Using Boolean Circuits for the Parallel Computation of Queries

Gagliardi, Edilma Olinda
Herrera, Norma Edith
Reyes, Nora Susana
Turull Torres, José María[1]

{oli, nherrera, nreyes, turull} @unsl.edu.ar

Universidad Nacional de San Luis
Facultad de Ciencias Físico, Matemáticas y Naturales
Departamento de Informática
Ejército de los Andes 950
San Luis, Argentina

ABSTRACT

We present partial results of a research project in which we use boolean circuits as a parallel computation model for the expression of queries to relational databases.

For that purpose, we use the well-known equivalence between First Order Logic (FO) and a class of restricted families of boolean circuits. First, we translate a given query, expressed through a FO formula, into a uniform family of boolean circuits. Then we analyse the depth of the boolean circuits, in order to optimize parallel time. For this sake, we work on the expression tree of the formula, looking for its transformation into an equivalent family of boolean circuits of minimum depth.

Key words: Computation Theory, Relational Databases, Boolean Circuits, Query Computability, and Parallel Computation.

1. Introduction

This work is part of a research project, in which we study different formalisms as computation models for queries to relational databases (see [Barroso et al.,97a], [Barroso et al.,97b], [Gagliardi et al.,99], [Pereyra et al.,98], [Grosso et al, 00a], [Grosso et al, 00b] and [Maldocena et al.,99]), in the consideration that the classic Computability Theory is not appropriate for that purpose ([Chandra et al.,80]). As Computer Science faces problems of growing complexity, the need for a solid theoretical foundation for it, is getting more and more important. The different developments which came out of the fields of Database Theory and Query Computability, lead to the definition of guidelines for the design of databases and queries that aim to avoid possible conflicts which result from the use of database engines which turned out to be inappropriate, from the theoretical perspective.

In the present article, and using theoretical results from complexity theory (see [Balcázar et al.,88], [Balcázar et al.,90] and [Denenberg et al.,86]), we show how a

[1] Universidad Tecnológica Nacional (FRBA);
Universidad Nacional de San Luis;
Partially funded by grants of Universidad CAECE and Universidad Nacional de Luján (project Prodigia)

query to a given relational database, expressed through a formula of First Order Logic (FO), can be translated into a finite subfamily of boolean circuits in the complexity class AC^0. In this way, we get an expression for the query, which is quite suitable for its parallel computation. In a previous work ([Gagliardi et al.,99] and [Pereyra et al.,98]) we used such finite subfamilies of boolean circuits as a suitable formalism for the expression of queries to relational databases. The main foundation for that work, is the well-known equivalence between First Order Logic (FO) and a class of restricted families of boolean circuits.

Given an r-ary query and a database with domain of size n, a finite subfamily of boolean circuits $C=\{C_0, C_1, \ldots, C_q\}$ is constructed, with $q = n^r - 1$, where each C_i decides whether the i-th r-tuple, built with elements of the domain of the given database, belongs to the answer of the query, considering the lexicographical order in the set of r-tuples of the database. Thus, for every n and a given fixed r-ary query, a finite subfamily of boolean circuits is built.

The class of all those subfamilies, for every natural n, constitutes the infinite family of circuits which computes a given query. This problem considered in the context of the Turing machine model, consists of a machine whose input is a formula and a natural number n, and whose output is a subfamily of boolean circuits C, as we described previously.

Here, we look for the optimization of the relation between the size and the depth of the circuits, so that we can make full use of the parallelizability of the query, thus optimizing the parallel time of its evaluation. Clearly, given a circuit and a sufficient number of processors working in parallel, such that they can cover the maximum width of the circuit, the time needed for the parallel evaluation of the query is given by the circuit depth.

We only consider in this work boolean circuits with gates with either one or two inputs (bounded fan-in) and whose output is used as input in at most one other gate (bounded fan-out). Hence, we only consider circuits, which are binary trees, so that we do not allow reusability of the different subcircuits.

First, we give the theoretical background needed to present our results. Then we give the rules for the translation of each FO connective and quantifier, into an equivalent subcircuit. Finally, we build the finite subfamilies of the corresponding boolean circuits for a given database, and we give a strategy for the minimization of the depth of the circuits in the subfamily, which is applicable in some cases.

2. Boolean Circuits

There are several abstract models for parallel computation. Boolean circuit is one of such models, and has been extensively used in the last years. The complexity of boolean circuits is of interest under both practical and theoretical points of view.

One important characteristic of boolean circuits, which makes them different from classical computation models, like Turing machines, is the fact that their input has a fixed length (i.e., boolean circuits constitute a non uniform model of computation, (see [Balcázar et al.,88], [Balcázar et al.,90]). Then, for each input length we must build a different boolean circuit. This fact leads to the generation of an infinite family of boolean circuits, one for each possible input length. On the other hand, if we consider a Turing machine or a program of some programming language, they both can describe an algorithm that solves a problem for any given input, independently of the input length. Such models are referred to as uniform computation models. Then, to turn the family of boolean circuits into a uniform model of computation, we need an algorithm which

given a natural number n as input, builds an encoding of the circuit which computes the given function for all the inputs of size n.

In our case, we build a finite subfamily of boolean circuits, which corresponds to a given FO formula, which expresses a query, and to a given size of the domain of a database. Thus, the property of uniformity is preserved. Our boolean circuits belong to the complexity class NC, which makes formal the notion of well parallelizability of a function.

Next, we give the formal definitions.

- An *m-ary boolean function* is a function $f:\{0,1\}^m \rightarrow \{0,1\}$, for some $m \in N$.
- A *family of boolean functions* is a sequence $f = (f^n)_{n \in N}$, where f^n is an n-ary boolean function.
- A *basis* is a finite set of boolean functions.

Note that a family of boolean functions can be infinite whereas a basis is always finite. We will use the basis $B = \{\neg, \wedge^2, \vee^2\}$.

In the context of our work, a *boolean circuit* is defined as a directed acyclic graph (DAG), which is constructed by associating to each labelled node a variable, a constant or a boolean gate ($\wedge, \vee, \neg$), and by connecting the node g_i to the node g_j with an edge, whenever the output of the gate g_i is an input to the gate g_j . Furthermore, they have a bounded input degree of one or two inputs (bounded fan-in), and they have also a bounded output degree of one (bounded fan-out). Note that, with these restrictions, there is no reusability of subcircuits.

Formally a boolean circuit can be defined as follows:

Let *B* be the basis. A *boolean circuit over B* with two inputs and one output, is a tuple $C = (V, E, \alpha, \beta, \omega)$, where (V, E) is a finite directed acyclic graph, $\alpha: E \rightarrow N$ is an injective function, $\beta: V \rightarrow B \cup \{x_1, x_2\}$, and $\omega: V \rightarrow \{y_1\} \cup \{*\}$ such that the following conditions hold:

- If $v \in V$ has in-degree 0, then $\beta(v) \in \{x_1, x_2\}$ or $\beta(v)$ is a 0-ary boolean function from B.
- If $v \in V$ has in-degree k, $0 < k < 2$, then $\beta(v)$ is a k-ary boolean function from B.
- For every i, $1 \leq i \leq 2$, there is at most one node $v \in V$, such that $\beta(v) = x_i$.
- There is only one node $v \in V$ so that $\omega(v) = y_1$.

Note that x_1, x_2, y_1, * are special symbols attached to certain nodes. With β we indicate the type of the gate v. The function ω defines some nodes as the output nodes, so $\omega(v) = *$ means that v is not an output node. The function α establishes an order between the edges, which induces an order between the nodes.

Thus, a boolean circuit C will compute a boolean function $f:\{0,1\}^2 \rightarrow \{0,1\}$. If a boolean circuit $C=(V,E,\alpha,\beta,\omega)$ computes the characteristic function f_C of some language A, then we also say that C accepts A. If $x \in A^*$ such that $f_C(x) = 1$ then we also say that C accepts x.

The circuits have a finite number of inputs, such that when they are used to compute a function $f:\{0,1\}^* \rightarrow \{0,1\}$ of infinite domain, we must construct a different circuit for each length of the elements in the domain of f. This gives raise to an infinite family of circuits $C=\{C_0, C_1, \ldots, C_n, \ldots\}$, where each C_n computes the function restricted to the domain $\{0,1\}^n$.

A standard encoding, $\overline{C}$, of a circuit C, is a sequence of four-tuples $q_1 \ldots, q_k$ of the form $q_h = (g, b, g_i, g_d)$, $1 \leq h \leq k$; where g represents the gate number; b the gate boolean operation; g_i the gate number or variable or constant, that supplies the left

input to g, and g_d the gate number or variable or constant that supplies the right input to g. There are no four-tuples in the encoding for the nodes labelled with input variables. The output gate of a circuit C is defined to be the gate encoded in the last four-tuple of $\overline{C}$.

The *size of C*, Size(C), is defined as the number of nodes or gates of C. And the *depth of C*, Depth(C), is defined as the length of a longest directed path in C. For $i \geq 0$, we define NC^i = Size-Depth($n^{O(1)}$, $\log^i n$) and NC = $\cup_{i\geq 0} NC^i$.

If we consider circuits of polynomial size and polylogarithmic depth, with unbounded input degree (unbounded fan-in), we define the following classes:

AC^i = Unbounded Size-Depth ($n^{O(1)}$, $\log^i n$), for $i\geq 0$

$AC = \cup_{i\geq 0} AC^i$

And we define:

AC^0 = Unbounded Size-Depth ($n^{O(1)}$, $\log^0 n$).

It was proved that AC^0 is the class of those languages that can be defined by FO formulas, thus FO ≡ AC^0 ([Vollmer,99]) and $NC^0 \subseteq AC^0 \subseteq NC^1$. Then any FO formula can be expressed as a subfamily of boolean circuits in the class Size-Depth(nO(1),log n).

Since we are considering only FO formulas and FO is equivalent to AC^0, then it is in uniform NC^1. Hence, queries expressed through FO formulas are also expressible with families of circuits of polinomial size.

In the general case, when transforming a DAG into an equivalent tree, we may obtain a boolean circuit of exponential size in terms of its depth. In our case, the expansion of a FO quantifier is represented in the corresponding boolean circuit by a subtree of logarithmic depth. If k is the quantifier rank and n is the domain size, the depth of the tree is given by k times the depth of the subtree generated by the expansion of a quantifier. Therefore, in the worst case, the maximum width of the tree will be smaller or equal to $2^{k.\log n}$, so that we get a tree of polynomial width and, hence, also of polynomial size.

Our goal is to get an equivalent subfamily of boolean circuits with logarithmic depth, i.e., in the class Size-Depth($n^{O(1)}$, log n), so that we actually get an important decrease in the time needed for the evaluation of the query, when changing from sequential to parallel computation.

The whole hierarchy NC is the class of the functions for which there exists "efficient" parallel algorithms. The complexity class NC^1 = Size-Depth($n^{O(1)}$, log n) is the appropriate class for our problem.

Bibliographical references: [Balcázar et al.,88], [Balcázar et al.,90], [Denenberg et al.,86] and [Vollmer,99].

3. Relational Databases and Queries

In the relational model, Codd ([Cood,70]) developed the Relational Algebra (AR) and the Relational Calculus (CR) as formal query languages, and he proved that the two languages are equivalent in their expressive power, and, further, equivalent to FO. Considering the relational model in the framework of finite model theory, a database schema is a finite relational signature σ, where each relation symbol in σ has some arity. A database instance is a finite σ-structure, where each relation has the corresponding arity for every symbol of σ, and is defined in the domain of the structure.

In this framework, we consider a (typed) query as a partial recursive mapping whose domain is the set of finite relational structures of a given signature, and whose range is the set of the relations defined in the domain of the corresponding finite

relational structure for some arity fixed, $f:\mathcal{E}_{\sigma,fin} \rightarrow \mathcal{E}_{<R>}$, and such that it preserves isomorphisms ([Chandra et al.,80]). This class was called *CQ* (Computable Queries).

Let σ be a finite relational signature, let L_σ be the first order language with signature σ, let φ in L_σ with r free variables $\{ x_1,x_2...,x_r \}$, and let B be a σ-structure, with $d_1,\ldots,d_n \in D^B$. We will denote as $B \models \varphi(x_1, \ldots,x_r)[d_{s_1},...,d_{s_r}]$ the fact that the formula φ evaluates to true, when interpreted by the structure B, with the element d_{s_j} assigned to the free variable x_j, for $1 \leq s_j \leq n$, $1 \leq j \leq r$. The assignment of the element d_{s_j} to the variable x_j, being x_j a free variable, is done through a *valuation*.

Note that, according to the semantics of the considered language, φ defines a r-ary query f_φ on structure B. That is, $f_\varphi(B)$, denoted by φ^B, it is a finite relation defined in structure B by the formula φ, whose arity will be determined by the amount of free variables of φ. In symbols:

$$\varphi^B = \{(d_{s_1},...,d_{s_r}): d_{s_1},...,d_{s_r} \in D^B \wedge B \models \varphi(x_1, \ldots,x_r)[d_{s_1},...,d_{s_r}], 1 \leq s_j \leq n , 1 \leq j \leq r \}$$

Bibliographical references:[Abiteboul et al.,95], [Chandra et al.,80], [Codd,70], [Ebbinghaus et al.,95], [Ebbinghaus et al.,84], [Hamilton,81], [Turull,96] and [Ullman,88].

4. Translation of FO to Boolean Circuits

When the atomic formula is interpreted for a given valuation, it is satisfied or no, that is, true or false. We can see it as a proposition and we can associate it a propositional variable that represents it. Thus, the atomic formulas expressed and valuated in FO can be translated to the Propositional Calculus (CP) as a propositional variable.

Considering the signature $\sigma = < R_1, R_2,\ldots, R_k >$, such that each R_i is an a_i-ary relation; let the σ-structure $B=\langle D^B, R_1^B,\ldots,R_k^B \rangle$, with $D^B=\{d_1,d_2,...,d_n \}$ and $|D^B|=n$; and let be $R^h(x_1,\ldots,x_{a_h})$ an atomic formula anyone of L_σ, with $1 \leq h \leq k$. We can observe when interpreting $R^h(x_1,\ldots,x_{a_h})$ under a given valuation in the σ-structure B, we obtain one a_h-tuple, that can be considered as a propositional variable p. If p belongs to R_h^B, the proposition p will have the value true, otherwise the truth value will be false. Let v be a valuation, then $B \models R_h(x_1,\ldots,x_r)$ if and only if $R^B(v(x_1),\ldots, v(x_r))$ is hold.

Since $|D^B|^r = n^r$, then associating to each valuation a propositional variable, we obtain n^r propositional variables, and each one can take a true or false truth-value, depending on the satisfaction of the atomic formula. The number of propositional variables depends on the relation arity and on the domain cardinality. The enumeration of all the propositional variables is done following the consecutive order of the relations of the signature. To each r-ary relation corresponds it n^r propositional variables, numbered from $h= \Sigma_{j<i} n^{aj}$ to $(h+n^{(ai)}-1)$.

Following [Denenberg et al.,86], [Vollmer,99] and [Gagliardi et al.,99], given a FO sentence φ, we can inductively build the circuit C that represents the formula, constructing a DAG equivalent a C. We construct an only node with output arc labelled φ. If $\varphi \equiv \varphi_1 \wedge \varphi_2$, this DAG has one labelled node $\wedge$ whose inputs are the labelled arcs φ_1 and φ_2 respectively. For the cases $\varphi \equiv \varphi_1 \vee \varphi_2$ and $\varphi \equiv \neg\varphi_1$ the representation in DAG is similar. If $\varphi \equiv \forall x\, \varphi_1(x)$, the corresponding DAG is a labelled node $\wedge$, with n input arcs labelled $\varphi_1(x)[d_1],\ldots, \varphi_1(x)[d_n]$ respectively, with $d_1,...,d_n \in D^B$, being D^B the domain of the interpretation. If $\varphi \equiv \exists x \varphi_1(x)$, the construction is similar, unless the node is labelled

with the boolean operator $\vee$. Continuing with the decomposition of φ in subformulae, it is arrived at the point in which φ it is a composition of atomic formulas, which valued in the interpretation, will be the propositional variables. Considering the resulting DAG with standard encoding $\overline{C}$ it is obtained the sequence of four-tuples that represents it.

If φ is $\forall x\ \varphi_1(x)$, it will imply the $\varphi_1(x)[d_1]$,…, $\varphi_1(x)[d_n]$ formulas evaluation and connection by the connective $\wedge$. The expression tree would degenerate in a list. For that reason, in this case, we will generate a tree balanced, (applying to the algorithm and-tree [Balcázar et al.,88]), obtaining the suitable depth. Analogously, if φ is $\exists x\ \varphi_1(x)$, the construction is similar, unless the connective is $\vee$ (in this case, we will use the algorithm or-tree).

Bibliographical references: [Balcázar et al.,88], [Balcázar et al.,90],[Denenberg et al.,86], [Vollmer,99], [Gagliardi et al.,99], [Pereyra et al.,98], [Hamilton,81] and [Turull,96].

5. Query Expressed as finite Subfamily of Boolean Circuits

Given the database $B = \langle D^B, R_1^B, \ldots, R_k^B \rangle$, with $D^B=\{d_1, d_2..., d_n\}$, $|D^B| = n$, the amount of possible combinations of the r free variables in D^B is defined by n^r. Thus the following cases appear:

a) $r > 0$ (r-ary query), expressed as $\varphi(x_1, \ldots,x_r)$, with r free variables in FO, each valuation v_i for the free variables $\{x_1,\ldots, x_r\}$ of φ, in the domain D^B, it generates a circuit C_i. We denote the assignation of the element k in the position p of the tuple $(x_1, \ldots,x_r)$ as d_{kp}, i.e.:

$C_0 \equiv \varphi(x_1, \ldots,x_r)[d_{11},\ldots,d_{r1}] \equiv \varphi_0$	Valuation of the r free variables in the first element of domain.
$C_1 \equiv \varphi(x_1, \ldots,x_r)[d_{11},\ldots,d_{r2}] \equiv \varphi_1$	Valuation of the r-1 first free variables in the first element of the domain and the r-th free variable in the second element of the domain.
⋮	⋮
$C_q \equiv \varphi(x_1, \ldots,x_r)[d_{1n},\ldots,d_{r\,n}] \equiv \varphi_q$	Valuation of the r free variables in the n-th element of domain.

For a r-ary query, $q = n^r - 1$, then we obtain a finite subfamily of boolean circuits $C=\{C_0, C_1,..., C_q\}$, equivalent to n^r boolean queries, such that for every valuation $v_i:\{x_1...,x_r\}\rightarrow D^B$, the boolean query φ_i is $\varphi(x_1...,x_r)[v_i(x_1)...,v_i(x_r)]$, for $0 \le i \le q$.

b) $r = 0$, 0-ary query, expressed a sentence φ in FO, it generates only one circuit C, since $n^0=1$.

In this way, we construct the finite subfamily of boolean circuits that represents $\varphi(x_1,\ldots, x_r)$.

With respect to the query evaluation, we denote $B \models \varphi(x_1,\ldots,x_r)[d_{s_1},\ldots,d_{s_r}]$, the fact that the formula φ evaluates to true, when interpreted by the structure B, with the element d_{s_j} assigned to the free variable x_j, for $1\le s_j \le n$, $1\le j \le r$. We denote φ^B, equally, to the formula and the relation that it express:

$$\varphi^B = \{(d_{s_1},\ldots,d_{s_r}) : d_{s_1},\ldots,d_{s_r} \in D^B \wedge B \models \varphi(x_1,\ldots,x_r)[d_{s_1},\ldots,d_{s_r}],\ 1\le s_j \le n,\ 1\le j \le r\}$$

It is represented in the boolean circuits formalism by a circuit C_h, for some h, equivalent to φ_h. We denote with $C_h(B)$ to C_h interpreted in structure B, with the element d_{s_j} assigned to the free variable x_j with $1 \le s_j \le n$, $1 \le j \le r$.

If $B \models \varphi(x_1,\ldots,x_r)[d_{s_1},\ldots,d_{s_r}]$ and $C_h \equiv \varphi(x_1,\ldots,x_r)[d_{s_1},\ldots,d_{s_r}] \equiv \varphi_q$, then $\varphi^B=\{(d_{s_1},\ldots,d_{s_r}): d_{s_1},\ldots,d_{s_r} \in D^B \wedge C_h(B) \equiv \text{true}, q = n^r - 1, 0 \le h \le q, 1 \le s_j \le n, 1 \le j \le r\}$. Note that h is associated to a specific r-tuple $(d_{s_1},\ldots,d_{s_r})$.

The evaluator returns the set of r-tuples for which each $C_h(B) \equiv$ true; in the case r = 0, that is, φ is a sentence, the answer is true sentence or false sentence.

Bibliographical references: [Abbiteboul et al.,95], [Gagliardi et al.,99], [Pereyra et al.,98], [Ebbinghaus et al.,84] and [Vollmer,99].

6. Generation of Suitable Equivalent Formulas for Parallelism

The idea is, we have a FO formula then we want to find another equivalent, such that its expression tree has special characteristics that allow to obtain a finite subfamily of boolean circuits whose width and depth are suitable for the class NC^1, this is Size-Depth($n^{O(1)}$,logn). Previously we introduce the following concepts:

An expression tree that representing a formula φ in FO is a binary tree, building as follows:

- If φ is an atomic formula $R(y_1, \ldots, y_r)$, then it is represented by a node with his corresponding subtrees being empties and containing, as an node attribute, the free variables set.
- In other case, being φ_1 and φ_2 formulas, if:
  a) $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \varphi_1 \vee \varphi_2$, then φ is represented with an expression tree whose root is a node labelled with the respective connective and whose left and right subtrees respectively correspond to the expression trees for φ_1 and φ_2.
  b) $\varphi = \neg\varphi_1$, then φ it is represented with an expression tree whose root is a node labelled with the connective $\neg$ and whose left subtree corresponds to the expression tree for φ_1.
  c) $\varphi = \forall x\, \varphi_1$ or $\varphi = \exists x\, \varphi_1$, then φ is represented with an expression tree whose root is a node labelled with the respective quantifier, whose left subtree corresponds to the expression tree for φ_1 and contains, as an node attribute, the variable x.

We define *weight* of a node as total mapping $P:V \to N$ (Naturals), where V is the nodes set of the formula expression tree, and where $P(v)$ indicates the depth of the corresponding boolean subcircuit that will be generated from *v*.

Let *v* a node of the expression tree of a FO formula, we define recursively the mapping P as follows:

- If *v* represents an atomic formula $R(y_1, \ldots, y_r)$, then $P(v) = 1$
- If *v* represents a formula $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, and being v_1 and v_2 its left child and right child respectively, i.e. they are the subtrees roots that represent φ_1 and φ_2, then $P(v) = 1 + \max(P(v_1), P(v_2))$.
- If *v* represents a formula $\neg\varphi_1$ and if v_1 is its left child, then $P(v) = 1+P(v_1)$.
- If *v* represents a formula $\forall x\, \varphi_1$ or $\exists x\, \varphi_1$, where v_1 is its left child, then $P(v)= 1+\lceil \log n \rceil + P(v_1)$, where n is the cardinality of the structure domain.

We define *width* of a node to a total mapping $A:V \to N$, where V is the nodes set of the formula expression tree, and where $A(v)$ indicates the amount of leaves of the corresponding boolean subcircuit that will be generated from *v*.

Let v a node of the expression tree of a FO formula, we recursively define the mapping A as follows:

- If v represents an atomic formula R (y_1, …, y_r), then A(v) = 1.
- If v represents a formula $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, and being v_1 and v_2 its left child and right child respectively, i.e. they are the subtrees roots that represent φ_1 and φ_2 , then A(v) = A (v_1) + A(v_2).
- If v represents a formula $\neg\varphi_1$ and if v_1 is its left child, then A(v) = A(v_1).
- If v represents a formula $\forall x\ \varphi_1$ o $\exists x\ \varphi_1$, where v_1 is its left child, then A(v) = n × A(v_1), where n is the cardinality of the structure domain.

In this way, when we want to do the syntactic analysis of the FO formula, by each node corresponding to a logical connective or a quantifier, we will consider the boolean circuit structure that will generate such node, and consequently it will receive the appropriate weight and width. For example, for the universal and existential quantifiers, we can see the boolean circuits creation that they will be replaced by a binary tree balanced (by means of the use of the algorithms and-tree and or-tree). For this matter, we must consider that node with the relative weight and width that potentially it will generate. When we quantify an atomic formula is *log n* and *n* respectively, being *n* the cardinality of the domain.

Let us consider the tree in which there is a node that is root of a "unbalanced" subtree. This is because the children weights differ in more than one. There is no form to make changes for balancing it, since to do it would be obtained a non-equivalent semantically formula to the original; that is the obtained expression would not represent the same query. For example, this happens when there is some immersed quantifier in some of the subtrees, (R(x) $\vee$ $\forall y\varphi_1$)) (or analogous ((R(x) $\wedge$ $\exists y\varphi_1$)) where the node corresponding to the first subformula has weight 1 and the other has weight greater than log n.

Note that we have used informally the expression "balanced tree", without specifying its definition. In fact, we will leave this expression replacing it by "partially balanced tree", because it does not correspond with the term use "balance".

Next, we give a method that allows obtaining a *partially balanced tree in weight P of FO formula φ.*

Let T an expression tree for FO formula φ, then we can distinguish different subtrees types T'. Each T' can be:

- The expression tree of an *atomic formula is good*.
- The expression tree having just one node for representing a *quantifier* ($\forall$,$\exists$).
- The expression tree having just one node for representing the *negation connective* ($\neg$).
- The formula expression tree which is maximal in amount of nodes and which all the nodes have the same connective *and* ($\wedge$) or *or* ($\vee$).

Then, for every subtree T' de T, if it leads the first case, it is trivially considered partially balanced tree.

For the other cases, must verify all the subtrees of such tree have been treated and they are partially balanced trees. If there is a subtree with modified weight then, considering T', appear the following cases:

- T' is a node with quantifier or negation connective: its weight is updated with the new weight of its subtree, which already is partially balanced.
- In another case, that is node with connectives $\wedge$ or $\vee$, the balance method is applied.

The *balance method* consists of taking all the subtrees weights depending of T' and whose roots are not nodes of T'. Let φ_1, φ_2,…, φ_k the formulas that represent these subtrees and let the weights p_1, p_2,…,p_k associated to the roots of such subtrees. The weights are increasing sorted, obtaining a sequence $s = <p_{i1}, p_{i2}, ..,p_{ik}>$, with $1 \leq i \leq k$. Now, we analysed:

- If all the pairs of weights differ by more than 1, we take the ordered sequence and we begin from the minor to the greater, building a new expression tree. This tree is estimably the best balance.

  Suppose the connective $\wedge$, this new expression tree will represent the formula $(...(\varphi_{i1} \wedge \varphi_{i2}) \wedge \varphi_{i3}) \wedge\ldots)\wedge\varphi_{ik})$, which is equivalent to the original formula, because the connective has the properties of associative and commutative. Analogously, do it for the connective $\vee$.
- In other case, if there is a maximal subsequent $s' = <p_i,..., p_j>$ *of* s, such that all the pairs of weights differ by more than 1, we build balance tree using the algorithm and-tree (or-tree). The subsequent *s'* is replaced by root weight of the new tree. Thus, the balance method is applied again to the new sequence.

These modifications in the expression tree can be done since the connectives that we are considering, *and* and *or*, satisfy the properties of commutative and associative.

In this way, we obtain an expression tree equivalent to the original expression, and partially balanced in weight. This tree will be better suitable for using parallel computation resources, since its depth is minor.

Let us observe that the width of a FO formula expression tree defines the necessary processors amount for processing it in its maximum parallelism expression. Consequently, with A(v) we indicate the processors amount required for the formula represented by the expression tree with root v, under these conditions.

Given FO formula φ, with expression tree T partially balanced we have:

- If T is totally balanced (in its habitual conception), we can conclude that the formula is well parallelizable and whether r is the root of T, only A(r) processors are necessary.
- Another case, at least there is a node v of T that is root of a tree where the subtrees weights differ by more than 1. Let T_1 and T_2 the left and right subtrees of v, whose roots are v_1 and v_2 respectively, so that $P(v_1) >> P(v_2)$ (the symmetrical case is analogous). We denote with D the difference of levels between both subtrees, according to the boolean circuit that represents v; this is $D = P(v_1) - P(v_2)$. Let $A(v_1)$ and $A(v_2)$ the respective widths of v_1 and v_2. Let T_1' the restriction of tree T_1 at the level $(P(v1) - D)$, or equivalent to the level P(v2), and let h the amount of leaves that has T_1. The variable h represents the authentic amount of processors that are still necessary and used for the processing of the level $(P(v_1) - D)$ of the tree T_1. The amount of processors released in such level can be calculated by means of the expression $(A(v1) - h))$, which we will denote with *l*. Then, having the amount of processors in use, h, and the amount of free processors, *l*, we can analyse the relation between the amount of released processors and the amount necessary to initiate the processing of the right subtree of v: $A(v_2)$. In this way, happen two possible situations:
  a) $l \geq A(v_2)$: we can initiate parallel processing in the right subtree of v.
  b) $l < A(v_2)$: it is not possible to initiate the parallel processing in his maximum parallelism expression.

Let us note that if we arrive to the tree root by the first situation, we obtain a similar result to a totally balanced tree. That is, all the levels of tree T were traversed

parallelly and never it happened a situation of parallelism cut that caused inevitably a sequential processing.

Bibliographical references: [Balcázar et al.,88], [Balcázar et al.,90], [Gagliardi et al.,99] and [Pereyra et al.,98].

7. Example

Let the FO formula φ as follows:

$$\varphi: ((((U(x_1, x_2, x_3) \wedge T(x_1, x_2, x_3)) \wedge \exists x_2 (R(x_1, x_2, x_3) \vee E(x_1, x_2, x_3)) \wedge S(x_1, x_2, x_3)) \wedge \neg R(x_1, x_2, x_3)$$

In the Figure 1.a) we graphically show the expression tree for φ. The Figure 1.b) illustrates the different types of expression subtrees that we distinguished for our analysis. The only subtree that is not partially balanced is T'. The Figure 1.c) indicates the subtrees that depending of T', and that represent the formulas φ_1, φ_2,…, φ_5, indicating the corresponding weights to each one of them. With the applied balance method, it obtains the expression tree for φ', equivalent to φ, which is partially balanced, where φ' is:

$$\varphi': ((((U(x_1, x_2, x_3) \wedge T(x_1, x_2, x_3)) \wedge S(x_1, x_2, x_3)) \wedge \neg R(x_1, x_2, x_3)) \wedge \exists x_2 (R(x_1, x_2, x_3) \vee E(x_1, x_2, x_3)))$$

The Figure 1.d) illustrates the expression tree for φ' and whose tree is partially balanced. Note that the tree depth for φ' is smaller than the tree depth for φ.

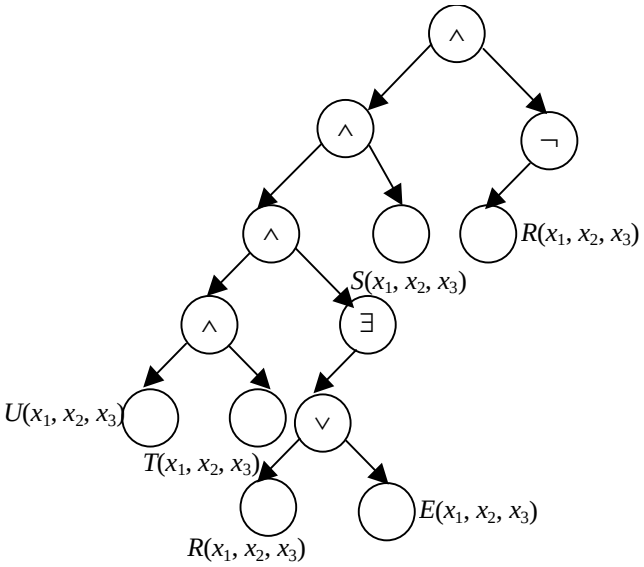

Figure 1.a) Expression tree for φ

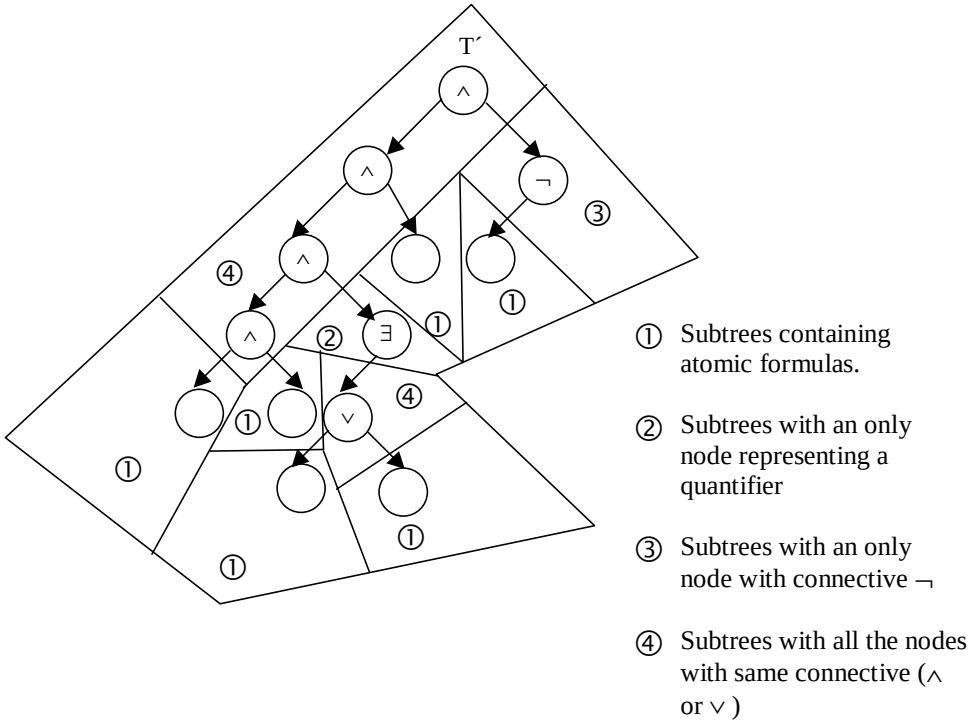

Figure 1.b) Different subtrees types present in the tree at Figure 1.a)

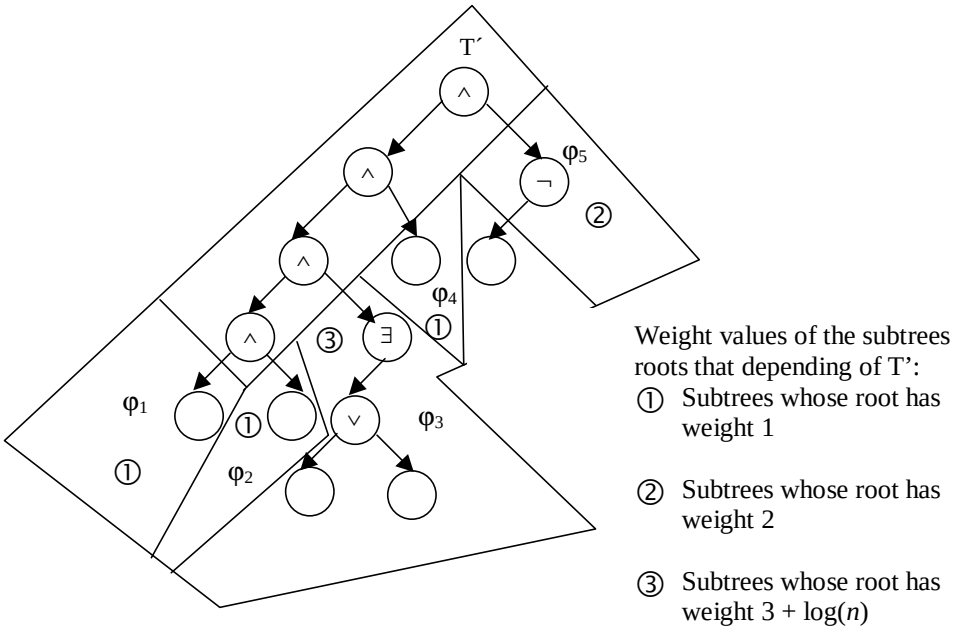

Figure 1.c) T' with the subtrees weights that are depending of it

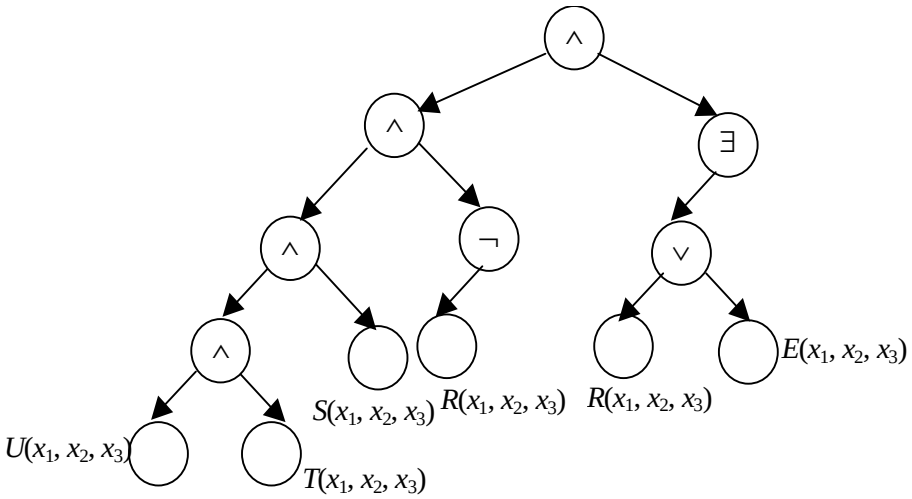

Figure 1.d) Partially balanced expression tree for φ'

8. Conclusions

Boolean circuits are a suitable theoretical model for the study of the computability and parallel complexity of queries to relational databases.

Given an r-ary query and a natural number n that represents the size of the domain of a given database, we showed how to build a finite subfamily of boolean circuits which preserves the property of uniformity, and which has a much better relation between size and depth, thus improving the time needed for the parallel evaluation of the query, as well as the appreciation of the parallelizability of the query.

We intend to generalise our results to general circuits, i.e., without restricting the output fan-in. A quite related topic, which we plan to consider, is the use of boolean circuits for the computation of queries to distributed relational databases. Considering the weakness of First Order Logic as to expressibility, we also plan to extend our results to other logics with bigger expressive power, like the extension of FO with more powerful quantifiers, and higher order logics.

Bibliographical References

[Abiteboul et al.,95] Abiteboul,S; Hull and Vianu, V.; "Foundations of Databases". Addison-Wesley Publishing Company, 1995.

[Abiteboul et al.,91] Abiteboul,S; Vianu, V.; "Generic Computation and Its Complexity", STOC 1991.

[Balcázar et al.,88] Balcázar, J.L;Díaz, J; Gabarró, J; "Structural Complexity I". Springer- Verlag, 1988.

[Balcázar et al.,90] Balcázar, J.L;Díaz, J; Gabarró, J; "Structural Complexity II". Springer- Verlag, 1990.

[Barroso et al.,97a] Barroso, L.J., Gagliardi, E.O., Molina, G.M.., Quiroga, J.A., and Turull, J.M. "An Interpreter for the Complete Language QL" CACIC'97, La Plata, Argentina (in Spanish).

[Barroso et al.,97b] Barroso, L.J., Molina, G.M.. and Quiroga, J.A. “Implementing an interpreter for QL Language”. Undergraduate Thesis in Computer Science, UNSL, 1997. Advisors: Turull, J.M. and Gagliardi, E.O. (in Spanish).

[Chandra et al.,80] Chandra, A.K.; Harel, D. “Computable Queries for Relational Data Bases”. Journal of Computer and System Sciences 21, 156-178. 1980.

[Codd, 70] Codd, E.F.; “A relational model of data for a large shared data banks”. Com of ACM 13/6):377-387,1970.

[Denenberg et al.,86] Denenberg, L.; Gurevich, Y; Shelah, S.; “Definability by Constant-Depth Polinomial- Size Circuits”, Information and Control 70, 2/3 (reprint), 1986.

[Ebbinghaus et al.,95] Ebbinghaus, H; Flum, J.; “Finite Model Theory”, Springer-Verlag, 1995.

[Ebbinghaus et al.,84] Ebbinghaus, H; Flum, J.;Thomas, W.; “Mathematical Logic”, Springer-Verlag, 1984.

[Gagliardi et al.,99] Gagliardi, E.O.; Grosso, A.L; Pereyra, S.R., Piffaretti, P. and Turull J.M., “Computability of Queries to Relational Databases through Boolean Circuits”, CACIC’99, Tandil, Argentina. (in Spanish)

[Grosso et al.,00a] Grosso, A.L., Maldocena, P.D. and Reyes, N.S. and Turull, J.M. “Implementation of an Interpreter for First Order Logic with Fixpoint” CACIC’00, Ushuaia, Argentina (in Spanish).

[Grosso et al.,00b] Grosso, A.L., Maldocena, P.D. and Reyes, N.S. and Turull, J.M. “An interpreter of databases queries expressed by means of first order logic with transitive closure” CACIC’00, Ushuaia, Argentina (in Spanish).

[Hamilton,81] Hamilton, A.G.; “Logic for mathematicians”, Paraninfo, 1981 (in Spanish).

[Maldocena et al.,99] Maldocena, P.D. Reyes, N.S. “Implementation of an Interpreter for First Order Logic with Transitive Closure” Undergraduate Thesis in Computer Science, UNSL, 1999. Advisors: Turull, J.M. and Grosso, A.L. (in Spanish).

[Pereyra et al.,98] Pereyra,S. and Piffaretti,P. “Computation of queries using boolean circuits”. Undergraduate Thesis in Computer Science, UNSL, 1998. Advisors: Turull, J.M., Gagliardi, E.O. and Grosso, A.L. (in Spanish)

[Turull,96] Turull Torres, J.M. “L-rigid Databases and the Expressibility of Incomplete Relational Languages”. Sc.D thesis, UNSL, 1996. (in Spanish)

[Ullman,88] Ullman, Jeffrey D. “Principles of Database and Knowledge Base Systems”. Computers Science Press, 1988.

[Vollmer,99] Vollmer, Heribert, “Introduction to Circuit Complexity, a uniform approach”. Springer Verlag, 1999.