) and Var (set of variables) modulo of the congruence,* $\Re$.
A congruent system can be expressed from its congruence and a set of equations of the form $(C = t)$ *, where C is a class of* $\Re$ *and* $t \in (M(F,Var) - Var)/\Re$.

Definition 14 *A congruent system is said to be acircular if it does not contain any sub-system where every class which occurs as the right-hand side of a congruent equation also occurs as the left-hand side of an equation.*

Proposition 4 *A congruent system verifies the occur-check iff it is acircular.*

Definition 15 *Let rs be a reduced system, then* $\Re_{rs}$ *is the congruence on the variables defined from the equalities of variables* $(U = V)$ *of rs, that is, the reflexive and transitive closure of:*

$$U\ \Re_{rs}\ V \ \ if\ \ \exists\ (U = V)\ \in\ rs$$

The congruence of a reduced system expresses the equalities between variables explicitly written.

A congruent system is an alternative to define a reduced system, that is, any reduced system can be expressed in the form of a congruent system. However, in this last form, there is no ambiguity about the form of the equations of variables $(U = V) \equiv (V = U)$.
It is the reason why the definition will be used in the following. Within the congruent systems, the reduction algorithm can be easily translated to a congruent algorithm because the notion of variables has been just substituted by the notion of class (or distinct set of variables):

Congruent algorithm
Let E be a system of equations. Let us suppose that all variables of E are substituted by the class composed of itself: U replaced by $\{U\}$, that is, the class of U in the empty congruence. The algorithm is a non-deterministic series of six types of transformations:

(1) $f(t_1,\cdots,t_n) = f(t'_1,\cdots,t'_n)$	replace by equations $t_1 = t'_1,\cdots,t_n = t'_n$
(2) $f(t_1,\cdots,t_n) = g(t'_1,\cdots,t'_n)$	halt with failure
(3) $C = C$	delete the equation
(4) $t = C$	replace by the equation $C = t$
(5) $C = C'$	delete the equation and merge C and C' $\{\ C, C' \leftarrow C \cup C'\ \}$
(6) $C = t\ ,\ C = t'$	replace (C = t') by (t = t') if $depth(t) \leq depth(t')$

Within the finite trees, the occur-check must be also done:

$$\nexists\ (C_1 = t_1),\cdots,(C_k = t_k) \ \ where\ \ \forall\ 2 \leq i \leq k,\ C_i \in t_{i-1}\ \ and\ \ C_1 \in t_k$$

Example Let rs be the following reduced system:
$\{U = a, V = U, X = b(U,U), Y = U, Z = X\}$
Hence, an equivalent form is: $\{C_1 = a, C_2 = b(C_1, C_1)\}$
where C_1 and C_2 are the classes of $\Re_{rs}$: $C_1 = \{U, V, Y\}$, $C_2 = \{X, Z\}$

Proposition 5 *Within the finite or rational trees, any reduced system can be expressed in the form of a congruent system which is circular or acircular according to the reduced system.*

Congruent unification algorithm
A unification algorithm of two congruent systems, cs_1 and cs_2, will be the same after a harmonisation of their congruences, $\Re_1$ and $\Re_2$.

1. Let $\Re$ be the transitive closure of $\Re_1 \cup \Re_2$, then all the classes of $\Re_1$ and $\Re_2$ in cs_1 and cs_2 are replaced by their new classes of $\Re$.

2. After this merge of the two congruences, the congruent algorithm can be applied to the union of the congruent equations of cs'_1 and cs'_2.

The usual notion of eliminable variables of a system of equations depends on the chosen reduced form of it. For example, U is the eliminable variable of $(U = V)$, but not of $(V = U)$. However, there is no ambiguity in the system, $(U = V\ ,\ V = a)$. We will use the notions of eliminable and possibly free variables.

Definition 16 *A class is said to be free if there is no congruent equation where it occurs on the left-hand side, otherwise, it is said to be substituted.*
The variable, U, is said to be

1. *eliminable if its class is substituted:* $eliminable(cs) = \bigcup_{(C=t)\in cs} C$

2. *possibly free if its class is free:* $pfree(cs) = \{U \in vars(cs) \ / \ \nexists (\bar{U} = t) \in cs\ \}$

The notion, fully free variable, will be a variable belonging to a free singleton class.

An eliminable variable is substituted in any idempotent most general unifier, and a possibly free variable is free in some of them. Obviously, in this definition, there is no ambiguity, that is, two equivalent variables (in $\Re_{cs}$) have the same type. Moreover, the type of a variable is the same in any congruent form of a solvable system of equations.

Proposition 6 *Two equivalent congruent systems have the same eliminable variables, and the same possibly free variables:*

$$cs_1 \equiv cs_2 \quad \Rightarrow \quad eliminable(cs_1) = eliminable(cs_2)\ ,\ pfree(cs1) = pfree(cs_2)$$

Two equivalent congruent systems also have the same fully free variables.

Proposition 7 *Let cs_1 and cs_2 be two congruent systems, and cs be the congruent system of their union, then:*

1. $eliminable(cs) \supset eliminable(cs_1) \cup eliminable(cs_2)$

2. $pfree(cs) \subset pfree(cs_1) \cup pfree(cs_2)$

Proposition 8 *Let cs_1 and cs_2 be two congruent systems having no common eliminable variables, and cs be the congruent system of their union, then:*

$$cs \;=\; cs_1 \cup cs_2$$

that is, obtained by the union of their congruences and their congruent equations.

Any congruent system is solvable and its idempotent most general unifier can be defined through the function, *representative*. Let us denote $\bar{C}$, the canonical element of the class, C, of congruence.

$$\begin{array}{lll} repr(\, f(t_1,\cdots,t_n)\,) \;=\; f(repr(t_1),\cdots,repr(t_n)) & & \\ repr(C) \;=\; repr(t) & \text{if } \exists (C = t) \in cs \\ repr(C) \;=\; \bar{C} & \text{otherwise.} \end{array}$$

A most general unifier of the congruent system, cs, is:

$$\{U \leftarrow repr(U) \,/\, \forall\, U \in vars(cs)\,,\; U \neq repr(U)\,\}.$$

The following functions have be translated in the congruent system context. Let t be a congruent tree and cs be a congruent system, then

- $fsize(t)$ = Number of function occurrences
  $fsize(cs) = \sum_{(t \in cs)} fsize(t)$
- $depth(x) = 1$ when x is a class or a constant
- $vars(cs)$ = the set of variables of the congruence, $\Re_{cs}$.
- $dim(cs)$ = the number of free classes in cs.
  (that is, the number of free variables in the idempotent mgu)

3.4 Minimal systems

In some cases, it is important to express the constraints between the variables as clearly as possible. This corresponds to the shortest expression.

Definition 17 *A congruent system, cs, is said to be minimal if for all equivalent congruent systems, cs', fsize(cs') are greater or equal than fsize(cs):*

$$\forall\, cs' \equiv cs,\;\; fsize(cs') \;\geq\; fsize(cs)$$

The viewpoint is quite opposite to the usual idempotent most general unifier.

Example Let cs_1 be the following congruent system:
$\{C_1 = a, C_2 = b(a,Y), C_3 = a, C_4 = b(V,a)\}$
where C_i are the classes of congruence: $C_1 = \{U,V\}$, $C_2 = \{X\}$, $C_3 = \{Y\}$, $C_4 = \{Z\}$
Hence, its minimal form is:
$\{C_1 = a, C_2 = b(C_1,C_1)\}$ where $C_1 = \{U,V,Y\}$, $C_2 = \{X,Z\}$
Let cs_2 be equal to $\{C = a(a(C))\}$ where C is a class of congruence,

then its minimal form is: $\{C = a(C)\}$

The minimal form corresponds to the shortest expression (fsize), but also to the most complete congruence ($\Re$), that is, all the constraints of equality between the variables are explicit in the minimal system.

Moreover, the minimal form can be used as the normal form of a solvable system and complexity measure:

Theorem 6 *There exists one and only one minimal form of a solvable system of equations. Two solvable systems of equations are equivalent iff they have the same minimal form.*

Lemma 9 *Let E be a solvable system and ms be its minimal form, then*

$$fsize(ms) \leq fsize(E)$$

Proof Any solvable system, E, has a reduced form, rs, such that:

$$fsize(rs) \leq fsize(E)$$

This is a consequence of the reduction algorithm of Colmerauer. Moreover, it is easy to see that there is a congruent form of rs which has the same fsize value, that is:

$$\forall E\ ,\ \exists cs \equiv E\ ,\ fsize(cs) \leq fsize(E)$$

Lemma 10 *Let σ be the idempotent most general unifier of a minimal system, ms, then*

$$U\ \Re_{ms}\ V \quad \Leftrightarrow \quad \sigma(U) = \sigma(V).$$

Proof By contradiction, let us suppose that there exist two variables which are substituted by the same tree in the mgu, but do not belong to the same class of congruence. It is easy to see that by merging the classes of these variables, the obtained system would be equivalent, but smaller because one congruent equation would be redundant.

Proof of the theorem By contradiction, let ms_1 and ms_2 be two different equivalent minimal systems, then there exist two different equations, $(C_1 = t_1) \in ms_1$ and $(C_2 = t_2) \in ms_2$, such that $C_1 \cap C_2$ is not empty.
Using the previous lemma, these classes must be equal: $C_1 = C_2$, that is, the congruent terms, t_1 and t_2, must be different.
Let $t_1 \wedge t2$ be the following term: $\forall m \in dom(t_1) \cap dom(t_2)$

$$\begin{aligned}(t_1 \wedge t2)(m) &= t_1(m) && \text{if } t_1(m) \text{ is a class}\\ &= t_2(m) && \text{otherwise}\end{aligned}$$

If $t_1 \wedge t2$ has the same fsize as those of t_1 and $t2$, then terms t_1 and t_2 are equal, except for some classes occurring in them.
Thus, there is an immediate contradiction. The variables occuring in these different classes must be substituted by the same tree in the mgu, and must, therefore, be equivalent, that is, belong to the same class of congruence.

If $t_1 \wedge t2$ does not have the same fsize as those of t_1 and $t2$, then this fsize must be smaller than at least the fsize of one of the two terms. Moreover, it is easy to see that we may substitute

t_1 and t_2 by $t_1 \wedge t2$ in the equations of ms_1 and ms_2. The new systems are equivalent to the systems, ms_1 amd ms_2, but the fsize of at least one system has strictly decreased. Hence, in this case, there is also a contradiction.

Remark 5 *For any solvable system, E, in addition to dim(cs), the following notions are well defined and can be used as complexity measures:*

1. $fsize_{min}(E)$ *is the fsize of its minimal form.*
2. $\Re_{max}(E)$ *is the congruence of its minimal form.*

Corollary 6 *Let E_1 and E_2 be two solvable systems of equations, let us suppose that $E_1 \cup E_2$ is solvable, then:*

$$\Re_{max}(E_1 \cup E_2) \supset \Re_{max}(E_1) \cup \Re_{max}(E_2)$$

$$fsize_{min}(E_1 \cup E_2) \leq fsize_{min}(E_1) + fsize_{min}(E_2)$$

Although the minimal notion can be defined within the finite or rational trees, a simple minimalisation algorithm is described here within the acircular congruent systems:

Theorem 7 *A acircular congruent system, cs, is minimal iff for any pair of congruent equations of cs, $(C_1 = t_1)$ and $(C_2 = t_2)$, the terms, t_1 and t_2, are different and neither of them occurs in the other one as a sub-term.*

Proof

$\Rightarrow$

Any congruent system verifies the given property about the pairs of congruent equations; otherwise, it would be easy to build an equivalent congruent system which would be smaller by replacing this subterm by a class.

$\Leftarrow$

Let cs be a congruent system which verifies the given condition about any pair of its equations, and ms be the minimal form of cs. Let us show, by contradiction, that these systems are equal.

Let t_1 and t_2 be two terms. If they verify the condition expressed in the theorem, then terms t_1 and t_2 will be said to be strongly different, denoted $t_1 \neq_s t_2$.

Let U be a variable, $(C_{cs} = t_{cs})$ be an equation of cs, and $(C_{ms} = t_{ms})$ be an equation of ms such that U belongs to two classes, C_{cs} and C_{ms}. If these equations are different, then either the classes are different or the congruent terms are different.

1. If the classes are different, then because of the minimal hypothesis, C_{cs} must be a subset of C_{ms} and thus there exists at least one other congruent equation, $(C'_{cs} = t'_{cs})$, such that C'_{cs} is also a subset of C_{ms}.

$$C_{cs} \neq C_{ms} \Rightarrow \exists (C'_{cs} = t'_{cs}) \ , \ C_{cs} \cap C_{ms} \neq \emptyset \ , \ t_{cs} \neq_s t'_{cs}$$

2. If the congruent terms are different, that is, there exists a path, m, such that t_{ms}/m and t_{cs}/m are different.

   (a) If these both sub-terms are classes, then there exist two equations, $(C1_{cs} = t1_{cs})$ and $(C2_{cs} = t2_{cs})$, such that C_1 and C_2 are subsets of the same class of ms.

(b) If only one of these sub-terms is a class, then there exist two other different equations, $(C'_{cs} = t'_{cs})$ and $(C'_{ms} = t'_{ms})$, such that C_{cs} and C_{ms} have common variables.

By recursive applications of that and because of the acircular hypothesis, this reasoning generates a contradiction.

Minimal algorithm
This algorithm is based on a non-deterministic series of two types of transformations:

(1) $(C_1 = t_1)$, $(C_2 = t_1)$	delete $(C_2 = t_1)$ and merge C_1 and C_2 $\{C_2,\ C_1 \leftarrow C_1 \cup C_2\}$
(2) $(C_1 = t_1)$, $(C_2 = t_2)$ (t_2 occurs in t_1)	replace the sub-term, t_2, in t_1 by C_2.

Theorem 8 *The minimal algorithm applied to an acircular congruent system, cs, returns an equivalent set of congruent equations in the minimal form.*

Proof This new algorithm terminates. Each transformation strictly diminishes the size of the system, without affecting the grounding solutions of the system. By construction and because of the previous theorem, the algorithm computes the equivalent minimal form.

Although this algorithm computes the minimal form of an acircular congruent system, it can be applied on the circular systems, because of the finite number of congruent systems whose fsize is less than that of the computed system. This means that the minimal form of any solvable system is computable.

3.5 Orthogonal systems

In order to show some invariant phenomena during the resolution and, in particular, for some recursive goals, an interesting notion is the orthogonal system.

Definition 18 *Two solvable systems of equations, E_1 and E_2, are said to be orthogonal if there exist two reduced systems, rs_1 and rs_2, respectively equivalent to E_1 and E_2, such that the union of rs_1 and rs_2 is also reduced (and equivalent to $E_1 \cup E_2$):*

$$E_1 \perp E_2 \quad \textit{if} \quad \exists\, rs_1 \equiv E_1,\ \exists\, rs_2 \equiv E_2,\ \ rs_1 \cup rs_2 \textit{ is reduced}$$

Intuitively, two systems are orthogonal if their union does not produce new information, that is, all the information has already been explicitly written in one of these systems. If the systems of equations, E_1 and E_2, are solvable and do not share any variable, then they are obviously orthogonal.

Theorem 9 *Two congruent systems, cs_1 and cs_2, are orthogonal iff the congruent form, cs, of their union exists and verifies:*

$$pfree(cs) \;=\; pfree(cs_1) \cap pfree(cs_2)$$

Obviously, this condition is equivalent to:

$$or\ \ eliminable(cs) \;=\; eliminable(cs_1) \cup eliminable(cs_2)$$

Proof Directly from the definitions of orthogonal systems, possibly free and eliminable variables

If two systems are orthogonal, a simpler unification algorithm can be used on their congruent forms and not on their reduced forms because of the ambiguity between (U =V) and (V = U).

The unification of two orthogonal congruent systems can be obtained by selection of one congruent equation and deletion of the other equation for any variable which occurs in two different classes.

Corollary 7 *Let cs_1 and cs_2 be two orthogonal congruent systems, then a congruent form of $cs_1 \cup cs_2$ can be computed by successive applications of the following transformation:*

(1) $(C_1 = t_1) \subset cs_1$, $(C_2 = t_2) \in cs_2$, $C_1 \cap C_2 \neq \emptyset$ — *delete one of these equations and merge C_1 and C_2 .*

Some variants of this algorithm can be introduced in relation to the choice of the deleted equation, for example, the deepest one. However, although cs_1 and cs_2 are minimal, the obtained system is generally not minimal.

The unification algorithm for orthogonal systems of equations is very simple. Thus, the two following operations are based on the choice of equations of the first congruent system.

Definition 19 *Let cs_1 and cs_2 be two congruent systems of equations.*

1. *$cs_1 \oplus cs_2$ is a congruent system composed from the "union" of the congruent equations of cs_1 and cs_2 and a series of the following transformations:*

   (1) $(C_1 = t_1) \in cs_1$, $(C_2 = t_2) \in cs_2$, $C_1 \cap C_2 \neq \emptyset$ — *delete the second equation and merge C_1 and C_2 everywhere.*

2. *$cs_1 \ominus cs_2$ is a smallest system such that:*

$$cs_1 = (cs_1 \ominus cs_2) \oplus cs_2$$

Proposition 9 *The operation $\oplus$ verifies some immediate properties:*

1. $cs_1 \perp cs_2 \Rightarrow cs_1 \oplus cs_2 \equiv cs_2 \oplus cs_1$
2. $cs_1 \equiv cs_2 \Rightarrow cs_1 \oplus cs_3 \equiv cs_2 \oplus cs_3$

From the previous theorem, an important consequence is also the following one:

Corollary 8 *Let ms_1 and ms_2 be two minimal systems and cs be a minimal form of $cs_1 \cup cs_2$. If ms_1 and ms_2 are not orthogonal, then the following properties are always verified:*

1. $fsize(ms) < fsize(ms_1) + fsize(ms_2)$
2. $pfree(ms) \subset\neq pfree(cs_1) \cap pfree(cs_2)$
3. $eliminable(ms) \supset\neq eliminable(cs_1) \cup eliminable(cs_2)$

This shows that if some new information is produced from the union of ms_1 and ms_2, then at least one redundancy appears in this union. Hence, the increase of the domain (or the decrease of the number of free variables) is closely linked with the decrease of function symbol occurrences.

4 Weighted Graph and Sequence of Global Rewritings

This section shows the link between weighted graphs and global rewriting rules. They are called *global rewriting rules*, because the whole term, not a part of it, is rewritten with respect to them. These rules are denoted L :- R in Prolog or $L \rightarrow R$ in term rewriting systems. L and R are finite trees. This paper uses the notation, $L \rightarrow R$, used also in Prolog II [Colmerauer 79].

Definition 20 *A term, T, is said to be globally rewritten to another term, T', with respect to* $L \rightarrow R$ *, if there exists a substitution,* θ*, such that* $\theta(L) = T$ *and* $\theta(R) = T'$.

The $L \rightarrow R$ rule is applied globally, which means that a term can be globally rewritten by the rule iff the term taken as a whole is an instantiation of the left term of the rule (see the example in the introduction).

The expressions *reduced term* and *reduction* are usually used for describing this notion, but these terms insinuate that the rewritten term is, for example, smaller or simpler than the first one within a undefined measure of complexity. Moreover, this measure generally cannot exist because any Turing machine can be simulated from the rewriting of one ground term by one rule.

Another definition could be *global rewriting with instantiation*, that is, a term, T, could be globally rewritten with instantiation to another term, T', with respect to $L \rightarrow R$, if $\sigma(T)$ were globally rewritten to T with respect to $L \rightarrow R$ where σ is the most general unifier of T and L. In term rewriting systems, this notion, rewriting with instantiation, has no meaning because the rewritten terms are ground. Obviously, term T and the $L \rightarrow R$ rule are assumed to share no variables.

4.1 Most general sequence of global rewritings using one rule

Proposition 10 *The most general sequence,* S_n*, of n global rewritings w.r.t.* $L \rightarrow R$ *is the most general solution of the following system:*

$$\{ L_i = R_{i-1} \ / \ \forall i \in [2, n] \}.$$

Let σ_n *be its most general unifier, then we will denote:*

$$S_n = \{ (1, S_n^1) , (2, S_n^2) , (n, S_n^n) , (n+1, S_n^{n+1}) \}$$

where, $\forall i \in [1, n]$, $S_n^i = \sigma_n(L_i)$
and $\forall i \in [2, n+1]$, $S_n^i = \sigma_n(R_{i-1})$

$$\Rightarrow \ S_n^1 \xrightarrow{r} S_n^2 \xrightarrow{r} \quad \cdots \quad \xrightarrow{r} S_n^n \xrightarrow{r} S_n^{n+1}.$$

Proof The variables of $L \rightarrow R$ have a local meaning which means that the variables must be renamed before applying the rule. At the i^{th} global rewriting, rule $L_i \rightarrow R_i$ is applied.

Definition 21 *A rule is said to be loop-generating if it has a most general infinite sequence of global rewritings. In others words,* $L \rightarrow R$ *is loop-generating iff it generates an infinity of rewritings for some goals.*

The most general fixpoint of a global rule is the most general term which is rewritten to an equivalent term by this rule.

Proposition 11 *Applying rule* $L \rightarrow R$ *n times is equivalent to applying* $S_n^1 \rightarrow S_n^{n+1}$ *once, that is,* $L \rightarrow R$ *can generate n rewritings for term T iff T and* S_n^1 *are unifiable.*

Similarly, $L \rightarrow R$ *can generate an infinity of rewritings for T iff T and* S_∞^1 *are unifiable.*

4.2 Weighted graph and global rewriting rule

The following theorem gives a fundamental justification of this new syntactic object.

Theorem 10 *Let $L \rightarrow R$ be a global rewriting rule. Let us denote L^0 and R^{-1} as the weighted graphs built from L and R by putting null weights on the arrows, no period and a root weight equal to 0 or -1, then*

$$S_n \;=\; I^{[1,n+1]}_{[1,n]}(L^0) \;\bigvee\; I^{[1,n+1]}_{[1,n]}(R^{-1}).$$

Proof For all i of [1,n], L_i is the unfolding of L^0 from the input weight, i, and in [1,n]. Similarly, for all i of [2,n+1], R_{i-1} is the unfolded result of R^{-1} from the input weight, i, and in [1,n].

$$\forall i \in [1,n]\ ,\quad L_i = U^i_{[1,n]}(L^0) \quad \text{and} \quad \forall i \in [2,n+1]\ ,\quad R_{i-1} = U^i_{[1,n]}(R^{-1})$$

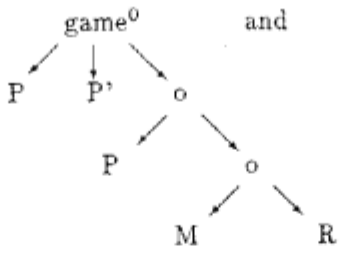
Thus, the following unification characterises the most general sequence of n global rewritings:

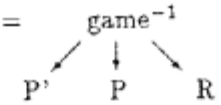
$$S_n \;=\; I^{[1,n]}_{[1,n]}(L^0) \;\bigvee\; I^{[2,n+1]}_{[1,n]}(R^{-1}) \;=\; I^{[1,n+1]}_{[1,n]}(L^0) \;\bigvee\; I^{[1,n+1]}_{[1,n]}(R^{-1})$$

Example Let the chess game rule be:

$$game(P, P', P \cdot M \cdot R) \;\rightarrow\; game(P', P, R).$$

The first argument is the name of the player who has to play, the second is the name of the player who will have to play at the next turn, and the third is the list of the name of the players and their moves.

$L^0 =$ 

and $R^{-1} =$ 

Theorem 11 *: Let $L \rightarrow R$ be a rule. Then there exists a constant, n_0, such that if R can generate a sequence of n_0 rewritings then the finite weighted graph, $L^0 \vee R^{-1}$, exists and there are four natural integers, a_1, a_2, b_1 and b_2, such that:*

$$I^{[1,n+1]}_{[1+a_1,n-b_1]}\,(L^0 \vee R^{-1}) \;\leq\; S_n \;\leq\; I^{[1,n+1]}_{[1-a_2,n+b_2]}\,(L^0 \vee R^{-1}).$$

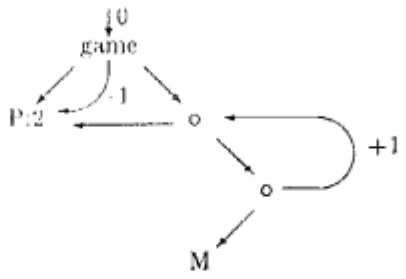
Proof Immediate application of fundamental theorem 3 .

Hence, the behaviour of $L \rightarrow R$ is characterised by the weighted graph, $L^0 \vee R^{-1}$, that is, this weighted graph can be viewed as a meta-term of the rule.

Because of this theorem, all properties of weighted graphs can now be applied within logic programming for a better understanding of recursive behaviour. The following theorems illustrate this.

The finite weighted graph of the chess game rule has already been computed:

$L^0 \vee R^{-1} =$



The periodicity of the first and second arguments is expressed by the period of P and the third argument is characterised by the third sub-graph that introduces the period notion (cf. 2.1.2).

The unfolding of this weighted graph, from k and in [0,n], is the k^{th} term of the most general sequence of n global rewritings, that is, the state of the chess game of n moves after the first (k-1) moves.

Remark 6 *If the terms, L and R, are not unifiable within the directed graphs, then L^0 and R^{-1} are not unifiable within the weighted graphs.*
Unifiability (without occur-check) is a necessary condition for the existence of the weighted graph, $L^0 \vee R^{-1}$.

Example The rule, *put(milk) :- put(coffee)*, has no characteristic weighted graph, because *put(milk)* and *put(coffee)* are not unifiable.

4.3 Infinite sequence of global rewritings and most general fixpoint

Theorem 12 *A rule is loop-generating, that is, it can generate an infinite sequence of global rewritings iff its weighted graph exists.*

Proof By application of theorem 11

Conjecture 1 *If its weighted graph does not exist, the length of the longest sequence of global rewritings is less than or equal to 2^n where n is the number of variables of the rule.*

$$\text{e.g.} \quad f(f(\cdots f(g,U),V)\cdots,W),X) \;\rightarrow\; f(X,f(W,\cdots f(V,f(U,h))\cdots))$$

This whole study is based on unification with occur-check, but a similar theorem can be easily proven within unification without occur-check.

Theorem 13 *A rule, $L \rightarrow R$, can generate a non-bounded sequence of global rewritings without occur-check iff L and R are unifiable.*

Proof

1. If L and R are unifiable, it is obvious that $L \vee R$ is a fixpoint of this rule, which means that it can be globally rewritten infinitely using rule $L \rightarrow R$.
2. If there is an infinite sequence of global rewritings, then (S_n^∞) is an infinite increasing sequence and its limit, S_∞^∞, is the most general fixpoint of the rule. This means that:
   (a) there exists a substitution, θ, such that $\theta(L) = S_\infty^\infty$
   (b) $\theta(R)$ and $\theta(L)$ are equivalent (that is, equal to using renaming)

Let us consider a grounding substitution, ρ, which instantiates all the variables with the same constant, then:

$$\rho(\theta(L)) = \rho(\theta(R)) \quad \Longrightarrow \quad \text{L and R are unifiable without occur-check.}$$

This unifier, $\rho(\theta(L))$, is generally infinite, so the unifier exists without occur-check.

Theorem 14 *The most general fixpoint of a loop-generating rule, $L \rightarrow R$, is expressed by the unfolded result of its weighted graph from any input weight without control (CR = Z):*

$$\textit{Most general fixpoint of } L \rightarrow R \quad = \quad U_Z^0\,(L^0 \vee R^{-1})$$

Proof The most general fixpoint of the rule is the most general tree which is globally rewritten in an equivalent form by the rule. Thus, S_∞^∞ is the most general fixpoint of the rule iff it expresses the most general bi-infinite sequence of global rewritings:

$$\cdots \theta_{-(n+1)}(S_\infty^\infty) \xrightarrow{r} \theta_{-n}(S_\infty^\infty) \;\cdots\; \theta_0(S_\infty^\infty) \xrightarrow{r} \;\cdots \xrightarrow{r} \theta_n(S_\infty^\infty) \xrightarrow{r} \theta_{n+1}(S_\infty^\infty) \;\cdots$$

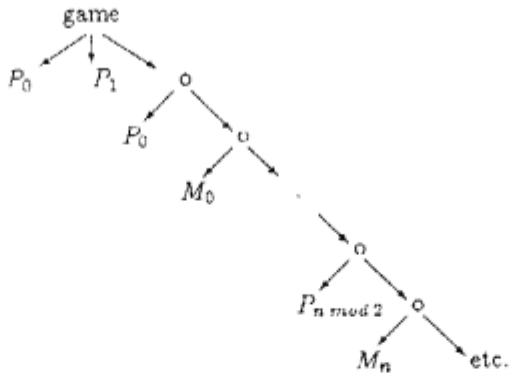
where θ_k are renaming substitutions. This corresponds to the system of equations:

$$\{\; L_i = R_{i-1} \;/\; \forall k \in Z \;\}$$

that is, the unification of L^0 and R^{-1} on Z:

$$I_Z^Z(L^0) \;\vee\; I_Z^Z(R^{-1}) \quad = \quad I_Z^Z(L^0 \vee R^{-1})$$

Example The most general fixpoint of the rule, *Chess game*, is the unfolding of the weighted graph, that is, the most general infinite chess game:



We can see in this example that the generalisation of the directed graph by putting weights and periods is necessary for expressing the most general fixpoint of this rule. Moreover, this generalisation has been proven sufficient.

Hence, this new object is perfectly adapted to the understanding of the basic recursivity.

Theorem 15 *For any loop-generating rule, the depth of terms S_n^1 is bounded iff its weighted graph has no positive basic loop.*
For any loop-generating rule, the depth of terms S_n^{n+1} is bounded iff its weighted graph has no negative basic loop.

Proof The depth of terms S_n^1 is bounded iff their limit, $S_{+\infty}^1$ does not exist or is finite. In the first case, the weighted graph does not exist, and, moreover, $S_{+\infty}^1$ is finite iff the weighted graph of rule has no positive basic loop:

1. $S_{+\infty}^1$ is finite iff for any k, $S_{+\infty}^k$ is also finite:
   (a) if $S_{+\infty}^1$ is finite, after k global rewritings, the term, $S_{+\infty}^k$, is also finite because of unification with occur-check.
   (b) $S_{+\infty}^1$ is the most general tree which can be globally rewritten infinitely by the rule, but $S_{+\infty}^k$ can also be globally rewritten infinitely. Thus, $S_{+\infty}^1 \leq S_{+\infty}^k$.
   This means that if $S_{+\infty}^k$ is finite, then $S_{+\infty}^1$ is also finite.
2. There exists k such that $S_{+\infty}^k$ is infinite iff the weighted graph of rule contains positive basic loops.
   By application of theorem 1, there exists k such that $S_{+\infty}^k$ is infinite iff there is an input weight k such that $U_{CR_{sub}}^k(L^0 \vee R^{-1})$ is infinite.
   It is easy to verify the equivalence to the existence of positive basic loops in the weighted graph of the rule.

Similarly, by reversing the rule, the limit of terms, S_n^{n+1}, is the smallest tree which can be globally rewritten infinitely using rule $R \rightarrow L$. Moreover, the weighted graph, $L^0 \vee R^{-1}$, of rule $L \rightarrow R$ and the weighted graph, $R^0 \vee L^{-1}$, (or rather $R^1 \vee L^0$) of the reverse rule are quite similar. Only the signs of the weights are opposite.

4.4 Finite sequence of global rewritings

The range of this sequence using a weighted graph interpreted from two counter range intervals defined from theorem 11 expresses the side effects of the rule.

4.4.1 Finite sequence = Finite weighted graph + Side effects

Remark 7 *There are side effects even within this particular use of the weighted graph. There exist rules for which there is no weighted graph h such that:*

$$\forall n \text{ sufficiently great.} \qquad S_n \supset I_{[a,n+b]}^{[1,n+1]}(h)$$

where a, b are constants belonging to Z.

Example The most general sequence of n global rewritings of $f(U,U) \rightarrow f(V,W)$ is:

$$f(U_1,U_1) \xrightarrow{r} f(U_2,U_2) \quad \cdots \quad \xrightarrow{r} f(U_n,U_n) \xrightarrow{r} f(V_n,W_n)$$

A more complex example is: $f(U,U,g(V,g(W,X)),Y) \rightarrow f(V,W,X,g(U,Y))$.
All the variables appear in both parts of this rule, and the side effects appear in all the terms of the sequence.

Conjecture 2 *If the terms of the rule are linear, there exists a weighted graph, wg, such that:*

$$\forall n \text{ sufficiently great,} \qquad S_n \supset I_{[a,n+b]}^{n}(wg)$$

where a and b are constants belonging to Z.

The last remark shows that the semantic power of a weighted graph is not enough for expressing precisely a finite sequence of global rewritings. There are side effects whose size is known and bounded. The principle of the next proofs is based on the fact that information about these side effects can be computed finitely.

4.4.2 Finite sequence and congruent system through weighted graphs

Definition 22 *The set of symbols of variables, EV (eliminable variables), is the set of symbols occurring in L^0 or R^{-1}, which have been substituted by a function symbol in the unification of L^0 and R^{-1}.*

Example In the weighted graph associated with the chess game rule:
$EV = \{ R \}$ because R has been substituted by the function symbol, o, during the unification.

Proposition 12 *Let cs_n be the congruent form of $\{L_i = R_{i-1} \ / \ \forall i \in [2,n] \}$, and let p be the period of the weighted graph, $L^0 \vee R^{-1}$, then there exist two natural integers, l_{se} (left side effect) and r_{se} (right side effect), such that:*

$$EV \times [1 + l_{se}, n - r_{se}] \ \subset \ eliminable(cs_n) \ \subset \ EV \times [1, n]$$

Proof Using theorem 11, comparing L_i, R_{i+1} and $U^i_{sub(n)}(L^0 \vee R^{-1})$, apart from the side effects, it is easy to see if any indexed variables in L_i have been substituted by a function symbol in the weighted graph.

This means that the behaviour of any eliminable indexed variable in the congruent system, cs_n, is well-known except for the indices belonging to the side effects.

Notation 4 *Let t be a term built from F and $Var \times Z$. We will denote $t^{\rightarrow i}$ the term t all of whose indices are incremented by i, a relative integer:*

$$t = f(X_p, g, Y_q) \quad \Rightarrow \quad t^{\rightarrow i} = f(X_{p+i}, g, Y_{q+i})$$

Similarly, let cs be a congruent system of equations, then $cs^{\rightarrow i}$ is the congruent system, cs, in which all the indices of variables are incremented by i.

The following theorem expresses that the recursivity of one global rewriting rule can be characterised by an iterative transformation:

Theorem 16 *For any loop-generating rule, there exists a constant, n_0, such that from n_0, the congruent system, cs_n, can be iteratively expressed:*

$$\forall n \geq n_0, \ cs_{n+1} \ = \ \alpha \oplus (cs_n)^{\rightarrow 1} \ = \ cs_n \oplus \omega^{\rightarrow (n+1)}$$

where α and ω are two constant congruent systems.

Proof

1. cs_{n+1} is a congruent form of $cs_n \cup (cs_n)^{\rightarrow 1}$:

$$\begin{aligned} cs_{n+1} &\equiv \{ L_i = R_{i-1} \ / \ \forall i \in [2, n+1] \} \\ &\equiv \{ L_i = R_{i-1} \ / \ \forall i \in [2, n] \} \ \cup \ \{ L_i = R_{i-1} \ / \ \forall i \in [3, n+1] \} \\ &\equiv cs_n \ \cup (cs_n)^{\cdot 1} \end{aligned}$$

2. There exists a constant, n_1, such that for all n greater or equal than n_1, the congruent systems, cs_n and $(cs_n)^{-1}$, are orthogonal:

$$\exists \, n_1, \ \forall n \geq n_1, \ \ cs_n \perp (cs_n)^{-1}.$$

From theorem 9, it is equivalent to show that:

$$\exists \, n_1, \ \forall n \geq n_1, \ \ eliminable(cs_{n+1}) \ = \ eliminable(cs_n) \ \cup \ eliminable((cs_n)^{-1}).$$

Using proposition 12, if a new eliminable variable appears in cs_{n+1}, then the index of this new variable must belong to the side effects, $[1, l_{se}] \cup [n - r_{se} + 1, n]$. The size of these side effects is bounded; therefore, this phenomenon can appear a finite number of times, that is, there exists a constant, n_1, such that for all n greater than n_1, cs_n and $(cs_n)^{-1}$, are orthogonal. We may assume that there is no hole in the indices of a eliminable symbol, that is:

$$i \leq j \ , \ V_i, V_j \ \in \ eliminable(cs_n) \quad \Rightarrow \quad \forall \, k \ \in [i, j] \ , \ V_k \ \in \ eliminable(cs_n)$$

Obviously, this is true for all n greater than $n_0 = 2n_1$ because of

$$cs_{2n_1} \ \equiv \ cs_{n_1} \cup (cs_{n_1})^{-n_1}$$

3. Let α be equal to $cs_{n_0} \ominus (cs_{n_0})^{-1}$ then from the definition of the operation $\ominus$, and from proposition 12, it is easy to show that:

$$eliminable(\alpha) \ \subset \ Var \times [1, l_{se}]$$

that is, α expresses the left side effects in the most general sequence of n rewritings.
It is also easy to see that the the indices of the eliminable variables in α, are the lowest indices, and, therefore:

$$\forall n \geq n_0, \ eliminable(\alpha) \ \cap \ eliminable((cs_n)^{-1}) \ = \ \emptyset$$

that is, α and $(cs_n)^{-1}$ are orthogonal.

$$\begin{aligned} & cs_{n_0+1} = \alpha \ \cup \ (cs_n)^{-1} && \text{(by definition)} \\ \Rightarrow \quad & cs_{n_0+2} = cs_{n_0+1} \ \oplus \ (cs_{n_0+1})^{-1} \\ & = \alpha \ \oplus \ (cs_{n_0})^{-1} \ \oplus \ (cs_{n_0+1})^{-1} \\ & = \alpha \ \cup \ (cs_{n_0+1})^{-1} \\ & = \alpha \ \oplus \ (cs_{n_0+1})^{-1} && (\alpha \perp (cs_{n_0+1})^{-1}) \end{aligned}$$

In a similar way, this is true for all n greater than n_0:

$$\forall n \geq n_0 \ . \ cs_{n+1} \ = \ \alpha \ \cup \ (cs_n)^{-1}$$

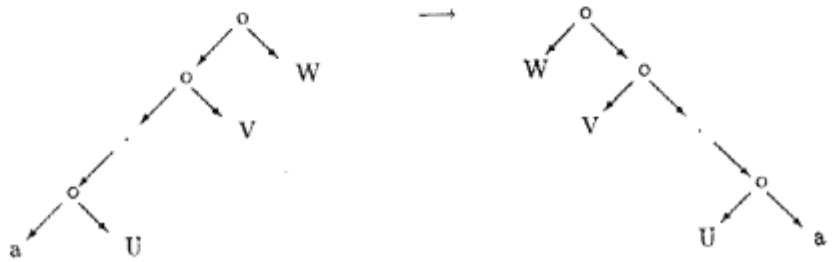
4. Let ω be equal to $(cs_{n_0})^{\rightarrow(-n_0)} \ominus (cs_{n_0})^{\rightarrow(-n_0+1)}$ then we can prove in the same way:

$$\forall\, n \geq n_0\ , \quad cs_{n+1} = cs_n \cup \omega^{\rightarrow(n+1)}$$

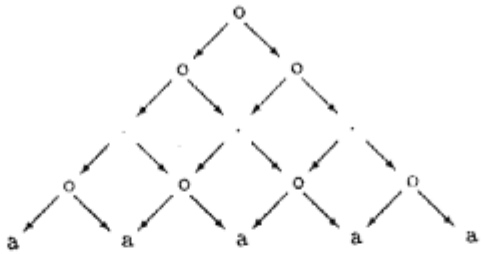
The congruent system, ω, expresses the right side effect of the most general sequence of n rewritings, ad its domain is included in $Var \times]n - r_{se}, n]$

Remark 8 *An equivalent form of the theorem is:* $\forall n \geq n_0$
$cs_n = \alpha \oplus \alpha^{\rightarrow 1} \oplus \cdots \oplus \alpha^{\rightarrow(n-n_0-1)} \oplus (cs_{n_0})^{\rightarrow(n-n_0)}$
$cs_n = cs_{n_0} \oplus \omega^{\rightarrow(n_0+1)} \oplus \omega^{\rightarrow(n_0+2)} \oplus \cdots \oplus \omega^{\rightarrow n})$

Remark 9 *For the usual recursive rules, the size of the side effects is nearly zero, but for some rules, this size is an exponential function of the number of variables of the rule:*



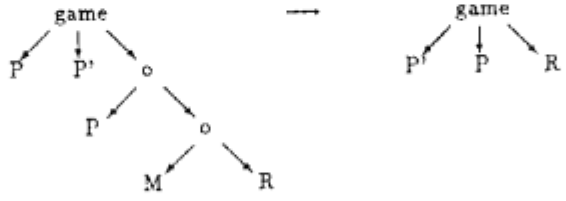
If this rule contains M variables (U, V, ..., W), then 2^M global rewritings must be applied before detecting a constant phenomenon. In this case, this rule becomes completely constant, that is that all the terms of the sequence S_n for n greater than 2^M are equal to the following tree:



Conjecture 3 *Let M be the number of variables of the rule or the depth of the rule terms, then the constant n_0 is less than or equal to 2^M:*

$$M = Card(range(L) \cup range(R)) \text{ or } sup(depth(L), depth(R)) \quad \Rightarrow \quad n_0 \leq 2^M$$

Example The chess game is characterised by the following rule:



A sequence of n recursive applications of this rule is characterised by the following system of equations:

$$E_n \;=\; \{\; game(P_i,\; P'_i,\; o(P_i, o(M_i, R_i))\;) \;=\; game(P'_{i-1},\; P_{i-1},\; R_{i-1}) \;/\; \forall i \;\in\; [2,n]\;\}$$

Its minimal form is:

$$ms_n \;=\; \{\; \vec{R}_{i-1} \;=\; o(\vec{P}_i,\; o(\vec{M}_i,\; \vec{R}_i)) \;/\; \forall\; i \;\in [2,n]\;\}$$

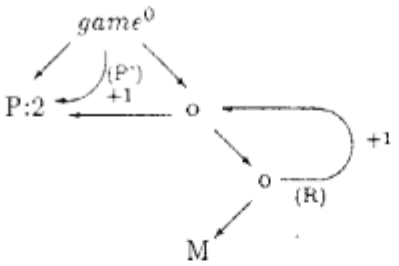
where the congruence, $\Re_{ms_n}$, is composed of:

1. $\forall i \in [1,n],\ \vec{R}_i = \{R_i\}$,
2. $\forall i \in [1,n],\ \vec{M}_i = \{M_i\}$,
3. $\forall i \in [1,n],\ \vec{P}_i = \{P_j \;/\; \forall j \in [1,n],\ j =_{mod\,2} i\} \;\cup\; \{P'_k \;/\; \forall k \in [1,n],\ k =_{mod\,2} i+1\}$
4. $\forall i \in [1,n],\ \vec{P'}_i = \{P'_j \;/\; \forall j \in [1,n],\ j =_{mod\,2} i\} \;\cup\; \{P_k \;/\; \forall k \in [1,n],\ k =_{mod\,2} i+1\}$

In this example, the constant n_0 is equal to 1, that is, there are no side effects:
- $ms_1 = \emptyset$
- $ms_2 \;=\; \alpha \;=\; (\omega)^{-2} \;=\; \{\; \vec{R}_1 = o(\vec{P}_2, o(\vec{M}_2, \vec{R}_2))\;\}$

and $\vec{R}_1 \;=\; \{R_1\},\ \vec{P}_2 \;=\; \{P'_1, P_2\},\ \vec{P'}_2 \;=\; \{P_1, P'_2\}$,
- $ms_{n+1} \;=\; \alpha \;\oplus\; (ms_n)^{\to 1} \;=\; ms_n \;\oplus\; (\omega)^{-(n+1)}$

This can be checked in the weighted graph of the rule:



4.4.3 Behaviour of the first and last terms of a finite sequence

Definition 23 *The lowest common multiple of the periods of the weighted graph is said to be the period of this weighted graph.*

Theorem 17 *The period of the rule is the period of its weighted graph and the growing branches of the terms S_n^1 and S_n^{n+1} increase linearly.*
Let p be the period of the weighted graph. There exists a function, f, linear or constant, such that:

$$\forall m \in Dom(S_n^1),\ \forall m' \in Dom(S_n^{n+1}),\ \forall n \geq f(sup(\mid m \mid, \mid m' \mid)),$$
$$S_n^1(m) \;=\; S_n^{n+1}(m) \quad \Longleftrightarrow \quad S_{n+p}^1(m) \;=\; S_{n+p}^{n+p+1}(m)$$

Lemma 11 *The lengths of the growing branches in S_n^1 and S_n^{n+1} increase linearly. There exists a function, f, linear or constant, such that:*

$$\forall m \in Dom(lim(S_n^1)),\ \forall n \geq f(\mid m \mid),\ \ m \in Dom(S_n^1)$$

$$\forall m' \in Dom(lim(S_n^{n+1})),\ \forall n \geq f(\mid m' \mid),\ \ m' \in Dom(S_n^{n+1})$$

Proof Using theorem 16, the term $S^1_{n+r_{se}}$ can be defined as the term S_n^1 modified by the new constraints, $\omega^{\Rightarrow(n+1)} \oplus \cdots \oplus \omega^{\rightarrow(n+r_{se})}$, knowing the previous set of constraints, cs_n.

Thus, an indexed variable of S_n^1 will be substituted in $S^1_{n+r_{se}}$ by a function symbol iff its index belongs to the right side effect ($[n, n+r_{se}]$) and its symbol of variable belongs to EV.

Therefore, the depth of the growing branches of S_n^1 increases at least linearly by 1 before r_{se} new rewritings.
Symmetrically, $S^{n+1+l_{se}}_{n+l_{se}}$ can be defined as $(S_n^{n+1})^{\rightarrow 1}$ modified by the new constraints, $\alpha \oplus \cdots \oplus \alpha^{\rightarrow l_{se}}$ knowing the previous constraints, $(cs_n)^{\rightarrow l_{se}}$.

Thus, a weak and lazy evaluation of this increase is:

$$depth_n \ \geq \ (n - n_0)/max(l_{se}, r_{se})$$

Moreover, we know (theorem 2) that the depth of the weighted graph interpretation is also bounded by a linear function of n; therefore, from theorem 11, the depth of the terms of a finite sequence is bounded by a linear function.

If the weighted graph is acyclic, the depth of the terms, S_n^1 and S_n^{n+1}, is bounded, that is, function f is constant.

Proof of the theorem Let m, m' and n be such that:

$$m \in Dom(S_n^1),\ m' \in Dom(S_n^{n+1}),\ n \geq f(sup(\mid m \mid, \mid m' \mid)),$$

then if the labels of the paths, m and m', are function symbols, it is true from the proof of the previous lemma:

$$\forall n, n' \geq f(\mid m \mid),\ \ S_n^1(m) \in F \ \ \Rightarrow \ \ S_{n'}^1(m) \in F$$

$$\forall n, n' \geq f(\mid m \mid),\ \ S_n^{n+1}(m) \in F \ \ \Rightarrow \ \ S_{n'}^{n'+1}(m) \in F$$

Moreover, $(S_n^1)_{n\geq 1}$ and $(S_n^{n+1})_{n\geq 1}$ are increasing sequences, that is, if $S_n^1(m)$ is a function symbol, for any n' greater than n, $S_n^1(m)$ and $S_n^{n+1}(m)$ are equal.

Thus, let us consider that the labels are indexed variables, but are never substituted by function symbols for longer sequences of rewritings.

Then, for n that is sufficiently great (linearly depending on the lenghts of m and m'), these variables can be shown as periodic, and their indices can be chosen such that they do not belong to the side effects.

Looking at the range of the finite sequence (theorem 11), we know the behaviour of the periodic variables (symbols of variables which are periodic or substituted by a periodic symbol in the weighted graph). The periods correspond to the periods in the weighted graph of the rule; therefore, the periodic equation will be true for the lowest common multiple of the periods of this weighted graph, that is, what is called the period of the rule.

5 Application within the Terms Rewriting Systems, Narrowing and Logic Programming

In this section, some open problems are solved using the characterisation of a recursive rule through its weighted graph.

5.1 Global rewriting systems

Proposition 13 *A ground term, T, can be rewritten n times by the rule, $L \rightarrow R$, iff S_n^1 and T are unifiable.*

Theorem 18 *The uniform termination of one global rewriting rule is decidable.*

Rule r generates finite global rewritings for all finite ground terms iff the weighted graph of the rule does not exist or contains positive basic loops.

Proof Obviously, if L^0 and R^{-1} are not unifiable, that is, the weighted graph of the rule does not exist, then uniform termination is verified (cf. theorem 11)
In other cases, rule $L \rightarrow R$ verifies the uniform termination iff the term $S_{+\infty}^1$ is infinite. From proposition 8, this is equivalent to the existence of positive basic loops in the weighted graph.

Remark 10 *This property is undecidable within rewriting systems. One rule is enough to simulate any Turing machine [Dauchet 87].*

Theorem 19 *The uniform termination of one global rewriting rule and <u>one</u> finite ground term is decidable. There exists a function, f, either linear if the weighted graph contains positive loops, or constant otherwise, such that a finite ground term, T, is globally rewritten finitely iff it can be rewritten more than $f(depth(T))$.*

Proof Based on theorems 2 and 15

5.2 Global narrowing

The only difference between global rewriting systems and global narrowing is that the term which is globally rewritten is not ground, but it may contain some variables.

Definition 24 *A term, T, is said to be globally rewritten with instantiation iff there exist a substitution σ and a term T' such that $\sigma(T)$ is globally rewritten to T'.*

Proposition 14 *A finite term, T, can be rewritten with instantiation n times by the rule, $L \rightarrow R$, iff S_n^1 and T are unifiable.*

Theorem 20 *Within global narrowing, the uniform termination of one rule $L \rightarrow R$ is decidable. For all terms, the rule generates a finite sequence of global rewritings with instantiation iff its weighted graph ($L^0 \vee R^{-1}$) does not exist.*

Proof Obviously, the rule generates a finite sequence of global rewritings for any term iff that is true also for the most general term expressed by any variable. This is equivalent to the existence of a bounded sequence of global rewritings (theorem 12) .

Remark 11 *Here, the problem of the uniform termination is linked to one rule and one term. That is, rule r and one term, T, verify the uniform termination iff the term, T, is finitely globally rewritten with instantiation wrt rule r.*

Theorem 21 *The uniform termination of one rule $L \rightarrow R$ and one finite linear term, T, is decidable.*
There exists a function, f, either linear if the weighted graph contains positive loops, or constant otherwise, such that a finite ground term, T, is globally rewritten finitely iff it can be rewritten more than $f(depth(T))$.

Proof of the theorem If the term, T, is linear (that is, there is one occurrence of each variable), then the proof of the theorem is simple. From theorem 17, for n greater than $f(depth(T))$ (positive loops) or a constant (no positive loops), there exists a constant substitution, σ_G, such that $S_n^1 \vee G = \sigma_G(S_n^1)$.

Example The recursive definition of the natural integers: $succ(U) \rightarrow U$.
The term, $succ^n(zero)$, is globally rewritten at most n times, but the term, $succ^n(V)$ can be infinitely globally rewritten by instantiation of the variable, V.

This theorem is still a conjecture in the non-linear case.

5.3 Logic programming

Let us consider the following structure of Prolog programs, called Prolog While because of its behaviour:

$$while(t_{end}) .$$
$$while(t_{before}) :- while(t_{after}) .$$

Theorem 22 *The SLD tree of a Prolog While program is finite for all goals, $while(t_{begin})$, iff the weighted graph of the rule does not exist.*

Proof This is equivalent to the theorem about uniform termination within global narrowing.

Theorem 23 *The SLD tree of a Prolog While program is finite for all ground goals, $while(t_{begin})$, iff the weighted graph of the rule does not exist or contains positive loops.*

Proof This is equivalent to the theorem about uniform termination within global rewriting.

Theorem 24 *The termination of the SLD resolution of one linear goal is decidable.*
There exists a function, f, linear (weighted graph exists and contains positive loops), constant (otherwise), such that the SLD resolution of a Prolog While program and a linear goal is finite iff there is no sequence of global rewritings whose length is greater than or equal to $f(depth(goal))$.

Proof This is equivalent to theorem 20 within global narrowing.

Theorem 25 *In the case of a Prolog While program whose fact is linear, the existence of solutions for a linear goal is decidable.*
There exists a function, f, linear (weighted graph exists and contains positive loops), constant (otherwise), such that the SLD resolution of a Prolog While program and a linear goal generate solutions iff some of them are obtained before $f(depth(goal))+period(rule)$ recursive rewritings.

Proof The proof is based on the same idea as the proof of theorem 21 within global rewriting systems.
Let us suppose that the fact and the goal are linear, (that is, there is one occurrence of each variable). Because the terms, S_n^1 and S_n^{n+1}, increase linearly according to n, then for all n greater than $f_{max}(depth(goal))$ and $f_{max}(depth(fact))$, all the branches smaller in S_n^1 than in G are constant and all the branches smaller in S_n^{n+1} are constant (in relation to n) or periodic (cf Theorem 17).

After a number of rewritings linearly depending on the depth of the goal, there exists a constant substitution, σ_G, such that $S_n^1 \vee Goal = \sigma_G(S_n^1)$ because the goal is linear.

Let p be the period of the rule. There exist a substitution constant in relation to n, denoted σ_f^{-n}, and a finite number of constant substitutions, denoted $\sigma_{f_n \bmod p}$ such that:

$$S_n^{n+1} \vee Fact = (\sigma_{f_n \bmod p} \cup \sigma_f^{-n})(S_n^{n+1})$$

where $Dom(\sigma_{f_n \bmod p}) \cap Dom(\sigma_f^{-n})$ is empty.

A solution exists for n recursive rewritings iff:
$S_n \cup (Goal = S_n^1) \cup (Fact = S_n^{n+1})$ is solvable.
For n greater than $f_{max}(depth(goal))$ and $f_{max}(depth(fact))$, that is equivalent to σ_G, $\sigma_{f_n \bmod p}$ and $(\sigma_f)^{-n}$ are unifiable.

Therefore, if there is no solution for a number of recursive belonging to

$$[\ f_{max}(depth(goal)) + f_{max}(depth(fact))\ .\ f_{max}(depth(goal)) + f_{max}(depth(fact)) + period(rule)\ [$$

there will be no solutions for longer sequences of recursive rewritings.

Otherwise, if there are solutions, the number of solution paths in the SLD tree is infinite.

Theorem 26 *In the case of a Prolog While program whose fact is linear, the existence of finite number of solution paths is decidable for a linear goal. There exists a function, f, linear (weighted graph exists and contains positive loops), constant (otherwise), such that the SLD tree of a Prolog While program and a linear goal has a finite number of solution paths iff some of them are obtained for a number of recursive rewritings belonging to $[f(depth(goal))\ .\ f(depth(goal)) + period(rule)[$.*

Proof Included in the proof of the previous theorem

Definition 25 *A Prolog program is said to be bounded if it is possible to eliminate recursion from the program.*

Theorem 27 *The boundedness property is decidable for a Prolog While program whose fact is linear.*

Proof The problem is equivalent to the decidability of the existence of a finite number of solutions of the following program:

$$
\begin{array}{l}
while(t_{end}) \ . \\
while(t_{before}) \ :- \ while(t_{after}) \ . \\
:- \ while(X) \ .
\end{array}
$$

This problem is decidable if the fact and goal are linear.

Theorem 28 *A Prolog While program is bounded if one of the following properties is true:*

1. *It has no weighted graph ($L^0 \vee R^{-1}$ does not exist).*
2. *The fact is ground and the weighted graph has at least one negative loop.*
3. *The fact is linear and the weighted graph has no positive loop.*
4. *The weighted graph is acyclic.*

Proof :
(1) is equivalent to theorem 20 within global narrowing.
(2) is the symmetric result in relation to theorem 18:
If the weighted graph of the rule contains negative loops, then the depth of the terms S_n^{n+1} is not bounded, that is, after a constant number of recursive rewritings, the depth of S_n^{n+1} will be greater than the depth of the ground fact. Therefore, no solutions can be found after this constant number of rewritings.
(3) By extending the previous proof (theorem 25), after a number of rewritings which depends linearly on the depth of Goal, the solutions are:

$$(\sigma_G \vee \sigma_{f_n \ modp} \vee (\sigma_f)^{-n})\ (\ S_n^1\ )$$

Moreover, if the weighted graph contains no positive loop, then after a constant number of rewritings, S_n^1 is constant.
(4) If the weighted graph is acyclic, then the size of the terms, S_n^1 and S_n^{n+1}, is bounded. Because there exists a finite number of pair (S_n^1, S_n^{n+1}), recursively applying rule L :- R is equivalent to applying once a finite number of rules, $S_n^1 \ :- \ S_n^{n+1}$.

Remark 12 *In condition (3), the linear hypothesis about the fact is important. Let us look at the following program, whose fact is nonlinear:*

$$
\begin{array}{l}
integer(X, X) \ . \\
integer(X, Y) \ :- \ integer(succ(X), Y) \ . \\
:- \ integer(zero, Y) \ .
\end{array}
$$

This program has an infinite number of solutions (all natural integers), but its weighted graph contains no positive loop.

In a few words and using an array, we can express, simply, some of the properties and the link between a Prolog While program and its weighted graph:

Weighted graph unification

$t_{before} \vee t_{after}$ does not exist	$\Longrightarrow$	$t^0_{before} \vee t^{-1}_{after}$ does not exist
No null loops in the weighted graph	==	Occur-check verified

Weighted graph		Prolog While
$t^0_{before} \vee t^{-1}_{after}$	$\equiv$	$while(t_{end})$. $while(t_{before})$:- $while(t_{after})$. :- $while(t_{begin})$.
Weighted graph exists	$\Longleftrightarrow$	Infinite SLD resolution for some goals
Infinite unfolding	==	Most general fixpoint
Period of the weighted graph	==	Period of the Prolog While program
Positive loops + Ground goal	$\Longrightarrow$	Finite SLD resolution
Negative loops + Ground fact	$\Longrightarrow$	Boundedness ($\exists$ equivalent nonrecursive program)
Acyclic weighted graph	$\Longrightarrow$	Boundedness
Linear fact and linear goal + Positive loops	$\Longrightarrow$	$\exists$ f, linear, such that, in the SLD resolution termination, existence of solutions, existence of a finity of solution paths can be checked after f(depth(goal)) rewritings
Linear fact and linear goal + No positive loops	$\Longrightarrow$	in the SLD resolution, termination, existence of solutions, existence of a finity of solution paths can be checked after a constant number of rewritings

6 Examples within Logic Programming

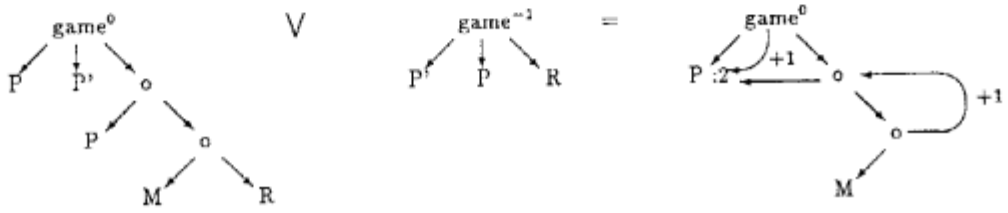
6.1 Chess game

This is an abstract of the properties proven along this paper about the chess game rule.
A chess game is characterised by the following Prolog While program:

```
game(P, P', Nil) .
game(P, P', P ∘ M ∘ R) :- game(P', P, R) .
```

The fact means that a chess game between two players, P and P', may be empty. The recursive rule expresses that if R is a chess game between the first player, P', and the second player, P, then the chess game whose two first elements are P and M and the rest is R is a chess game whose first player is P and second is P'.

The **behaviour of this rule** is characterised by the **unification of two weighted graphs**: its left term whose root is weighted by 0 and its right term whose root is weighted by -1 (theorem 11):

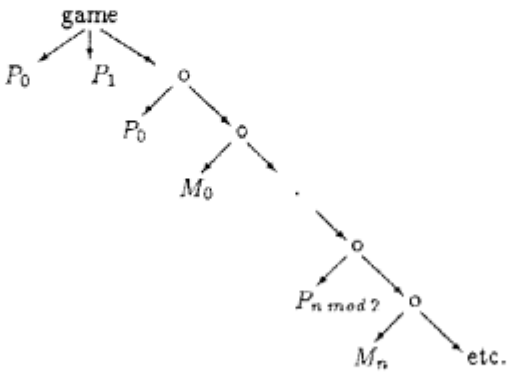


The SLD resolution is infinite for some goals because its characteristic **weighted graph exists** (theorem 22).

The following program will generate an infinite SLD computation (with an infinity of solutions):

```
game(P, P', Nil) .
game(P, P', P ∘ M ∘ R) :- game(P', P, R) .
:- game(kasparov, karpov, Z) .
```

The **most general fixpoint of the rule**, that is, the most general term which is rewritten by the rule to an equivalent term, is the **infinite unfolding of the characteristic weighted graph** (theorem 14):



This term represents an infinite chess game.

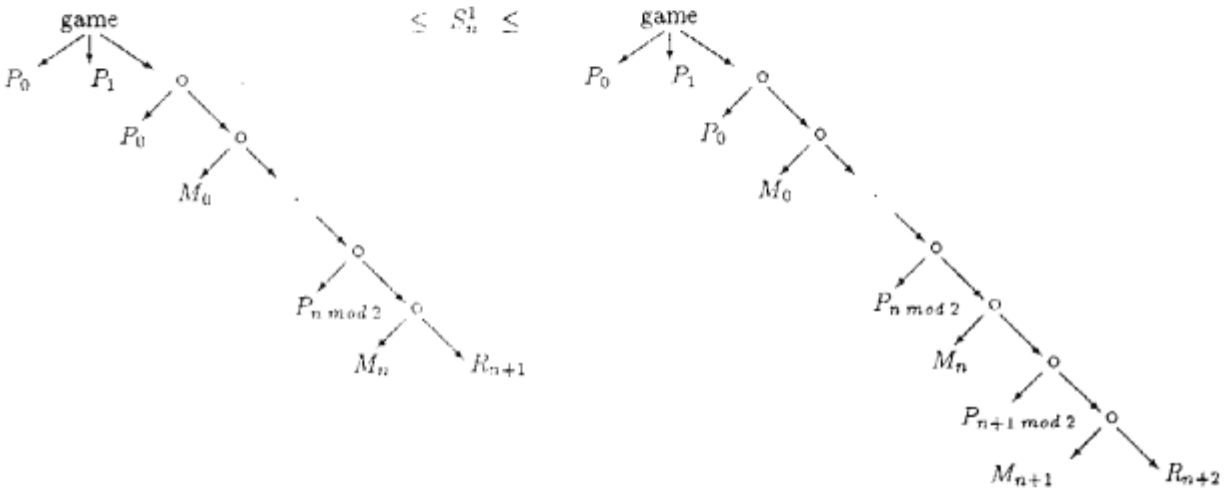
The **most general finite sequence of recursive rewritings**, S_n, is ranged across by the **finite interpretation of the weighted graph** of the rule from two intervals approximately equal to [1,n] (fundamental theorem 11):

$$I^{[1,n+1]}_{[1+a_1,n-b_1]}(L^0 \vee R^{-1}) \leq S_n \leq I^{[1,n+1]}_{[1-a_2,n+b_2]}(L^0 \vee R^{-1})$$

In this case, these constants can be chosen as, respectively, 0, 0, 0 and 1. The inequation about the first element is:

$$U^1_{[1,n]}(L^0 \vee R^{-1}) \leq S^1_n \leq U^1_{[1,n+1]}(L^0 \vee R^{-1})$$

$$\leq S^1_n \leq$$



The low value of this inequality is the real value of S^1_n.

For **all finite ground goals**, this rule will generate a **finite computation** because of the existence of **positive loops** in the weighted graph.

Any finite ground term can be rewritten n times iff it is unifiable with S^1_n. The depth of S^1_n increases linearly along the positive loop of the weighted graph.

The **period is 2**, that is, the **lowest common multiple of periods** in the weighted graph (theorem 17).

The only periodic phenomenon will be based on the names of players because they play in turn.

For linear goals, termination, existence of solutions, finity of solution paths can be checked after depth(goal-1)/2 recursive steps (theorems 24, 25, 26).

1. The SLD resolution is finite iff there is no sequence of recursive rewritings whose length is greater than depth(goal-1)/2.

2. There are solutions iff some of them can be found for sequences of recursive rewritings whose length is less than depth(goal-1)/2 + 2 .

3. There is a finite number of solutions iff none of them can be found for sequences of recursive rewritings whose length is an element of

$$[\text{depth(goal-1)}/2 \ , \ \text{depth(goal-1)}/2 + 2[$$

Let us look at some examples of goals:

1. :- game(X, Y, R) .
   Infinite computation, infinite number of solutions.

2. :- game(X, Y, kasparov o e2e4 o karpov o e7e5 o Nil) .
   Finite computation (at most two recursive rewritings), one solution.

3. :- game(X, Y, kasparov o M o P' o M' o karpov o R) .
   Finite computation (at most two recursive rewritings), no solutions.

Let us change the fact of the program:

```
game(hal, kasparov , Nil) .
game(P, P', P o M o R) :- game(P', P, R) .
:- game(kasparov, karpov, Chessgame) .
```

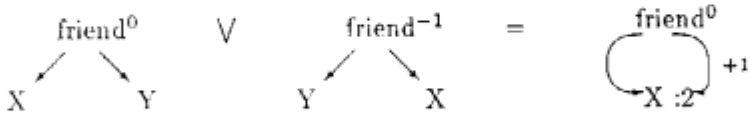
This gives an infinite computation without solutions.

6.2 Commutativity

```
friend(taizo, hirohisa) .
friend(X,Y) :- friend(Y,X) .
```

The fact means that taizo is a friend of hirohisa, and the recursive rule is that X is a friend of Y if Y is a friend of X.

The behaviour of the rule is characterised by the following **weighted graph:**

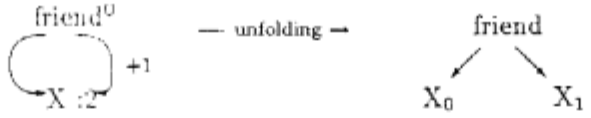


The SLD resolution is infinite for some goals because **its weighted graph exists.**

The following program will generate an infinite SLD computation (without solution):

```
friend(taizo, hirohisa) .
friend(X,Y) :- friend(Y,X) .
friend(X, katsumi) .
```

The most general fixpoint of the rule is the infinite unfolding of the characteristic weighted graph:



This term is finite because the weighted graph is acyclic.

The most general sequence of recursive rewritings, S_n, is ranged across by the interpretation of the weighted graph of the rule from two intervals approximately equal to [1,n]:

$$I^{[1,n+1]}_{[1+a_1,n-b_1]}(L^0 \vee R^{-1}) \;\leq\; S_n \;\leq\; I^{[1,n+1]}_{[1-a_2,n+b_2]}(L^0 \vee R^{-1})$$

These constants can be chosen as, respectively. 0, 0, 0 and 1, that is, the real value of the sequence, S_n, is the upper bound of the inequality:

$$S_n = I^{[1,n+1]}_{[1,n+1]}(L^0 \vee R^{-1})$$

For some **finite ground goals**, the **SLD resolution is infinite** because there are **no positive loops** in the weighted graph.

Any ground instantiation of friend(X,Y) can be rewritten infinitely.

The **period of the rule is 2**, that is, the **lowest common multiple of the periods** in the weighted graph.

The rule expresses the commmutativity of the friend relation.

This program is **bounded** because its weighted graph is **acyclic.**

The following two programs have the same solutions:

1 - friend(t_1, t_2) . friend(X, Y) :- friend(Y, X) .	2 - friend(t_1, t_2) . friend(t_2, t_1) .

Therefore, the complexity of the rule is constant. There is a constant, 1, such that:

1. The SLD resolution is infinite iff the recursive can be applied once.
2. There are solutions iff some of them can be found with ϕ or one recursive rewriting.
3. There is a finite number of solutions iff none of them can be found for one or no recursive rewriting.

Let us look at some examples of goals:

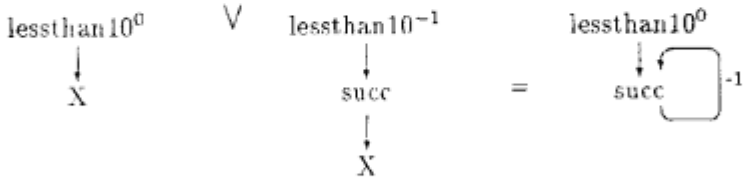
1. :- friend(hirohisa, Y) .
   Infinite computation, finite number of solutions (one).
2. :- brother(hirohisa, Y) .
   Finite computation (no rewriting is possible), no solution.
3. :- friend(Y, katsumi) .
   Infinite computation, no solutions.

6.3 Function "less than 10" on the natural integers

lessthan10(succ^9(zero)) .
lessthan10(X) :- lessthan10(succ(X)) .

The fact means that 9 is less than 10, and the recursive rule is that X is less than 10 if succ(X) verifies the same thing.

The **behaviour of the rule** is characterised by the following **weighted graph**:



The **SLD resolution is infinite for some goals** because **its weighted graph exists**.

The following program will generate an infinite SLD computation:

lessthan10(succ^9(zero)) .
lessthan10(X) :- lessthan10(succ(X)) .
:- lessthan10(zero) .

The most general fixpoint of the rule is the infinite unfolding of the characteristic weighted graph, here, the rational and ground term: $lessthan10(succ^{\infty})$.

For some **finite ground goals**, the **SLD resolution is infinite** because there are **no positive loops** in the weighted graph.

There is no periodic phenomenon during the SLD resolution, because there is no period in the weighted graph.

This program is **bounded** because its weighted graph contains **one negative loop and the fact is ground**. Therefore, the complexity of this Prolog While program is constant.

The following two programs have the same solutions for all goals:

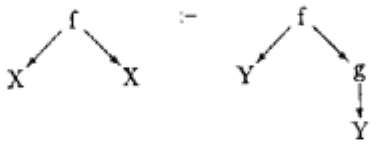
1 - lessthan10(succ^9(zero)) .
lessthan10(X) :- lessthan10(succ(X)) .

2 - lessthan10(succ^9(zero)) .
lessthan10(succ^8(zero)) .
...
lessthan10(zero) .

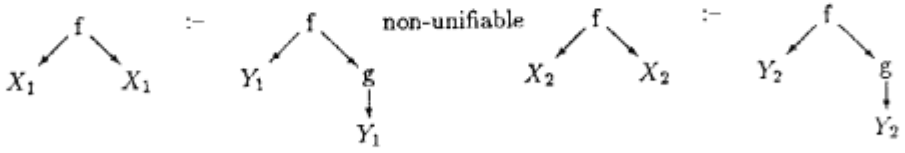
The second program is composed of 10 ground facts, and its complexity is constant.

6.4 Occur-Check

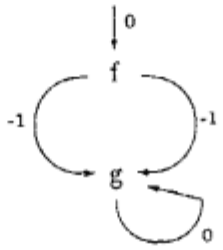
Let us consider the following rule:



This rule is not loop-generating because the longest sequence of rewritings has length 1. Two recursive applications of this rule are not possible because of the occur-check:



The weighted graph which is computed by the unification algorithm is:



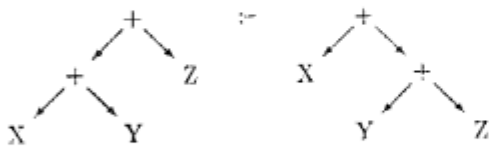
This weighted graph is not finite, that is, there are null loops (from node g). Hence, the unification algorithm fails because of the occur-check.

The complexity of the rule is constant: the SLD resolution is finite for any goal (at least one recursive rewriting), and any goal has a finite number of solutions (at least two solutions). There is no fixpoint and no periodic phenomenon.

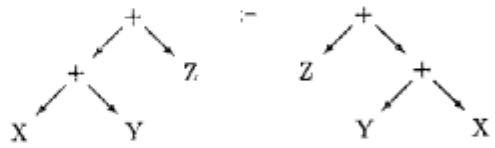
6.5 Similar patterns, different behaviour

Let us look at the following three recursive rules which have no real meaning, except for the first rule which expresses the associativity of the addition:

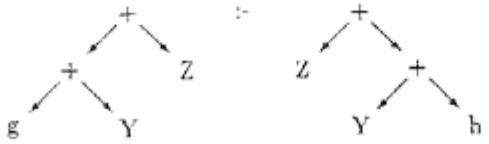
Rule 5



Rule 6

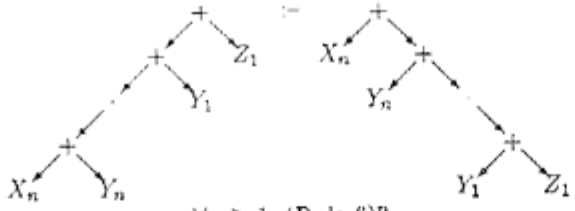


Rule 7

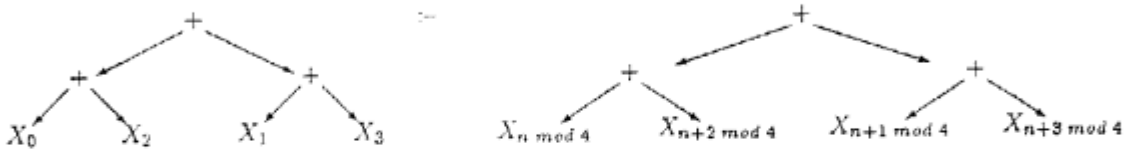


They have the same patterns, but their behaviour is quite different. n recursive applications of the rules 5 and 6 are equivalent to one application of the following rules:

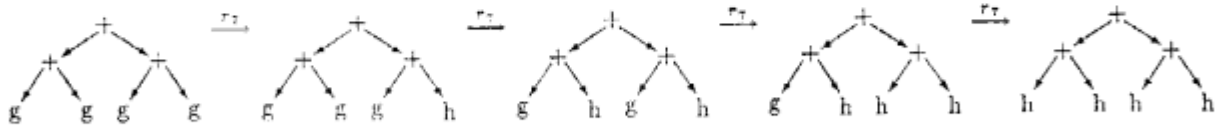
$(\text{Rule } 5)^n$



$\forall n \geq 1, (\text{Rule } 6)^n$

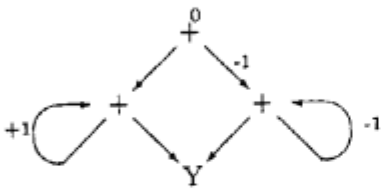


The longest sequence of recursive rewritings using rule 7 is:

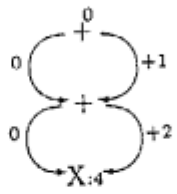


Let us compare and check their recursive behaviour by computing their weighted graphs.

Rule 5



Rule 6



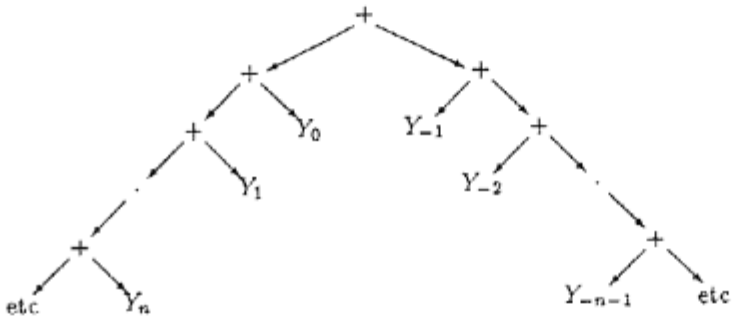
Rule 7

No weighted graph

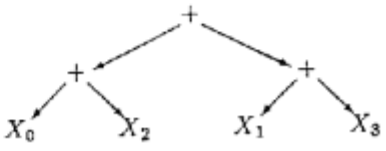
Rules, 5 and 6, can generate an infinite SLD resolution for some goals, because they have their characteristic weighted graph; however, rule 7 always generates a finite SLD resolution because it has no characteristic weighted graph.

The most general fixpoint of rules 5 and 6 are the infinite unfoldings of their weighted graphs:

Rule 5



Rule 6



The fixpoint of rule 5 is infinite because its weighted graph is cyclic. This fixpoint is irrational because it contains an infinite number of variables; and therefore, cannot be expressed by a directed graph. The fixpoint of rule 6 is finite because its weighted graph is acyclic. Rule 7 has no fixpoint because it cannot generate an infinite sequence of rewritings.

For all finite ground terms, rules 5 and 7 generate a finite SLD resolution because weighted graph (5) contains positive loops, and weighted graph (7) does not exist.

However, for some finite ground terms, rule 6 can generate an infinite SLD resolution. Any ground instantiation of its finite fixpoint can be rewritten infinitely.

There is no periodic phenomenon in recursive rules 5 and 7, but rule 6 has a period 4 from the period of the weighted graph.

The complexity of rules 6 and 7 is constant because weighted graph (6) is acyclic, and weighted graph (7) does not exist. After four rewritings in each case, the termination (rule 6), the existence of solutions (rules 6 and 7) and the existence of a finite number of solution paths (rule 6) can be checked. However, the complexity of rule 5 will be constant if the fact is ground (existence of negative loops), but its complexity will generally be linear (existence of positive loops).

	Rule 1 (chess)	Rule 2 (friend)	Rule 3 (inf)	Rule 4 (occur-check)	Rule 5 (associativity)	Rule 6	Rule 7
Infinite SLD for some goals ≡ Finite weighted graph exists	Yes	Yes	Yes	No	Yes	Yes	No
Infinite SLD for some ground goals ≡ Finite wg + positive loop	No	Yes	Yes	No	No	Yes	No
Period of the rule = Period of its fwg	2	2	0	0	0	4	0
Boundedness (constant complexity)	No	Yes (acyclic wg)	Yes (ground fact+ negative loop)	Yes (no fwg)	No	Yes (acyclic wg)	Yes (no fwg)
Number of recursive rewritings for termination, solutions, ...							
Any goal, fact		1	9	1		4	4
Linear goals, facts (x: depth(goal))	x/2				x		

7 Conclusion

This syntactic and structural study of termination and complexity in logic programming began in 1984 at the Laboratoire d'Informatique Fondamentale of Lille.

The first part of this report introduced a generalisation of the directed graphs, called weighted graphs, by putting weights on the arrows and periods on the variables. It presented some formal properties of this new syntactical object, finite and infinite interpretation and unification.

From weighted graphs and through adequate systems of equations, the second part of this work was devoted to establishing the decidability of the termination and the existence of solutions for all programs with the following structure:

$$
\begin{array}{l}
while\,(t_{end})\ . \\
while\,(t_{before})\ :-\ while\,(t_{after})\ . \\
:-\ while\,(t_{begin})\ .
\end{array}
$$

where the terms, t_{end} and t_{begin}, are linear [Devienne 88b]. It is expected that these properties are true even for non-linear facts and goals.

The features of this approach are, first, some coherence for studying the recursive manipulation of terms; these terms have been generalised in the form of weighted graphs which are based on the same algebraic theory and share the same basic operations; second, these results can be understood on three levels:

(1) Algebraic theory: weighted graphs can be studied formally independent of all applications.

However, through the equivalence between the behaviour of a rule, $L \rightarrow R$, and its weighted graph, $L^0 \vee R^{-1}$, the weighted graph properties can be applied within logic programming in two directions.

(2) The weighted graph is a tool of proof and automatic evaluation of termination and complexity for linear recursivity, and can therefore be used for improving strategy or proving the decidability of some properties, for example, the uniform termination of one global rewriting rule.

(3) The weighted graph is a methodological tool that can be used by the programmer for a better understanding of the behaviour of recursive rules.

Although the Böhm-Jacopini theorem has an equivalent formulation in Prolog, that is, any pure Prolog program has a strongly equivalent program of the form [Devienne and Lebegue 88]:

$$
\begin{array}{l}
choice\,(t_1)\ . \\
choice\,(t_2)\ . \\
while\,(t_{end})\ . \\
while\,(t_{before})\ :-\ choice\,(t)\ ,\ while\,(t_{after})\ . \\
:-\ while\,(t_{begin})\ .
\end{array}
$$

it is hoped that this result is not a real limit, as the [Böhm and Jacopini 66] theorem was not a real limit in imperative programming, and that by using the weighted graphs of recursive sub-structures of some Prolog programs, it will therefore be possible to understand their whole behaviour.

Then, the structured logic programming will be established as an efficient and methodological approach.

Acknowledgements

I am most grateful to Max Dauchet for illuminating discussions. I would also like to thank Bruno Courcelle, Pierre Deransart and all the other participants of the working group *Méthodes et outils théoriques en programmation logique* for their helpful comments.

References

[Böhm and Jacopini 66] "Flow diagrams, Turing machines and languages with only two formation rules", *Communications of the Association for Computing Machinery*, Vol.9, pp.366-371, 1966.

[Colmerauer 79] "Prolog II, Manuels de reference, theorique et pratique", *GIA, Marseille*, 1979.

[Colmerauer 84] "Equations and Inequations on Finite and Infinite Trees", *FGCS'84 Proceedings*, Nov. 1984., 85-99.

[Courcelle 83] "Fundamental properties of infinite trees", *Theor. Comp. Sci.*, 17, pp.95-169, 1983.

[Courcelle 86] "Equivalence and transformations of regular systems. Applications to recursive programs schemes and grammars", *Theor. Comp. Sci.*, Vol 42, pp.1-122, 1986.

[Dauchet 87] "Termination of rewriting is undecidable in the one rule case", Internal Report IT110, LIFL Lille, France, 1987.

[Dershowitz 85] "Termination", *First International Conference on Rewriting Techniques and Applications*, Dijon, France, pp.180-224, 1985.

[Dershowitz 87] "Termination of rewriting", *J. Symb. Comp.* 3, pp.69-116, 1987.

[Devienne and Lebegue 86] "Weighted graphs, a tool for logic programming", *11th Colloquium on Trees in Algebra and Programming*, Nice, pp.100-111, 1986.

[Devienne 87] "Les graphes orientés pondérés, un outil pour l'étude de la terminaison et de la complexité dans les systèmes de réécritures et en programmation logique", *Thesis*, Lille, France, 1987.

[Devienne 88a] "Strongly reduced systems of equations", *37th Annual Conference on Information Processing*, Kyoto, Japan, 1988.

[Devienne 88b] "Weighted graphs, a tool for expressing the behaviour of recursive rules in logic programming", *FGCS'88 Proceedings*, Tokyo, Japan, to appear, 1988.

[Devienne and Lebegue 88], "All programming can be done with at most one right recursive rule and three facts", in preparation.

[Eder 85] "Properties of substitutions and unifications", *Journal of Symb. Comp.*, Vol. 1, pp.31-46, 1985.

[Fages] "Notes sur l'unification des termes du premier ordre finis ou infinis", Internal Report, INRIA-LITP, France.

[Gaiman and Mairson 87] "Undecidable optimisation problems for database logic programs", *Symposium on Logic in Computer Science*, New-York, pp.106-115, 1987.

[G. Huet 76] "Resolution d'equations dans les langages d'ordre 1, 2, $\cdots$,Ω", These de doctorat d'etat, Universite Paris VI, 1976.

[Huet and Lankford 78] "On the uniform halting problem for term rewriting systems", *Rapport Laboria 283*, INRIA Le Chesnay, France, 1978.

[Huet 80] "Confluent reductions: Abstract properties and applications to term rewriting systems", *JACM* 27, pp.797-821, 1980.

[Ionnadis 85], "A time bound on the materialisation of some recursively defined views", *Proc. 11th International Conference on very large data bases*, Stockholm, pp.219-226, 1985.

[Lassez, Maher and Marriot 87] "Unification Revisited", *Workshop on Logic and Data Bases*, J. Minker, 1987.

[Lipton and Snyder 77] "On the halting of tree replacement systems", *Conference on Theoretical Computer Science*, Waterloo, Canada, pp.43-46, 1977.

[Lloyd 84, 87] "Foundations of logic programming", *Springer Verlag*, 1984, 1987.

[Naughton 86] "Data independent recursion in deductive databases", *Symp. on Principles of Database Systems*, Cambridge, pp.267-279, 1986.

[Vardi 88] "Decidability and undecidability results for boundedness of linear recursive queries", *Symp. on Principles of Database Systems*, Austin, pp.341-351, 1988.