PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

WEIGHTED GRAPHS, A Tool for Expressing the Behaviour of Recursive Rules in Logic Programming

Philippe DEVIENNE*
Institute for New Generation Computer Technology, Tokyo, Japan
LIFL, Universite de Lille I, 59655 Villeneuve d'Ascq, France†

ABSTRACT

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. Some of the formal properties of these graphs are presented, namely, 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, prove simply the decidability of uniform termination of one rule and range across a finite sequence of recursive rewritings.

Tools of proof of termination and complexity can also be used by the programmer as methodologic tools, or in resolution for improving strategy.

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

*Supported by INRIA grant
†On leave during 1988

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 nonterminating, 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 <u>one</u> 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,

1. friend(X,Y) :- friend(Y,X). (infinite)
   X is the friend of Y if Y is the 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, generally, the non-linearity of the terms, the existence of some variables on one side of the clause, and the permutation of variables during rewriting make intuitive comprehension of behaviour impossible.

The 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→R within term rewriting systems or L :- R in logic programming.

Any looping 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 WEIGHTED GRAPH

The basic syntactic objects used in term rewriting systems and in logic programming are trees, directed acyclic graphs (dags) and directed (or oriented) graphs. They are defined from a finite set of function symbols, denoted F, and a countable set of variables, Var.

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. In term rewriting systems, the term which has to be rewritten is *ground*, that is, without variables. A tree is said to be *linear* if there is only one occurrence of its variables.

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. Thus, the form is a graph whose arrows are directed, but this graph has no cycle. By unfolding, they characterise the finite trees.

Another generalisation exists in the form of a directed graph, which may contain cycles [Courcelle 83]. 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).

Unfortunately, these syntactic objects are too poor to allow good comprehension of the basic recursivity.

2.1 What Is a Weighted Graph ?

Informally, a *weighted graph* is a directed graph where the root and the arrows are weighted by relative integers and the variables may be periodic.

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

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

- X is a set of nodes
- Lab is the label function from X to $F \cup Var$
- Succ is a function from $X \times N$ to $X \underline{\times Z}$
$Succ(x,i) = (x_i, w_i)$: x_i is the i^{th} successor of x and this arrow is weighted by w_i.
- $\underline{Period}$ is a function from Var to N
- $(Root, \underline{w_R}) \in X \underline{\times Z}$, a weighted root.

In comparison to the directed graph definition, only the underlined parts have been added, that is, weight and period. Another definition is proposed in [Devienne 87] : periods on the nodes. The advantage is unification without occur-check, but the major disadvantage is a more complex definition of unfolding.

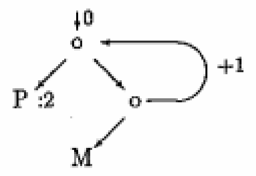


Fig. 1

This graph is directed and contains loops, and therefore looks like a directed graph; however the root (o) is weighted by 0, one arrow is weighted by +1, and variable P has period 2.

2.2 Unfolding of a Weighted Graph

In comparison to directed graph unfolding, four new notions appear, first from the definition, weight and period, and second, for the unfolding control, counter range and initial weight. During unfolding, the weights along a branch are added using a counter.

Weight renames the variables by incrementing the counter, like in resolution (i^{th} inference, variables V_i), and *Period* expresses some periodic phenomena.

The *counter range* is an interval of Z, more precisely, the domain of valid counter values. If the counter value becomes invalid, the unfolding of the branch is stopped and its leaf is labelled by a special variable (associated with the node) indexed by the counter value.

The *initial weight* is considered in addition to the weights of the root and the arrows. This is equivalent to saying that the counter contains an initial value.

The paths in a weighted graph are defined through the following recursive function, named Descendant:

Definition 2 : *Let x be a node, c be a counter value and m be a path, then*

. $Desc_{CR}(x, c, nil) = (x, c)$
. $Desc_{CR}(x, c, i \cdot m) = Desc_{CR}(x_i, c + w_i, m)$
$\quad if\ c \in CR\ \&\ Succ(x, i) = (x_i, w_i)$.

The unfolding of a weighted graph, wg, from the input weight, k, in the counter range, CR, denoted $\mathcal{U}^k_{CR}(wg)$, is defined as follows:

Definition 3 : *Let m be a path and (x, c) be $Desc((Root, w_R + k), m)$, then the node associated with m in the tree, $\mathcal{U}^k_{CR}(wg)$, is equal to:*

. f	$if\ c \in CR\ \&\ Lab(x) = f \in F$
. $V_{c \bmod period(V)}$	$if\ c \in CR\ \&\ Lab(x) = V \in Var$
. Vx_c	$if\ c \notin CR\ \&\ Vx$ *is the special variable of* x.

Let us consider a chess game between two players, P_1 and P_2, who play any move, M_i, in turn. Player P_1 plays move M_1, then P_2 plays M_2, and P_1 plays M_3, and so on. Move M_n is, therefore, played by player $P_{n \bmod 2}$.

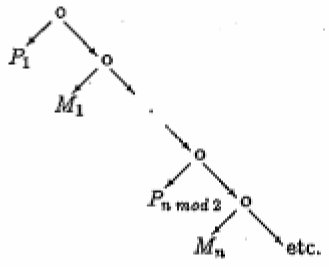
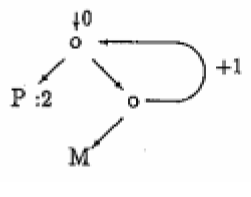


Fig. 2

If we want to express a chess game of n moves, especially if n is infinite, the directed graph is not powerful enough. Let us choose the following weighted graph unfolded from the input weight, 1, in the counter range, CR = [1,n]:

Weighted graph



Along each path of the weighted graph, the unfolding is applied knowing that:

(+1) : the initial value of the counter is 1.
( ) : the counter value is incremented by the weights.
□ *: unfolding stops if the counter value no longer belongs to CR.*

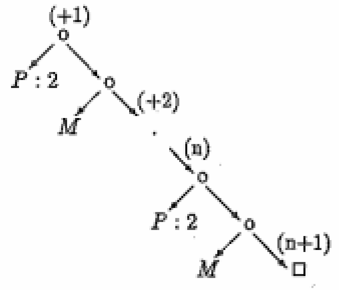


Fig. 3

The variable, M, is indexed by the counter value, that is, the weight sum from the root to the leaf, and the variable, P, by this value modulo of 2, because of the periodicity, 2, of P. The leaf corresponding to the unfolding stop is labelled by the special variable of the node, U, indexed by the invalid counter value, n+1.

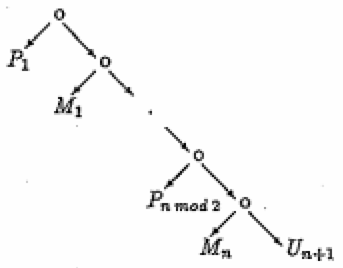


Fig. 4

The obtained tree expresses the most general chess game of n moves. From the input weight, k, belonging to [1,n], the unfolding would be the same chess game, but without the first (k-1) moves.

If CR is Z, that is, the set of relative integers, any counter value is valid, that is, unfolding is applied without control, and, for this example, characterises the most general infinite chess game.

Theorem 1 : *Weighted graphs generalise directed graphs.*

$$Finite\ tree\ \equiv\ Dag\ \subset\ Directed\ graph\ \subset\ Weighted\ graph\ \subset\ Tree.$$

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

Moreover, the weighted graph can express some nonregular trees because of the infinity of variables, and therefore can express no irrational ground trees.

The unfolding in Z of a weighted graph is comparable to directed graph unfolding, in the sense that the unfolded graphs are equal, except for the variables.

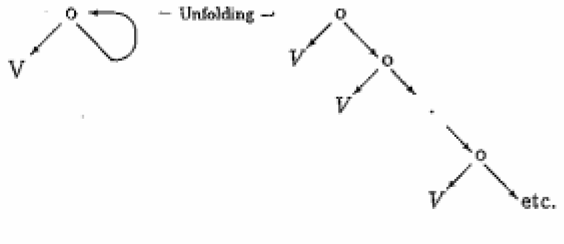


Fig. 5

By putting a weight, 1, on the looping arrow, the unfolded tree is the most general infinite list:

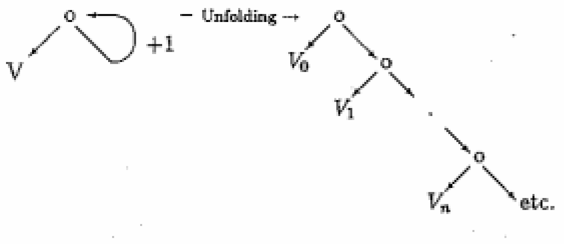


Fig. 6

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

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

Proposition 1 *: The greater the counter range, the more precise and deep the interpretation is:*

$$CR \subset CR' \;\Rightarrow\; \exists\sigma \text{ (substitution)},$$
$$\sigma(\,\mathcal{I}^{IW}_{CR}(wg)\,) \;=\; \mathcal{I}^{IW}_{CR'}(wg).$$

2.3 Finite Weighted Graph

A loop is a path from a node to itself. It is said to be basic if all the nodes, except for the first and the last, are different. The weight of a path is the sum of the arrow weights along it. The path is said to be positive (negative, null respectively) if its weight is positive (negative, null respectively).

Definition 5 *: A weighted graph is said to be finite if it contains no null finite loop.*

Proposition 2 *: A weighted graph is finite iff all the basic loops from a same node have the same sign and are not null.*

Therefore, this property can be easily verified and will correpond to the occur-check.

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

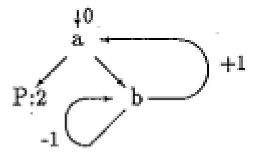


Fig. 7

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

Proposition 3 *: The depth of the unfolding of a finite weighted graph is bounded by a linear function of the size of the counter range.*

2.4 Unification

Definition 6 *: Two weighted graphs are said to be unifiable from IW and in CR if the following system is solvable (with occur-check):*

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

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

$$\{\,(k, \sigma(\,\mathcal{U}^k_{CR}(wg)\,)\;/\;\forall k \in IW\,\}.$$

The unification algorithm of weighted graphs is presented here in the form of a generalisation of the directed graph algorithm of [Fages]. Let us denote the highest common factor as *hcf*, it 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 ($(Root, w_R), (Root', w'_R)$) ;
% $(Root, w_R)$ and $(Root', w'_R)$ are the weighted roots
of wg and wg' %
  If (Root = Root')
  Then period-hcf($Root, |\, w_R - w'_R\,|$)
  Else
    If (Root' is labelled by the variable V')
    Then hcf-period($Root, period(V')$)
         replace($Root', (Root, w_R - w'_R)$)
    Else
      If (Root is labelled by the variable V)
      Then hcf-period($Root', period(V)$)
           replace($Root, (Root', w'_R - w_R)$)
      Else
        If (Root and Root' are labelled by the same
          function symbol)
        Then replace($Root', (Root, w_R - w'_R)$)
          For i:= 1 A n Do
            % Let $Succ(Root, i)$ be $(x_i, w_i)$
            and $Succ(Root', i)$ be $(x'_i, w'_i)$ %
            unification ($(x_i, w_i + w_R), (x'_i, w'_i + w'_R)$);
          Done
        Else Failure: The unifier does not exist.
```

```
Procedure hcf-period ($Root$ , $p$) ;
% Let wg be the weighted graph of Root, %
all the variables of wg must have at least period p %
  If ($p \neq 0$)
  Then If (wg is not cyclic)
       Then $\forall V \in wg, Period(V) := hcf(Period(V), p)$
       Else Failure: The unifier does not exist
```

```
Procedure replace ($Root_1$ , ($Root_2, w$)) ;
% All the arrows going to $Root_1$ have been redirected
to $Root_2$ with the weight correction, w. %
  $\forall x$, $\forall i$, If ($Succ(x,i) = (Root_1, w_1)$)
          Then $Succ(x,i) := (Root_2, w_1 + w)$
```

Theorem 3 : *The weighted graph algorithm applied to two finite weighted graphs, wg and wg', will return a finite weighted graph, denoted $wg \vee wg'$, iff they are unifiable in Z, and*

$$\mathcal{I}_Z^Z(wg)\,)\ \bigvee\ \mathcal{I}_Z^Z(wg') \;=\; \mathcal{I}_Z^Z(wg \vee wg').$$

It will return failure otherwise.

Hence, in the case of Z, the unification of the interpretations of wg and wg' is equal to the interpretation of the unification of wg and wg'.

An immediate and important consequence will be that the most general fixpoints of global rewriting rules are weighted graphs.

The weighted graph unification algorithm is quite similar to the directed graph one, except that the computation of weights and periods has been added.

Corollary 1 : *Let g, g' and $d(wg \vee wg')$ be the directed graphs obtained from wg, wg' and $wg \vee wg'$ by removing the weights and periods, then*

$$g \vee g' \;=\; d(wg \vee wg').$$

Although, generally, the result of the unification of the interpretation of two weighted graphs is not the interpretation of a weighted graph, the weighted graph unifier in Z can be an *approximate solution* of the unification, that is, by decreasing CR, the interpretation of $wg \vee wg'$ is more general than the exact solution, and by increasing CR, the unfolding of $wg \vee wg'$ is less general than the exact solution.

Notation 1 : *$I_{(a\rightarrow\leftarrow b)}$ denotes the sub-interval of I where the low bound of I has been incremented by a and the high bound decremented by b:*

$$[min, max]_{(a\rightarrow\leftarrow b)} \;=\; [min + a, max - b].$$

Similarly, $[min, max]_{(a\leftarrow\rightarrow b)} \;=\; [min - a, max + b]$. These operations can be applied at infinite intervals, for example: $Z_{(a\rightarrow\leftarrow b)} \;=\; Z \;=\; Z_{(a\leftarrow\rightarrow b)}$.

Theorem 4 : *Let wg and wg' be two finite weighted graphs, then there exists a constant, sd, such that for any IW and CR sharing at least sd 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 theorem 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.

Theorem 5 : *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 $(IW \cup CR)_{(a\leftarrow\rightarrow b)}$:*

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

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

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

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.

Let the following be two finite weighted graphs to unify:

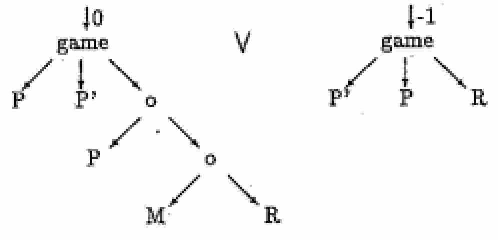


Fig. 8

The constants, a *and* b*, are equal to 1 and 0, and the algorithm gives the following weighted graph:*

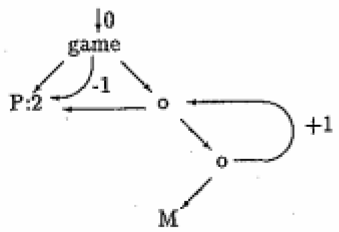


Fig. 9

Note that if all weights and periods are removed in these graphs, this unification is still true within the directed graphs.

3 LOGIC PROGRAMMING

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

3.1 Sequence of Global Rewritings

Definition 7 *: A term, T, is said to be globally rewritten to another term, T', with respect to L→R , if there exists a substitution, θ, such that $\theta(L) = T$ and $\theta(R) = T'$, $T \xrightarrow{r} T'$.*

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→R , if $\sigma(T)$ were globally rewritten to T with respect to L→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, the term, T, and the rule, L→R , are assumed to share no variables.

Proposition 4 *: The most general sequence, S_n, of n global rewritings w.r.t. L→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:

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

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

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

The variables of L→R have a local meaning which means that the variables must be renamed before applying the rule. At the ith global rewriting, rule $L_i \to R_i$ is applied.

Definition 8 *: A rule is said to be looping if it has a most general infinite sequence of global rewritings.*
In others words, L→R is looping 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 5 *: Applying rule L→R n times is equivalent to applying $T_n^1 \to T_n^{n+1}$ once, that is, L→R can generate n rewritings for the goal, G, iff G and T_n^1 are unifiable.*

Simirlarly, L→R can generate an infinity of rewritings for G iff G and T_∞^1 are unifiable.

3.2 Weighted Graph and Global Rule

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

Theorem 6 *: Let L→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*

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

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].

Let the chess game rule be:

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

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 (Fig. 4).

The weighted graphs of this rule, L^0 and R^{-1}, are those which have been unified (Fig. 8).

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.

Theorem 7 *: A rule, L→R , is looping iff the weighted graph, $L^0 \vee R^{-1}$, exists.*
Any looping rule has a weighted graph and two constants, a and b, such that:

$$\mathcal{I}^{[1,n+1]}_{[a,n-b]}(L^0 \vee R^{-1}) \;\le\; S_n \;\le\; \mathcal{I}^{[1,n+1]}_{[-a,n+b]}(L^0 \vee R^{-1}).$$

This is the immediate application of fundamental theorem 4 and shows that the behaviour of L→R is characterised by the weighted graph, $L^0 \vee R^{-1}$. Hence, this weighted graph can be viewed as a meta-term of the rule.

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

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

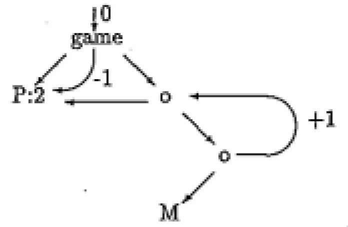


Fig. 10

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 unfolded in Fig. 4.

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.

Theorem 8 *: The most general fixpoint of rule L→R is expressed by the unfolded result of its weighted graph from any input weight without control (CR = Z):*

$$\textit{Most general fixpoint} \;=\; \mathcal{U}_{\mathbb{Z}}^{k}\,(L^0 \vee R^{-1}).$$

Hence, the most general fixpoints of global rewriting rules are weighted graphs.

The most general general fixpoint of the chess game rule is:

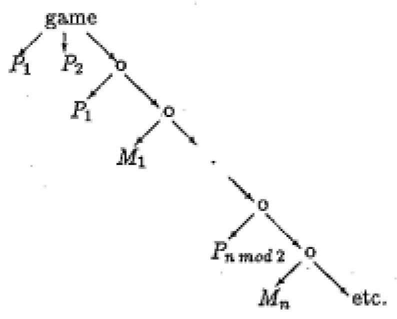


Fig. 11

Proposition 6 *: The depth of terms T_n^1 is bounded iff the weighted graph of the rule does not exist or has no positive basic loop.*
The depth of terms T_n^{n+1} is bounded iff the weighted graph of the rule does not exist or has no negative basic loop.

Apart from the side effects, a term will be finitely rewritten by L→R if one of its ground branches (that is, those whose leaf is labelled by a constant) corresponds to a positive loop in the weighted graph.

This means that the positive loops of the weighted graph can be regarded as possible input informations of the rule (that is, those forcing the termination).

Within term rewriting systems, a rule verifies the uniform termination iff all finite ground terms are finitely rewritten by this rule. In the case of ground terms, the previous remarks give the decidability of uniform termination of one global rewriting rule.

Theorem 9 *: The uniform termination of one global rewriting rule is decidable.*

A rule generates finite global rewritings for all finite ground terms iff the weighted graph, $L^0 \vee R^{-1}$, does not exist or contains positive basic loops.

The chess game rule verifies the uniform termination because of the existence of a positive loop in its weighted graph (Fig. 10).
However, the rule, $friend(U,V)\ :-\ friend(V,U)$, can generate an infinity of rewritings because of no positive loops in its weighted graph :

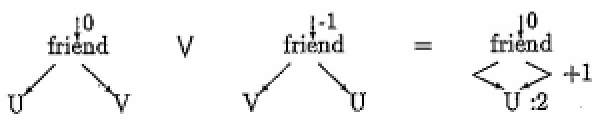


Fig. 12

Remark 1 *: Although the criterion is simple, this result is not obvious and all my attempts to prove it empirically failed. Here, the weighted graph is an efficient tool of proof.*

Moreover, this property is undecidable within rewriting systems. One rule is enough to simulate any Turing machine [Dauchet 87]. This shows all the expressive power of the rewriting.

Theorem 10 *: The uniform termination of one global rewriting rule and <u>one</u> finite ground term is decidable. There exists a linear function, f, depending on the rule such that a finite ground term, T, is globally rewritten finitely iff the length of this sequence of global rewritings is less than f(depth(T)).*

This means that for ground terms, the complexity of L→R is at most linear.

4 CONCLUSION

This paper is the first part of a syntactic and structural study about termination and complexity in logic programming begun in 1984 at the Laboratoire d'Informatique Fondamentale of Lille.

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→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.

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

From weighted graphs and through adequate systems of equations, the second part of this work is 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.

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, therefore, that using the weighted graphs of recursive sub-structures of some Prolog programs, it will be possible to understand their whole behaviour.

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*

[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 systêmes 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 studying termination and complexity in term rewriting systems and logic programming", Technical Report, ICOT, Japan, 1988

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

[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

[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

[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

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