ORB ACADEMIC PUBLISHER

Int. J. Comput. Sci. Theor. App., 2015, vol. 3, no. 1., p. 13-22.
Available online at www.orb-academic.org

International Journal of
Computer Science:
Theory and Application
ISSN: 2336-0984

Inference Control on Information Flow in Logic Programming

Antoun Yaacoub and Ali Awada*

Department of Applied Mathematics, Lebanese University, Lebanon
Email: antoun.yaacoub@ul.edu.lb, al_awada@ul.edu.lb
**Corresponding author.*

ABSTRACT
This paper proposes a formal representation of inference control on information flow theory in logic programming. In order to control the fact that the result returned by a query can convey confidential information, we propose the notion of indistinguishability of flow and elaborate definitions of protection mechanisms, secure mechanisms, precise mechanisms and confidentiality policies based on this notion. We give a secure and precise protection mechanism that prohibits any undesirable inferences and minimizes the number of denials of legitimate actions.

KEYWORDS
Logic programming — Information flow — Inference control — Security mechanism — Flow indistinguishability.

1. Introduction

Data security is the science and study of methods of protecting data in computer and communication systems from unauthorized disclosure and modification. One of the aspects of data security is the control of information flow in the system. In some sense, an information flow should describe controls that regulate the dissemination of information. These controls are needed to prevent programs from leaking confidential data, or from disseminating classified data to users with lower security clearances. The theory of information flow is well defined for imperative programming. Different models of information flow were proposed, namely, the Bell-LaPadula Model [1], nonlattice and nontransitive models [2, 3] of information flow, and nondeductibility and noninterference [4]. Each model has rules about the conditions under which information can move throughout the system. For example, in the Bell-LaPadula Model which describes a lattice-based information flow policy, information can flow from an object in security level A to a subject in security level B if and only if B dominates A. Both compile-time mechanisms [5] and runtime mechanisms [6] supporting the checking of information flows were also proposed. Intuitively, information flows from an object x to an object y if the application of a sequence of commands causes the information initially in x to affect the information in y. For example, the sequence $tmp := x; y := tmp;$ has information flowing from x to y because the value of x at the beginning of the sequence is revealed when the value of y is determined at the end of the sequence. Several studies [7] addressed information flow in imperative programming, but none were concerned to bring answers of what could be an information flow in security systems for logic programming. In fact, logic programming is a well-known declarative method of knowledge representation and programming based on the idea that the language of first-order logic is well-suited for both representing data and describing desired outputs. Logic programming was developed in the early 1970s based on work in automated theorem proving, in particular, on Robinson's resolution principle.

Many researchers tried to link the first order logic with secure systems. DeTreville [8] introduced the concept of an open logic-based security language that encodes security statements as components of communicating distributed logic programs, used to express security statements in a distributed system.

Bertino *et al.* [9] proposed a formal framework for reasoning about access control models. The proposed framework is based on a logical formalism and is general enough to model different access control models. Each instance of the proposed framework corresponded to a C-Datalog program, interpreted according to a stable model semantics.

Wang *et al.* [10] presented a framework that models access control using logic programming with set constraints of a computable set theory. The framework specified policies as stratified constraint flounder-free logic programs that admit primitive recursion.

Bai *et al.* [11] proposed a knowledge oriented formal language to specify the system security policies and their reasoning in response to system resource access request. The semantics of the language was provided by translating it into epistemic logic program in which knowledge related modal operators are employed to represent agents' knowledge in reasoning.

Coetzee *et al.* [12] defined a logic-based access control approach for Web Service endpoint. A logic-based authorization manager provided formal foundation of logical reasoning, that enforced the consistency of access control decisions over the resources of Web Services.

Information flow in logic programming was introduced in [13, 14]. Yaacoub *et al.* defined the information flow in logic programming and developed mechanism to detect such flows. In this paper, we extend the previously presented information flow theory in logic programming to the field of inference control. In section 2, we briefly discuss the syntax and semantics and information flow detection mechanisms in Datalog logic programming. In section 3, we introduce the indistinguishability of the flow in logic programming. We extend this definition and propose the notion of level of goals in logic programming. In section 4, we formally define protection mechanisms, secure mechanisms and confidentiality policies. We end the section by giving specifications of secure protection mechanism for deductive databases using the previously exposed notions. We then give a formal proof of the security of this protection mechanism.

2. Framework

2.1 Syntax and Semantics

We consider here the first order predicate logic [15] language denoted L. This language has countable disjoint sets of variables and predicate symbols. In this language, a term is a variable or a constant. We denote the Herbrand universe of L by U_L, which consists of the set of all ground terms that can be formed with the constants in L. Recall that a ground term is a term where no variable occurs in it. An atom, denoted $p(t_1, \cdots, t_n)$, is obtained by applying an n-ary predicate symbol p to n terms t_i ($1 \leq i \leq n$). An expression $A \leftarrow B_1, \cdots, B_n$ where $A, B_1, \cdots, B_n$ are atoms, is called a `clause`: A its `head` and $B_1, \cdots, B_n$ its `body`. An expression with an empty head is a `goal` (an empty goal is denoted $\Box$), while an expression with an empty body is a `fact`. A finite set of predicate definitions constitutes a logic program.

An idempotent mapping from a finite set of variables to terms is called a `substitution`. ε denotes the identity substitution.

An SLD-derivation for a goal $G = \leftarrow A_1, \cdots, A_n$ with respect to a program P is a sequence of goals $G_0, \cdots, G_i, G_{i+1}, \cdots$, such that $G_0 = G$, and if $G_i = B_1, \cdots, B_m$, then $G_{i+1} = \theta B_1, \cdots, \theta B_{i-1}, \theta B'_1, \cdots, \theta B'_k, \theta B_{i+1}, \cdots, \theta B_m$ such that $1 \leq i \leq m$, and $B \leftarrow B'_1, \cdots, B'_k$ is a variant of a clause in P.
From goal G_i, we obtain G_{i+1} by means of a resolution step, and B_i is said to be the resolved atom. A derivation that is finite and maximal (i.e. the final goal G_n can not be resolved), is called a terminating computation:

- If G_n is the empty goal then, we say that $P \cup \{G\}$ succeeds and the computation is said to succeed with answer substitution θ, where θ is the substitution obtained by restricting the substitution $\theta_n \cdots \theta_1$ to the variables occurring in G;
- If G_n is not the empty goal, then the computation is said to fail. We say that $P \cup \{G\}$ fails if all computations from G in P fail;
- If the derivation is infinite, the computation does not terminate.

2.2 Information flow in Datalog Logic Programming

The theory of information flow in logic programming is based on the innovative work done by Yaacoub *et al*. [13, 14]. They proposed several definitions for flow detection. These definitions correspond to what can be observed by the user when a query $G(x, y)$ is run on a logic program P.

The first definition is based on Success/Failure (SF) of the goals. Let P be a Datalog program and $G(x, y)$ be a two variables goal. There is a flow of information from x to y in P ($x \overset{SF}{\longrightarrow}{}^{P}_{G} y$ based on *SF* in goal G and Program P) if and only if there exists $a, b \in U_{L(P)}$ such that $P \cup \{G(a, y)\}$[1] succeeds and $P \cup \{G(b, y)\}$ fails. This means that when the user only sees the outputs of computations in terms of successes and failures, there exists two different $a, b \in U_{L(P)}$ such that the user can distinguish between the outputs of the goals without seeing what concerns a and b. Now let us show an example to make it clearer.

Example 2.1. *Let P be a program:*
$p(a, b) \leftarrow;$
$p(c, b) \leftarrow;$
and let $G(x, y)$ be the following goal: $\leftarrow p(x, y)$
Since $P \cup \{G(a, y)\}$ succeeds and $P \cup \{G(b, y)\}$ fails, then $x \overset{SF}{\longrightarrow}{}^{P}_{G}$ y based on SF, goal G and Program P. In other words, if we hide a and b from the goals and since the first goal succeeds and the second one fails, we can distinguish by looking at the facts that the first constant is a whereas the second one is b, consequently the flow occurs.

The second definition is based on the substitution answers of the goals. Let P be a Datalog program and $G(x, y)$ be a goal. We can say that there is an information flow from x to y in $G(x, y)$ with respect to substitution answers in P if and only if there exists $a, b \in U_{L(P)}$ such that $\theta(P \cup \{G(a, y)\}) \neq \theta(P \cup \{G(b, y)\})$. In this definition, the user only sees the outputs of computations in terms of substitution answers. Consequently, there is a flow of information from x to y if this user can distinguish the outputs of $P \cup \{G(a, y)\}$ and $P \cup \{G(b, y)\}$.

Example 2.2. *Let us consider the following example of program P:*
$p(a, b) \leftarrow;$
$p(c, d) \leftarrow;$

[1] $P \cup \{G(a, y)\}$ means running the goal $G(a, y)$ on the program P.

and let $G(x,y)$ be the following goal: $\leftarrow p(x,y)$
Since $\theta(P \cup \{G(a,y)\}) = \{y \rightarrow b\}$ and $\theta(P \cup \{G(b,y)\}) = \emptyset$, there is a flow from x to y ($x \overset{P}{\xrightarrow{SA}}_G y$) based on substitution answers in $G(x,y)$ and P. In other words, let us hide both a and b from the goals. The first answer is"b", by looking at the facts we know that y is substituted by "b", then "a" is the hidden constant. Whereas in the second goal, y is substituted by empty set, this means that the first constant is either "b" or "d", consequently, there is a flow.

2.2.1 Results

As in [13], the complexity results obtained for the following two decision problems

$$\pi_{SF} \begin{cases} \text{Input: A logic program } P\text{, a two variables goal } G(x,y) \\ \text{Output: Determine whether } x \overset{P}{\xrightarrow{SF}}_G y \end{cases}$$

$$\pi_{SA} \begin{cases} \text{Input: A logic program } P\text{, a two variables goal } G(x,y) \\ \text{Output: Determine whether } x \overset{P}{\xrightarrow{SA}}_G y \end{cases}$$

are as follows:

- In the general settings of Prolog, the two decision problems are undecidable.
- If the language is restricted to Datalog programs then determining the existence of information flows becomes decidable.
  - π_{SF} is EXPTIME-complete for Datalog programs.
  - π_{SA} is EXPTIME-complete for Datalog programs.

3. Level of indistinguishability of information flow in logic programming

We will proceed in this section to refine the notion of information flow for Datalog logic programs. For this, we propose the notion of level of indistinguishability of the flow.

For a Datalog logic program P and a goal $G(x,y)$ with the variable x considered as an input variable and y as an output variable, let $\equiv$ be a binary relation over $U_{L(P)}$ of cardinality n. Let a,b be two distinct elements of $U_{L(P)}$.

- For the first definition of information flow (based on success/failure), we say that $a \equiv b$ if and only if both $P \cup \{\leftarrow p(a,y)\}$ and $P \cup \{\leftarrow p(b,y)\}$ succeed or both $P \cup \{\leftarrow p(a,y)\}$ and $P \cup \{\leftarrow p(b,y)\}$ do not succeed (in the sense that both goals can fail or not terminate because of the presence of loops).
- For the second definition of information flow (based on substitution/answers), we say that:
  $a \equiv b$ iff $\theta(P \cup \{\leftarrow p(a,y)\}) \cap \theta(P \cup \{\leftarrow p(b,y)\}) \neq \varnothing$.

Lemma 3.1. *The binary relation $\equiv$ is reflexive.*

Lemma 3.2. *The binary relation $\equiv$ is symmetric.*

Lemma 3.3. *The binary relation $\equiv$ is transitive.*

Lemma 3.4. *$\equiv$ is an equivalence relation.*

Proof.
By lemmas 3.1, 3.2 and 3.3.
□

We note here that the definitions presented in [13, 14] rely on the fact that for a logic program P and a goal $G(x,y)$, an information flow passes from x to y if one can find just two distinguishable equivalence classes. In the next subsection, we will use this notion of equivalence classes and its cardinality to define the level of an information flow as one of its characteristics.

3.1 Level of information flows in logic programs

In this section, we present the definitions of the level of information flow based on the notion of equivalence classes.

Definition 3.1 (Level of a logic goal)**.** For a Datalog logic program P and a goal $G(x,y)$, the level of the goal $G(x,y)$ is equal to the cardinality of the smallest equivalence class.

Example 3.5. *Let P be the following program:*
$C_1 : p(a,b) \leftarrow;$
$C_2 : p(a,c) \leftarrow;$
$C_3 : p(b,c) \leftarrow;$
$C_4 : p(c,b) \leftarrow;$
The Herbrand Universe $U_{L(P)}$ is equal to $\{a,b,c\}$.

For the definition of the flow based on success/failure, it is easy to see that:
$P \cup \{\leftarrow p(a,y)\}$ succeeds,
$P \cup \{\leftarrow p(b,y)\}$ succeeds, and
$P \cup \{\leftarrow p(c,y)\}$ succeeds.
Thus, $a \equiv b \equiv c$. Consequently, the cardinality of the equivalence class corresponding to success is equal to 3, while the one corresponding to failure is equal to 0. Thus, the level based on success and failure which corresponds to cardinality of the smallest equivalence class is equal to 0.

For the definition of the flow based on substitution/answers, we have:
$\Theta[P \cup \{\leftarrow p(a,y)\}] = \{y \mapsto b, y \mapsto c\}$,
$\Theta[P \cup \{\leftarrow p(b,y)\}] = \{y \mapsto c\}$, and
$\Theta[P \cup \{\leftarrow p(c,y)\}] = \{y \mapsto b\}$.
Thus, $a \equiv c$, and $a \equiv b$. Consequently, the cardinality of each equivalence class is equal to 2. Consequently, the level based on substitution answers is equal to 2.

One can see that for an Herbrand universe $U_{L(P)}$ of cardinality n, if the level of indistinguishability is n then we have 1 equivalence class which is the class of cardinality n $\{a_1, \cdots, a_n\}$.

Theoretically, this level can be calculated for each of the two definitions of information flow previously presented. For example, for the first definition of flow based on success and failure, one propose the following algorithm:

Require: A Datalog logic program P, a goal $G(x,y)$, finite Herbrand Universe $U_{L(P)} = \{a_1, \cdots, a_n\}$
Ensure: Level of $G(x,y)$
begin
  $Level_{succ} \leftarrow 0$; counter on the number of successful goals
  $Level_{no-succ} \leftarrow 0$; counter on the number of non-successful goals
  $i \leftarrow 1$; counter on the set of the Herbrand universe
  while $i \leq n$ **do**
    if $P \cup \{\leftarrow p(a_i, y)\}$ *succeeds* **then**
      $Level_{succ} \leftarrow Level_{succ} + 1$;
    end
    else
      $Level_{no-succ} \leftarrow Level_{no-succ} + 1$;
    end
    $i \leftarrow i + 1$;
  end
  return $min(Level_{succ}, Level_{no-succ})$;
end

Algorithm 1: Goal level based on success/failure flow definition

Example 3.6. *Let P be the same program as in example 3.5. We saw that the level is equal to 0. By running the algorithm on the same example, the calculated level based on success and failure is thus equal to 0.*

As for the second definition of information flow based on substitution answers, one can write an algorithm similar to Algorithm 2.

Example 3.7. *Let P be the same program as in example 3.5. We saw that the level is equal to 2. We will now run the algorithm on the same example and prove the same result. Recall that the Herbrand Universe is equal to $\{a, b, c\}$, and that the total number of possible and different substitution answers for all the possible goals $P \cup \{\leftarrow p(a,y)\}$, $P \cup \{\leftarrow p(b,y)\}$ and $P \cup \{\leftarrow p(c,y)\}$ is equal to 2, namely, $y \mapsto b$ and $y \mapsto c$. By running the algorithm, the level calculated based on substitution answers is equal to 2.*

3.2 Specification of information flows in Datalog logic programs

In order to introduce the notion of specification, we motivate it first by giving an example.

Consider a system composed by a deductive database (represented as facts in logic programming) and a user who can run queries (in the form of logic goals).

Example 3.8. *Let P be the following program representing:*

- *the three floors and its corresponding departments in a hospital*
  $hospital(floor_1, cancerology) \leftarrow;$
  $hospital(floor_2, cardiology) \leftarrow;$
  $hospital(floor_2, urology) \leftarrow;$
  $hospital(floor_3, gynaecology) \leftarrow$
- *and some of the patients location in the hospital*
  $location(Ana, floor_1) \leftarrow;$
  $location(Bob, floor_1) \leftarrow;$
  $location(Carl, floor_2) \leftarrow;$
  $location(David, floor_2) \leftarrow;$
  $location(Elianor, floor_3) \leftarrow$

The goal is to prevent the user to know exactly for example the existence of the exact departments on each floor or the exact location of some specific patient.

Require: A Datalog logic program P, a goal $G(x,y)$, finite Herbrand Universe $U_{L(P)} = \{a_1, \cdots, a_n\}$, m the total number of possible substitution answers of the y variable for all the goals $P \cup \{\leftarrow G(a_1, y)\}, \cdots, P \cup \{\leftarrow G(a_n, y)\}$.
Ensure: Level of $G(x,y)$
begin
  $table\ T[m]$; table of counters, all initialized to 0, corresponding to the m different substitution answers. The indexes of the table T are the m different substitution answers and the corresponding value represent the total number of occurrence of that substitution answer
  $i \leftarrow 1$; counter on the set of the Herbrand Universe
  $sub \leftarrow \theta[P \cup \{p(a_1, y)\}]$;we initialize the sub by the first substitution answer of y
  while $i \leq n$ **do**
    $tmps \leftarrow \Theta[P \cup \{\leftarrow p(a_i, y)\}]$; as $P \cup \{\leftarrow p(a_i, y)\}$ can have multiple substitution answers, $tmps$ is the table containing these substitution answers
    $j \leftarrow 1$; counter on the set of the substitution answers for the goal $P \cup \{\leftarrow p(a_i, y)\}$
    while $j <= count(tmps)$ **do**
      $T[tmps[j]] \leftarrow T[tmps[j]] + 1$;
      if $T[tmps[j]] < T[sub]$ **then**
        $sub \leftarrow tmps[j]$;
      end
      $j \leftarrow j + 1$;
    end
    $i \leftarrow i + 1$;
  end
  return $T[sub]$;
end

Algorithm 2: Goal level based on substitution answers flow definition

Definition 3.2. For a Datalog logic program P and a two goals $F(x,y)$ and $G(x,y)$, we say that:

- $F(x,y)$ is critical iff $level(F(x,y)) = 1$.
- $F(x,y)$ is weaker than $G(x,y)$ iff $level(F(x,y)) > level(G(x,y))$.
- $F(x,y)$ is stronger than $G(x,y)$ iff $level(F(x,y)) < level(G(x,y))$.

Example 3.9. *It is easy to verify that for the definition of flow of information based on substitution/answers, there are three* *equivalence classes for the goal $P \cup \{\leftarrow hospital(x,y)\}$ of cardinalities 1 and 2. Thus the level of $P \cup \{\leftarrow hospital(x,y)\}$ is equal to 1. Whereas, for the goal $P \cup \{\leftarrow location(x,y)\}$, one can count three equivalence classes too, two of cardinality 2 and one of cardinality 1. So, the level of $P \cup \{\leftarrow location(x,y)\}$ is equal to 1. Consequently, both goals are critical.*

Lemma 3.10. *For a Datalog logic program P and a goal $F(x,y)$, if $F(x,y)$ is critical, then the output variable y reveals information about the variable x.*

Proof.

By definition, if a goal $F(x,y)$ is critical then the level of $F(x,y)$ is equal to 1. Moreover, when a level of a goal is equal to 1, it means that the cardinality of the corresponding equivalence class of the goal $F(x,y)$ is equal to 1. Thus, the variable x will be uniquely identified.

□

Example 3.11. *As both goals $P \cup \{\leftarrow hospital(x,y)\}$ and $P \cup \{\leftarrow location(x,y)\}$ are critical, the output variable y can convey information as we can see next. Suppose that a user runs the goal $P \cup \{\leftarrow hospital(floor_1,y)\}$, then the corresponding output variable y will be unified uniquely with cancerology. Thus the disclosure of this information will render the identification of the probable disease of the patient residing on the first floor very easy.*

Whereas, if the user runs the goals $P \cup \{\leftarrow location(Ana,y)\}$ and $P \cup \{\leftarrow location(Bob,y)\}$, the corresponding output variable y will be unified uniquely with $floor_1$, and the user will still know that both Ana and Bob are sharing the same floor but without knowing anything about their diseases. Moreover, let us suppose that the user runs the goals $P \cup \{\leftarrow location(Carl,y)\}$ or $P \cup \{\leftarrow location(David,y)\}$, the y variable will be unified with cardiology and urology. So Carl and David can be both in the same cardiology or urology department or each one of them in a different department. A natural question arises here, ***what should the system do when it detects that some queries are critical ?***

As for the other forms of specifications, one can verify that the level of the goal $P \cup \{\leftarrow location(Carl,y)\}$ is greater than the one of $P \cup \{\leftarrow location(Elianor,y)\}$ and thus the last goal is stronger than the former one.

4. Secure and Precise Security Mechanisms

Based on what we have presented, one can ask the following question: is it possible to devise a generic procedure for developing a mechanism that is both secure and precise using the notion of information flow for logic programs?

For this, we will consider here logic programs as a set of clauses, having all the same predicate definition. The atoms in this logic programs have several input positions but one single output position. Data is *brought in* to a clause through the input positions, and *sent out* through the output positions.

Example 4.1. *Let P be the following Datalog logic program:*
$C_1 : q(a,b) \leftarrow;$
$C_2 : r(b,a) \leftarrow;$
$C_3 : p(x,y) \leftarrow q(x,z), r(z,y);$

Let $\alpha, \beta, \gamma, \delta, \zeta$ and κ be following argument positions[2] of all the variables in the clause C_3:
$\alpha = < C_3, 0, p, 1 >$, $\beta = < C_3, 0, p, 2 >$, $\gamma = < C_3, 1, q, 1 >$, $\delta = < C_3, 1, q, 2 >$, $\zeta = < C_3, 2, r, 1 >$ *and* $\kappa = < C_3, 2, r, 2 >$.
Let α, γ and ζ be in $I(C_3)$, β, δ and κ in $O(C_3)$. $I(C_3)$ denotes the input positions of the clause C_3 and $O(C_3)$ denotes the output positions of the clause C_3.

Seeing that the program P have three different predicate definitions, namely, q, r and p, we can rewrite the program in such a way to have only one predicate definition:
Let P' be P's equivalent program:
$C'_1 : t(q,a,b) \leftarrow;$
$C'_2 : t(r,b,a) \leftarrow;$
$C'_3 : t(p,x,y) \leftarrow t(q,x,z), t(r,z,y)$
P' has now one predicate definition, namely t. t has 3 arguments. The first two are input arguments and the last one is an output argument. It is easy to see that P and P' are equivalent according to the least fixpoint semantics. Thus, one can rewrite any logic program like P into an equivalent logic program having one predicate definition. In the next, we will only consider logic program composed simply of facts, and we will denote a program P by its predicate definition.

We will represent logic programs as abstract functions:

Definition 4.1. (Logic programs as abstract functions) For a logic program P denoted by its predicate definition $t(I_1, \cdots, I_n, O)$, where $I_1, \cdots, I_n$ are input positions and O one output position, let p be the function $p : I_1 \times \cdots \times I_n \times O \rightarrow R$. Then p is the function with n **inputs positions** $i_k \in I_k, 1 \leq k \leq n$, and **one output position** $o \in O$, and **one result** $r \in R$. O is the set of substitution/answers associated to the output position o. Depending on each definition of information flow, R can be equal to $\{success, failure\}$, or to the set of substitution/answers corresponding to the output position o, or to the SLD-tree of the goal $P \cup \{\leftarrow t(i_1, \cdots, i_n, o)\}$.

With respect to the Lemma 3.10, we assume that the result $r \in R$ of the function $p(i_1, \cdots, i_n, o)$ conveys information about the input variables $i_1, \cdots, i_n$.

Dealing with confidentiality, a natural question arises here, whether if the result of $p(i_1, \cdots, i_n, o)$ contains any information that could violate the policy. For this, **protection mechanisms** are proposed. A protection mechanism produces for every input that do not violate the policy, the same value as for p, and for inputs that would impart confidential information an error message. For this, let E be the set of results from a program p that indicate errors.

[2] If c is a clause of the form $a_0 \leftarrow a_1, \cdots, a_n$, the position of the k^{th} argument of the j^{th} literal is uniquely defined in the program P by the tuple $< c, j, p, k >$, where p is the predicate symbol of the j^{th} literal of c

Definition 4.2. (Protection mechanism) Let p be a function $p: I_1 \times \cdots \times I_n \times O \to R$. A protection mechanism m is a function $m: I_1 \times \cdots \times I_n \to R \cup E$ for which, when $i_k \in I_k, 1 \le k \le n, o \in O$, either

- $m(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$ or
- $m(i_1, \cdots, i_n) \in E$

Example 4.2. *We consider here a logic program P, with one input position and one output position, that contains the age of some individuals.*
$age(ann, 56) \leftarrow$;
$age(billy, 27) \leftarrow$;
$age(carl, 34) \leftarrow$

The program P is represented by the function: $age: I, O \to R$. Queries would be of the form $P \cup \{\leftarrow age(ann, A)\}$ for example.

A protection mechanism would be for example to answer correctly (in the terms of the different information flow definitions) every query whenever its input position variable corresponds to one of the Herbrand Universe constants. More formally, let m be the following function:
$m: I \to R \cup E$ for which:

- *$m(i) = age(i, o)$ when $i \in U_{L(P)}$, ($o \in O$)*
- *$m(i) = Error$, otherwise.*

Now we define a confidentiality policy.

Definition 4.3. (Confidentiality policy) A confidentiality policy for the logic program $p: I_1 \times \cdots \times I_n \times O \to R$ is a function $c: I_1 \times \cdots \times I_n \to J_1 \times \cdots \times J_n$, where $J_1 \subseteq I_1, \cdots, J_n \subseteq I_n$.

Informally, the sets $J_i, 1 \le i \le n$ corresponds to sets of inputs that may be revealed. The function c acts as a filter by bearing leakage of confidential inputs by seeing that the complements of J_i with respect to I_i represent the sets of inputs that must be kept confidential.

Example 4.3. *Let c be the confidentiality policy that bears leaking information about ann for example; thus, for $c: I \to J$, where $I = \{ann, billy, carl\}$ and $J = \{billy, carl\}$, $c(billy) = billy, c(carl) = carl$ and $c(ann)$ is undefined.*

Now we define what we hear about a **secure** mechanism.

Definition 4.4. (Secure mechanism) Let $c: I_1 \times \cdots \times I_n \to J_1 \times \cdots \times J_n$ be a confidentiality policy for a program p. Let $m: I_1 \times \cdots \times I_n \to R \cup E$ be a security mechanism for the same program p. Then the mechanism m is **secure** (i.e. confidential) if and only if there is a function $m': J_1 \times \cdots \times J_n \to R \cup E$ such that, for all $i_k \in I_k, 1 \le k \le n, m(i_1, \cdots, i_n) = m'(c(i_1, \cdots, i_n))$.

Example 4.4. *Let us check if this security mechanism is secure and this for the first two definitions of information flow previously presented:*

	Success /failure	*Substitution /answers*
***Query**: $P \cup \{\leftarrow age(billy, A)\}$*	*success*	$\theta = \{A \mapsto 27\}$
***Protection mec**: $m(billy)$*	*success*	$\theta = \{A \mapsto 27\}$
***Sec. mec**: $m(c(billy))$*	*success*	$\theta = \{A \mapsto 27\}$
***Query**: $P \cup \{\leftarrow age(diana, A)\}$*	*failure*	$\theta = \{\}$
***Protection mec**: $m(diana)$*	*error*	*error*
***Sec. mec**: $m(c(diana))$*	*error*	*error*
***Query**: $P \cup \{\leftarrow age(ann, A)\}$*	*success*	$\theta = \{56\}$
***Protection mec**: $m(ann)$*	*success*	$\theta = \{56\}$
***Sec. mec**: $m(c(ann))$*	*error*	*error*

In this example, we have showed for three queries the result of the protection mechanism, and checked if it is secure. Even if for the first two queries, the results show that this is the case, the third query reported a discordance. Thus, the protection mechanism presented in this example is not secure. We will present later in this paper a mechanism that is secure.

Despite the fact that a secure mechanism ensures that the policy is obeyed, it may disallow actions that do not violate it and thus be overly restrictive. Next we define the notion of precision which measures the degree of **overrestrictiveness**.

Definition 4.5. Let m_1 and m_2 be two distinct protection mechanisms for the logic program p under the policy c. In the rest, o is an output position.

Then m_1 is as precise as m_2 ($m_1 \succ m_2$) provided that, for all inputs $(i_1, \cdots, i_n)$, if $m_2(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$, then $m_1(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$.

We say that m_1 is more precise than m_2 ($m_1 \gg m_2$) if ($m_1 \succ m_2$) and there is an input $(i'_1, \cdots, i'_n)$ such that $m_1(i'_1, \cdots, i'_n) = p(i'_1, \cdots, i'_n, o)$ and $m_2(i'_1, \cdots, i'_n) \ne p(i'_1, \cdots, i'_n, o)$.

$m_1 \gg m_2$ implies that m_1 never gives a violation notice when m_2 does not. This implies that the utility of m_1 is at least as high as of m_2.

Lemma 4.5. *The relation $\succ$ is reflexive and transitive on the protection mechanisms for a given p and c.*

Proof.

- $\succ$ is *reflexive*: let m_1 be a protection mechanism for p and c, then $m_1 \succ m_1$ because for all the inputs $(i_1, \cdots, i_n)$, if $m_1(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$, then obviously $m_1(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$.
- $\succ$ is *transitive*: let m_1, m_2 and m_3 be three protection mechanisms for p and c such that $m_1 \succ m_2$ and $m_2 \succ m_3$. $m_2 \succ m_3$ means that for all inputs $(i_1, \cdots, i_n)$, if $m_3(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$, then $m_2(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$ and $m_1 \succ m_2$ means that for all inputs $(i_1, \cdots, i_n)$, if $m_2(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$, then $m_1(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$.
  Thus, for all inputs $(i_1, \cdots, i_n)$, if $m_3(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$, then $m_1(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$. This establishes that $m_1 \succ m_3$.

□

Lemma 4.6. *The relation $\gg$ is a strict ordering on the protection mechanisms for a given p and c.*

Proof.

- $\gg$ is *irreflexive*: let m_1 be a protection mechanism for p and c, then $m_1 \not\gg m_1$ since there is no input $(i'_1, \cdots, i'_n)$ such that $m_1(i'_1, \cdots, i'_n) = p(i'_1, \cdots, i'_n, o)$ and $m_1(i'_1, \cdots, i'_n) \neq p(i'_1, \cdots, i'_n, o)$.
- $\gg$ is *asymmetric*: let m_1 and m_2 be two protection mechanisms for p and c such that $m_1 \gg m_2$. $m_1 \gg m_2$ implies that $m_1 \succ m_2$ and there is an input $(i'_1, \cdots, i'_n)$ such that $m_1(i'_1, \cdots, i'_n) = p(i'_1, \cdots, i'_n, o)$ and $m_2(i'_1, \cdots, i'_n) \neq p(i'_1, \cdots, i'_n, o)$.
  Thus, $m_2 \not\succ m_1$, since there will be the input $(i'_1, \cdots, i'_n)$ for which $m_1(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o)$, but $m_2(i_1, \cdots, i_n)$ $\neq p(i_1, \cdots, i_n, o)$. Thus $\gg$ is asymmetric.
- $\gg$ is *transitive*: let m_1, m_2 and m_3 be three protection mechanisms for p and c such that $m_1 \gg m_2$ and $m_2 \gg m_3$. $m_2 \gg m_3$ implies that $(m_2 \succ m_3)$ and there is an input $(i'_1, \cdots, i'_n)$ such that $m_2(i'_1, \cdots, i'_n) = p(i'_1, \cdots, i'_n, o)$ and $m_3(i'_1, \cdots, i'_n) \neq p(i'_1, \cdots, i'_n, o)$. $m_1 \gg m_2$ means that $(m_1 \succ m_2)$ and there is an input $(i"_1, \cdots, i"_n)$ such that $m_2(i"_1, \cdots, i"_n) = p(i"_1, \cdots, i"_n, o)$ and $m_3(i"_1, \cdots, i"_n) \neq p(i"_1, \cdots, i"_n, o)$.
  Thus, there is an input $(i'_1, \cdots, i'_n)$ such that $m_1(i'_1, \cdots, i'_n)$ = $p(i'_1, \cdots, i'_n, o)$ and $m_3(i'_1, \cdots, i'_n) \neq p(i'_1, \cdots, i'_n, o)$.
  Since the relation $\succ$ is transitive, the relation $\gg$ is thus transitive.

□

Example 4.7. *For the same previous example, let $m_1 : I \to R \cup E$ be the following protection mechanism*

- *$m_1(i) = p(i, o)$ when $i \in U_{L(P)}$,*
- *$m_1(i) = Error$, otherwise.*

and $m_2 : I \to R \cup E$ a protection mechanism that uses a counter cn on the number of queries already asked. cn is initialized to 1, and incremented by 1 on every query ran against the program.

- *$m_2(i) = p(i, o)$ when $cn\%2 = 0$,*
- *$m_2(i) = Error$, otherwise.*

Suppose that the user asks the following set of queries : $\{P \cup \{\leftarrow age(billy, A), P \cup \{\leftarrow age(ann, A), P \cup \{\leftarrow age(david, A)\}$. Next, we will be interested only with the second definition of flow, i.e. based on substitution/answers, as it could be easily generalized to the other definitions.
The corresponding answers are as follows: $\theta = \{A \mapsto 27\}$, $\theta = \{A \mapsto 56\}$ and $\theta = \{\}$.
For the first protection mechanism, the answers would be respectively, $\theta = \{A \mapsto 27\}$, $\theta = \{A \mapsto 56\}$ and error, while for the second protection mechanism, the answers would be $\theta = \{A \mapsto 27\}$, error and $\theta = \{\}$ respectively.

In this example, m_2 is not as precise as m_1, because $m_1(ann) = p(ann, o)$ and $m_2(ann) \neq p(ann, o)$. m_1 is not as precise as m_2, because $m_2(david) = p(david, o)$ and $m_1(david) \neq p(david, o)$. Thus, one cannot establish that $m_1 \gg m_2$ or $m_2 \gg m_1$. A question arises here, what about combining two protection mechanisms?

By combining two protection mechanisms, we obtain a new mechanism that is as precise as the two original ones.

Definition 4.6 (Union of protection mechanisms)**.** Let m_1 and m_2 be protection mechanisms for the program p. Then their union $m_3 = m_1 \cup m_2$ is defined as

$$m_3(i_1, \cdots, i_n) \begin{cases} = p(i_1, \cdots, i_n, o) \text{ when} \\ \qquad m_1(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o) \text{ or} \\ \qquad m_2(i_1, \cdots, i_n) = p(i_1, \cdots, i_n, o) \\ = m_1(i_1, \cdots, i_n) \text{ otherwise.} \end{cases}$$

One can see that the previous definition is not symmetric, since $m_1 \cup m_2 \neq m_2 \cup m_1$.

Example 4.8. *For the same logic program in example 4.2, let $m_1 : I \to R \cup E$ be the following protection mechanism*

- *$m_1(i) = p(i, o)$ when $i \in U_{L(P)} \backslash \{ann\}$,*
- *$m_1(i) = Error$, otherwise.*

and $m_2 : I \to R \cup E$ the following one:

- *$m_2(i) = p(i, o)$ when $i \in U_{L(P)}$,*
- *$m_2(i) = Error$, otherwise.*

Then,
$m_3(ann)$ = $m_1(ann) \cup m_2(ann) = m_2(ann) = p(ann, o)$
$m_3(billy)$ = $m_1(billy) \cup m_2(billy) = m_1(billy) = p(billy, o)$
$m_3(carl)$ = $m_1(carl) \cup m_2(carl) = m_1(carl) = p(carl, o)$
Note that here $m_2 \gg m_1$.

From this definition and the definitions of secure and precise, we have:

Theorem 4.9 (Union of secure protection mechanisms)**.** *Let m_1 and m_2 be secure protection mechanisms for a program p and policy c. Then $m_1 \cup m_2$ is also a secure protection mechanism for p and c. Furthermore, $m_1 \cup m_2 \succ m_1$ and $m_1 \cup m_2 \succ m_2$.*

From secure protection mechanisms $m_1, m_2 \cdots$, one can define the secure protection mechanism $m^* = m_1 \cup m_2 \cup \cdots$ such that $m^* \succ m_1, m^* \succ m_2, \cdots$. Thus, we have the following generalization of the previous theorem:

Theorem 4.10. *For any program p and security policy c, there exists a precise, secure mechanism m^* such that, for all secure mechanisms m associated with p and c, $m^* \succ m$.*

Proof.

Let $m = \{m' | m'$ is a secure protection mechanism for p and $c\}$. Let m^* be $\cup_{n \in m} n$. Then by Theorem 4.9, $m^* \succ n$ for any secure protection mechanism m^*; hence, m^* is maximal. □

m^* is the mechanism that ensures security while minimizing error messages.

Example 4.11. *We consider here a logic program P, with one input position and one output position, that contains the salary of some individuals in euros.*
$salary(abby, 2500) \leftarrow;$
$salary(bob, 2500) \leftarrow;$
$salary(carla, 2400) \leftarrow$

The program P is represented by the function: $salary : I, O \to R$.

Let c be the confidentiality policy that bears leaking information about all input variables, namely abby, bob and carla. Thus, for $c : I \to J$, where $I = \{abby, bob, carla\}$ and $J = \{\}$, $c(abby)$ is undefined, $c(bob)$ is undefined and $c(carla)$ is undefined.

A trivial protection mechanism for example, would be in this case to not answer any query. Formally, let m be the following function:
$m : I \to R \cup E$ for which:

- *$m(i) =$ no answer, where $i \in U_{L(P)}$.*

Obviously, this protection mechanism is secure, but what about the existence of mechanisms that are both secure and precise in the sense that the mechanism ensures security while minimizing the number of denials of legitimate actions. For this, we will use the notion of level of a flow, previously presented in section 3 to define our secure protection mechanism. In this example, we allow the observer to see the query sequences issued by the users and to have some a priori knowledge by retaining all previously returned answers. ***Note that in the query sequences visualized by the observer, the input parameters are kept hidden****.*

Recall that for every query $\leftarrow p(i, o\}$ in a program p, we associate an ***equivalence class****, denoted $\bar{o}$, and thus a cardinality. In the program p above, abby $\equiv$ bob, since $\Theta(P \cup \{\leftarrow p(abby, o\}) = \Theta(P \cup \{\leftarrow p(bob, o\}) = \{o \mapsto 2500\}$, and consequently the $card(\overline{2500}) = 2$. One can see that $card(\overline{2400}) = 1$.*

For our protection mechanism, we associate to each equivalent class of cardinality higher or equal than 1, a random number $\alpha > 1$.

As long as the queries are asked, the system counts the number of queries asked in each equivalence class. If the level associated to the query is equal to 1, the protection mechanism will respond by no answer. Also, when the number of queries corresponding to an equivalence class is equal to its associated random number α, the protection mechanism will respond by no answer. Otherwise, the protection mechanism will answer the query by giving its substitution answer sets.
Formally, let m be the following protection mechanism:
$m : I \to R \cup E$ for which:

- *$m(i) =$ no answer, if for $P \cup \{\leftarrow p(i, o)\}$, $card(\bar{o}) = 1$,*
- *$m(i) =$ no answer, if for $P \cup \{\leftarrow p(i, o)\}$, $nc_{\bar{o}} = \alpha_{\bar{o}}$,*
- *$m(i) = p(i, o)$, otherwise.*

Above, $nc_{\bar{o}}$ is a counter corresponding to the number of queries already asked associated to the equivalence class of the goal $P \cup \{\leftarrow p(i, o)\}$.

Let the random numbers associated be as follows:
$\alpha_{\overline{2500}} = 1$, and $\alpha_{\overline{2400}} = 1$.

As stated earlier, the observer can visualize query sequences with the input parameter hidden (in the next shown in red) and can have ***some*** *a priori knowledge. An a priori knowledge that should not contain any information about the random numbers α, because an omniscient observer can easily violate the confidentiality policy, as shown next:*
Suppose that the observer sees the following query sequences:
$\{P \cup \{\leftarrow salary(abby, o)\}, P \cup \{\leftarrow salary(abby, o)\},$
$P \cup \{\leftarrow salary(bob, o)\}, P \cup \{\leftarrow salary(carla, o)\}\}$.

Let Query 1 be $P \cup \{\leftarrow salary(abby, o)\}$. The protection mechanism won't return a result. Since $\alpha_{\overline{2500}} = 1$, the protection mechanism will not answer the ***first*** *query of this equivalence class $\overline{2500}$. As it is the first query, the observer cannot deduce anything about the input parameter. In fact, the input parameter could be any element of $\{abby, bob, carla\}$.*

Let Query 2 be $P \cup \{\leftarrow salary(abby, o)\}$. The protection mechanism will return $\{o \mapsto 2500\}$ as a result. This is the second query in the class $\overline{2500}$, the protection mechanism will reply by giving the substitution answer. By giving the substitution answer of a query belonging to the equivalence class $\overline{2500}$, the observer is now sure that the ***first query*** *concerns the same class $\overline{2500}$.*

Let Query 3 be $P \cup \{\leftarrow salary(bob, o)\}$. The protection mechanism will return $\{o \mapsto 2500\}$ as a result. The is the third query in the class $\overline{2500}$, the protection mechanism will reply by giving the substitution answer. The observer learns nothing more from the third query.

Let Query 4 be $P \cup \{\leftarrow salary(carla, o)\}$. The protection mechanism won't return a result. Since $\alpha_{\overline{2400}} = 1$, the protection mechanism will not answer the ***first*** *query of this equivalence class $\overline{2400}$. As the observer knew that a previously asked query concerned the equivalence class $\overline{2500}$ and returned no answer, the current returned result no answer necessarily concerns the equivalence class $\overline{2400}$, and thus,* ***the observer is able to state that the hidden input parameter for this query is carla****.*

Thus, the random numbers α associated to each equivalence class should not be disclosed to the observer.

*Let us now show that the protection mechanism is secure. For this, we need to define what is meant by '****the observer can infer the exact value of the hidden input parameter****'. For this, we associate to the query sequences issued by the user, a vector of the observed substitution answers. Formally, for the query sequence $Q = \{P \cup \{\leftarrow p(a_1, o_1\}, P \cup \{\leftarrow p(a_2, o_2\}, \cdots, P \cup \{\leftarrow p(a_n, o_n\}\}$, we associate the observed returned results in terms of substitution answers $\Theta = (\theta_1, \theta_2, \cdots, \theta_n)$.*

We say that the observer can guess the exact value of a hidden input parameter a_i, if for the corresponding θ_i, and in ***all*** *the query vectors $\{P \cup \{\leftarrow p(a_1,o_1\}, P \cup \{\leftarrow p(a_2,o_2\}, \cdots, P \cup \{\leftarrow p(a_n,o_n\}\}$, a_i have the same value.*

Let us show now that our mechanism is secure.
Suppose that the logic program is composed by at least two facts (the case where the logic program is composed by 1 fact is trivial, as the hidden input parameter is unique). Suppose that there exists a substitution answer θ_i for which a_i has the same value in ***all*** *the query vectors $\{P \cup \{\leftarrow p(a_1,o_1\}, P \cup \{\leftarrow p(a_2,o_2\}, \cdots, P \cup \{\leftarrow p(a_n,o_n\}\}$, that is*
$\theta(p(a_i,o_i)) = \theta_i$.
Thus, for this θ_i, there is a unique associated input parameter a_i. Furthermore, there is a unique fact of the form $p(a_i,b) \leftarrow$ with $\theta(p(a_i,o_i)) = \{o_i \mapsto b\}$.
But, according to the protection mechanism; for each equivalence class of cardinal 1, the associated returned answer by the mechanism is no answer. So, in this case, $\theta_i = \varepsilon$ (ε is used to note the no answer returned value).
Thus, seeing that for $\theta_i = \varepsilon$, there is ***one*** *associated input parameter a_i, this means that the* ***logic program is composed by one fact only****, because according to the mechanism, a no answer should be returned to* ***one*** *of the queries in each equivalence class (or to all the queries for equivalence classes with cardinality equal to 1). Note that for $\theta_i = \varepsilon$, there should be a number of distinct a_i equal to the number of different equivalence classes in the program. This contradicts the fact that the logic program is composed by at least two facts.*

5. Conclusion

In this paper, we used some definitions of information flow for Datalog logic programs to introduce the notion of flow indistinguishability level. We proposed an equivalence relation between the elements of the Herbrand universe relatively for a Datalog logic program P. We showed that the notion of indistinguishability proposed, coincides with the one presented in [13, 14] since practically it suffices to find two indistinguishable classes to state that an information flow passes from x to y in the goal $P \cup \{G(x,y)\}$. Based on the notion of equivalence classes, we proposed also the definition of the level of an information flow. Algorithms were proposed to calculate this level, and this for the two definitions of information flow, namely, based on success/failure and substitution answers. We then discussed the specifications of the flow and give an example to emphasize the fact that the result returned by the query can convey confidential information.

To control this, we focused on the notion of inference control and we proposed definitions of protection mechanisms, secure mechanisms, precise mechanisms and confidentiality policies. We ended by giving an example of a secure and precise protection mechanism that prohibits any undesirable inferences and minimizes the number of denials of legitimate actions.

As a future work, one can first conduct thorough experiments and comparisons between the different mechanisms proposed in the litterature. It is desirable to investigate the defintion of flow detection mechanism for logic programs based on bisimulation [13]. It is tempting to check whether protection mechanisms, secure mechanisms, precise mechanisms and confidentiality policies could be expressed using this defintion.
Moreover, since nowadays, security in distributed systems is an important issue, one can try to couple the works of [8] and [16] to study and formalize security mechanisms in distributed systems.

References

[1] Bell, D. and LaPadula, L. Secure Computer Systems: Mathematical Foundations and Model. *The MITRE Corporation Bedford MA Technical Report M74244*, 1973, vol. 1.

[2] FOLEY, Simon N. A model for secure information flow. *In: Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on.* IEEE, 1989, p. 248-258.

[3] DENNING, Dorothy E. A lattice model of secure information flow. *Commun. ACM 19, 5 (May 1976)*, 1976, vol. 19, no 5, p. 236-243.

[4] GOGUEN, J. and MESEGUER, J. Security policies and security models. *IEEE Symposium on Security and Privacy*, 1982, p. 11-20.

[5] DENNING, Dorothy E. et DENNING, Peter J. Certification of programs for secure information flow. *Communications of the ACM*, 1977, vol. 20, no 7, p. 504-513.

[6] FENTON, J. FENTON, Jeffrey Stewart. Memoryless subsystems. *The Computer Journal*, 1974, vol. 17, no 2, p. 143-147.

[7] DENNING, D. ROBLING DENNING, Dorothy Elizabeth. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

[8] DETREVILLE, J. DETREVILLE, John. Binder, a logic-based security language. *In : Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2002. p. 105-113.

[9] BERTINO, Elisa, CATANIA, Barbara, FERRARI, Elena, et al. A logical framework for reasoning about access control models. In: *Proceedings of the sixth ACM symposium on Access control models and technologies (SACMAT '01)*, 2001, p. 41-52.

[10] WANG, Lingyu, WIJESEKERA, Duminda, et JAJODIA, Sushil. A logic-based framework for attribute based access control. In : *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*. ACM, 2004. p. 45-55.

[11] BAI, Yun et KHAN, Khaled M. A modal logic for information system security. In : *Proceedings of the Ninth Australasian Information Security Conference-Volume 116*. Australian Computer Society, Inc., 2011. p. 51-56.

[12] COETZEE, Marijke et ELOFF, J. H. P. A logic-based access control approach for Web Services *ISSA 2004 (Information Security for South Africa)*, 2004.

[13] BALBIANI, Philippe et YAACOUB, Antoun. Deciding the bisimilarity relation between datalog goals. In : *Logics in Artificial Intelligence. Springer Berlin Heidelberg*, 2012. p. 67-79.

[14] YAACOUB, Antoun. Towards an information flow in logic programming. *International Journal of Computer Science Issues (IJCSI)*, 2012, vol. 9, no 2.

[15] LLOYD, J.W. *Foundations of Logic Programming*, 2nd Edition. Springer, 1987.

[16] YAACOUB, Antoun, AWADA, Ali et KOBEISSI, Habib. Information Flow in Concurrent Logic Programming. *British Journal of Mathematics & Computer Science*, 2015, Vol. 5, no 3, p. 367-382.