# Using SAT and SQL for Pattern Mining in Relational Databases

**Emmanuel Coquery**[1] and **Jean-Marc Petit**[1] and **Lakhdar Sais**[2]

**Abstract.** In this paper, we present an ongoing work bridging the gap between pattern mining, SQL and SAT for a particular class of patterns. We extend the work presented in [2] that proposes a logical query language for rule patterns satisfying Armstrong's axioms. Our contributions are the following: firstly, we allow a large part of the relational tuple calculus (SQL) to be used in the specification of queries. Secondly, we propose a boolean encoding of the query that can be used to compute answers even in the case of non Armstrong-compliant queries. Some experiments have been performed on top of Derby (embedded Java DBMS) and a modified version of MiniSat to show the feasibility of the approach.

## 1 INTRODUCTION

Declarative approaches for pattern mining attracted a growing attention in the recent years. On one hand, a way to increase declarativity is to devise high level query languages for data mining [10, 13, 14, 5, 8, 11, 2]. On the other hand, more declarativity often induce a greater expressivity, at the cost of a reduced efficiency. Constraint programming approaches for pattern mining, initiated in [15], were proposed to provide a good compromise between expressivity and efficiency.

In this paper, we propose to use SQL and SAT together for pattern mining. We extend the work presented in [2] that proposes a logical query language for rule patterns satisfying Armstrong's axioms. While such patterns can be enumerated with efficient algorithms they might be seen as too restrictive. The first contribution of this paper is to allow a large part of the relational tuple calculus (SQL) to be used in the specification of queries. The second one is the use of a SAT solver for enumerating patterns even in the case of non Armstrong-compliant queries.

The rest of this paper is organized as follows. Section 2 presents the $\mathcal{RLT}$ query language. Section 3 presents the basic principles of the query boolean encoding. Implementation principles are presented in section 4 together with a few optimizations, while section 5 provides some ways to reduce the number of answers. Finally some experimental results are presented in section 6 and we conclude in section 7.

## 2 THE $\mathcal{RLT}$ LANGUAGE

In this section, we introduce the syntax and semantics of the $\mathcal{RLT}$ language, which is based on a mixture of tuple relational calculus [1] and $\mathcal{RL}$ language [2].

### 2.1 Preliminaries

We introduce the following definitions and notations used in the $\mathcal{RLT}$ language:

- $\mathcal{U}$ is a set of *attributes*, noted $\bar{A}, \bar{B} \ldots$,
- $\mathcal{CST}$ is a set of constants,
- $\mathcal{CMP}$ is a set of binary[3] comparisons over $\mathcal{CST}$. It is assumed that if $\bar{c}$ and $\bar{c'}$ are two constants, and if $\square \in \mathcal{CMP}$, then $\bar{c} \ \square \ \bar{c'}$ is computable in constant time.
- a schema $R$ is a finite, nonempty set of attributes from $\mathcal{U}$,
- a tuple $\bar{t}$ over a schema $R$ is a total function from $R$ to $\mathcal{CST}$,
- $\bar{t}[\bar{A}]$ denotes the value of $\bar{t}$ for attribute $\bar{A}$,
- a relation $\bar{r}$ over a schema $R$ is a set of tuples over $R$.

- $s, t, u, s_1, \ldots$ are tuple variables,
- $A, B, C, A_1, B_1 \ldots$ are attribute variables, i.e. capital letters from the beginning of the alphabet,
- $X, Y, Z, X_1, Y_1 \ldots$ are schema variables, i.e. capital letters from the end of the alphabet,
- $r, r_1, r' \ldots$ are relation symbols.

To avoid ambiguity with variables, we shall use the following notations for attributes, set of attributes and tuples:

- $\bar{A}, \bar{B}, \bar{C}, \bar{A_1}, \bar{B_1} \ldots$ are single attributes,
- $\bar{X}, \bar{Y}, \bar{Z}, \bar{X_1}, \bar{Y_1} \ldots$ are set of attributes,
- $\bar{s}, \bar{t}, \bar{t_1}, \bar{t_2}, \ldots$ are tuples,
- $\bar{c}, \bar{c_1}, \bar{c'}$ are constants.

For any function $f : E \rightarrow E'$, we denote by $f[e := e']$, where $e \in E$ and $e' \in E'$, the function $f'$ which maps $e$ to $e'$ and maps $e_1$ to $f(e_1)$ if $e \neq e_1$.

For any function $f : E \rightarrow E'$, and any subset $E_1 \subseteq E$, we denote by $f_{|E_1}$ the restriction $f' : E_1 \rightarrow E'$ of $f$ to $E_1$, which maps $e_1 \in E_1$ to $f(e_1)$.

### 2.2 $\mathcal{RL}$ Formulas

This section recalls the syntax and semantics of the $\mathcal{RL}$ language [2]. In this paper, we restrict the use of tuple variable quantifiers, by removing them from definition 1 and reintroducing them in section 2.4. We also limit the use of attribute quantifiers to some form of restricted quantification.

[1] Université de Lyon, CNRS, France, email: firstname.lastname@liris.cnrs.fr

[2] Université d'Artois, CNRS, France, email: sais@cril.univ-artois.fr

[3] the *binary* restriction can be easily lifted, allowing arbitrary SQL boolean expressions

Let $A, B$ be attribute variables, $t, s$ tuple variables, $\bar{c}$ a constant, $X$ a schema variable.

**Definition 1** *The set of $\mathcal{RL}$-formulas, noted $\delta, \delta_1, \delta, \ldots$, is inductively defined as the smallest set verifying:*

- *$t.A \,\square\, \bar{c}$, $t.A \,\square\, s.B$, $A = B$ and $A = \bar{A}$ are* atomic *$\mathcal{RL}$-formulas*
- *If $\delta$ is a $\mathcal{RL}$-formula and $A$ an attribute variable, $\forall A(X)(\delta)$ is a $\mathcal{RL}$-formula*
- *If $\delta$ is a $\mathcal{RL}$-formula and $A$ an attribute variable, $\exists A(X)(\delta)$ is a $\mathcal{RL}$-formula*
- *If $\delta_1$ and $\delta_2$ are $\mathcal{RL}$-formulas, then $\neg\delta_1$ and $(\delta_1 \wedge \delta_2)$ are $\mathcal{RL}$-formulas*

Other logical connectors such as $\vee, \Rightarrow$ and abbreviations **true, false** are defined as usual.

**Definition 2** *A* $\mathcal{RL}$-interpretation *is a quadruplet $(R, \Sigma, \sigma, \tau)$ where:*

- *$R \subseteq \mathcal{U}$ is a schema,*
- *$\Sigma$, the schema interpretation, is a function mapping each schema variable $X$ to a subset of $R$,*
- *$\sigma$, the attribute interpretation, is a function mapping each attribute variable $A$ to an attribute $\bar{A} \in R$,*
- *$\tau$, the tuple interpretation, is a function mapping each tuple variable $t$ to a tuple $\bar{t}$ over $R$.*

**Definition 3** *Let $\delta$ be a $\mathcal{RL}$-formula. The* satisfaction *of $\delta$ with respect to a $\mathcal{RL}$* interpretation *$(R, \Sigma, \sigma, \tau)$, denoted by $(R, \Sigma, \sigma, \tau) \models \delta$, is defined inductively as follows:*

- $(R, \Sigma, \sigma, \tau) \models t.A \,\square\, \bar{c}$ *if* $\tau(t)[\sigma(A)] \,\square\, \bar{c}$
- $(R, \Sigma, \sigma, \tau) \models t.A \,\square\, s.B$ *if* $\tau(t)[\sigma(A)] \,\square\, \tau(t)[\sigma(B)]$
- $(R, \Sigma, \sigma, \tau) \models A = \bar{A}$ *if* $\sigma(A) = \bar{A}$
- $(R, \Sigma, \sigma, \tau) \models A = B$ *if* $\sigma(A) = \sigma(B)$
- $(R, \Sigma, \sigma, \tau) \models \forall A(X)(\delta)$ *if for all* $\bar{A} \in \Sigma(X)$, $(R, \Sigma, \sigma[A := \bar{A}], \tau) \models \delta$
- $(R, \Sigma, \sigma, \tau) \models \exists A(X)(\delta)$ *if for some* $\bar{A} \in \Sigma(X)$, $(R, \Sigma, \sigma[A := \bar{A}], \tau) \models \delta$
- $(R, \Sigma, \sigma, \tau) \models \neg\delta$ *if* $(R, \Sigma, \sigma, \tau) \not\models \delta$
- $(R, \Sigma, \sigma, \tau) \models (\delta_1 \wedge \delta_2)$ *if* $(R, \Sigma, \sigma, \tau) \models \delta_1$ *and* $(R, \Sigma, \sigma, \tau) \models \delta_2$

## 2.3 Relational Calculus

Here we recall some definitions of the tuple relational calculus [1] (abbreviated TRC in the following). Let $\bar{A}, \bar{B}$ be attributes, $t, s$ be tuple variables, $\bar{c}$ be a constant and $r$ be a relation symbol.

**Definition 4** *The set of TRC-formulas, noted $\psi, \psi_1, \psi', \ldots$, is inductively defined as the smallest set verifying:*

- *$t.\bar{A} \,\square\, c$ and $t.\bar{A} \,\square\, t.\bar{B}$ are* atomic *TRC-formula*
- *$r(t)$ is an* atomic *TRC-formula*
- *if $\psi$ is a TRC-formula, and $t$ is a tuple variable then $\exists t(\psi)$ is a TRC-formula*
- *if $\psi_1$ and $\psi_2$ are TRC-formulas then $(\psi_1) \wedge (\psi_2)$ and $\neg(\psi_1)$ are TRC formulas*

Other logical connectors such as $\vee, \Rightarrow$, quantifier $\forall$ and abbreviations **true, false** are defined as usual.

Now we recall the logical semantics of TRC-formulas. For sake of simplicity, we assume that all relations and schemas are defined over the same schema $R$. This restriction can easily be lifted though.

**Definition 5** *A TRC interpretation is a pair $(d, \tau)$ where:*

- *$d$ is a function, the database, mapping each relation symbol $r$ to a relation $\bar{r}$ over $R$,*
- *$\tau$ is a function, the tuple interpretation, mapping each tuple variable $t$ to a tuple $\bar{t}$ over $R$.*

**Definition 6** *Let $\psi$ be a TRC-formula. The satisfaction of $\psi$ with respect to a TRC interpretation $(d, \tau)$, denoted $(d, \tau) \models \psi$, is inductively defined as follows:*

- $(d, \tau) \models t.\bar{A} \,\square\, \bar{c}$ *if* $\tau(t)[\bar{A}] = \bar{c}$
- $(d, \tau) \models t.\bar{A} \,\square\, t.\bar{B}$ *if* $\tau(t)(\bar{A}) = \tau(t)(\bar{B})$
- $(d, \tau) \models r(t)$ *if* $\tau(t) \in d(r)$
- $(d, \tau) \models \neg(\psi)$ *if* $(d, \tau) \not\models \psi$
- $(d, \tau) \models \exists t(\psi)$ *if there exists a tuple $\bar{t}'$ over $R$ such that* $(d, \tau[t := \bar{t}']) \models \psi$
- $(d, \tau) \models (\psi_1) \wedge (\psi_2)$ *if* $(d, \tau) \models \psi_1$ *and* $(d, \tau) \models \psi_2$

In the rest of the paper, we restrict ourselves to *authorized relational calculus* [1], a syntactical restriction of the relation calculus which garanties domain independence. That is, given a database $d$, the set of tuple interpretations $\tau$ such that $(d, \tau) \models \psi$ only depends on $d$ and $\psi$.

**Definition 7** *Given a TRC formula $\psi$, with $t_1, \ldots, t_k$ as free variables, the answer of $\psi$ w.r.t. a database $d$, denoted by $ans(\psi, d)$, is defined as:*

$$\{\tau_{|\{t_1,\ldots,t_k\}} \mid (d, \tau) \models \psi\}$$

Note that for an authorized TRC formula $\psi$ and a database $d$ that associate only finite relations to relation symbols, the answer $ans(\psi, d)$ is finite.

## 2.4 $\mathcal{RLT}$ Queries

We introduce $\mathcal{RLT}$ queries, which constitute the $\mathcal{RLT}$-language.

**Definition 8** *A $\mathcal{RLT}$ query is of the form:*

$$\{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k \;\psi \Rightarrow \delta\}$$

*where*

- *$X_1, \ldots, X_n$ are schema variables*
- *$t_1, \ldots, t_k$ are tuple variables over the same schema $R$*
- *$\psi$ is an authorized TRC-formula, that has exactly $t_1, \ldots, t_k$ as free variables.*
- *$\delta$ is a $\mathcal{RL}$-formula in which:*
  - *the only (free) tuple variables are $t_1, \ldots, t_k$*
  - *the only (free) schema variables are $X_1, \ldots, X_n$*
  - *there is no free attribute variable.*

*$\forall t_1 \ldots \forall t_k \;\psi \Rightarrow \delta$ is said to be the ($\mathcal{RLT}$) formula of the query.*

In order to give an idea of how querying can be done using $\mathcal{RLT}$, let us consider the following query, which finds functional dependencies in a relation $r$ over $R$:

$$Q_1 = \{\langle X, Y\rangle : R \mid \forall t \forall s (r(t) \wedge r(s)) \Rightarrow \\ (\forall A(X)(t.A = s.A)) \Rightarrow (\forall B(Y)(t.B = s.B))\}$$

Non Armstrong-compliant queries can also be expressed, such as:

$$Q_2 = \{\langle X, Y\rangle : R \mid \forall t \forall s (r(t) \wedge r(s)) \Rightarrow \\ (\exists A(X)(t.A = s.A)) \Rightarrow (\exists B(Y)(t.B = s.B))\}$$

or the following query for finding influence of genes

$$Q_3 = \{\langle X, Y\rangle : R \mid \forall t\ r(t) \wedge r(s) \wedge s.\bar{id} = t.\bar{id} + 1 \Rightarrow \\ (\forall A(X)(t.A > 0.6)) \Rightarrow (\forall B(Y)(t.B < 0.4))\}$$

assuming that $>$, $<$ and $\bar{c} = \bar{c'} + 1$ are comparisons in $\mathcal{CMP}$.

We now define the semantics of the language, first by defining the satisfaction of $\mathcal{RLT}$ query formulas, and then by defining the answers of a $\mathcal{RLT}$ query w.r.t. a given database.

**Definition 9** *A $\mathcal{RLT}$ interpretation is a triple $(d, R, \Sigma)$ where:*

- *$R \subseteq \mathcal{U}$ is a set of attributes;*
- *$d$ is a function, the database, mapping each relation symbol $r$ to a relation $\bar{r}$ over $R$;*
- *$\Sigma$, the schema interpretation, is a function mapping each schema variable $X$ to a subset of $R$.*

**Definition 10** *Let $\zeta = \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta$ be a $\mathcal{RLT}$-formula. $\zeta$ satisfies a $\mathcal{RLT}$ interpretation $(d, R, \Sigma)$, denoted $(d, R, \Sigma) \models \zeta$, if for any tuple interpretation $\tau$, and any attribute interpretation $\sigma$, if $(d, \tau) \models \psi$ then $(R, \Sigma, \sigma, \tau) \models \psi$.*

Taking any tuple interpretation corresponds to the use of $\forall$ quantifiers in $\mathcal{RLT}$ formulas. On the other hand, attribute interpretation are not important since $\delta$ does not contain free variables.

**Definition 11** *Given a database $d$ and a $\mathcal{RLT}$ query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$, the answer of $Q$ in $d$, denoted by $ans(Q, d)$ is defined as:*

*$ans(Q, d) = \{\langle \Sigma(X_1), \ldots, \Sigma(X_n)\rangle \mid (d, R, \Sigma)$ is a $\mathcal{RLT}$ interpretation and $(d, R, \Sigma) \models \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$*

## 3 BOOLEAN ENCODING

In order to find answers of a query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$ w.r.t. a database $d$, we propose an encoding of the query and the database into a boolean formula. More precisely, for each interesting tuple interpretation $\tau$, we generate a boolean formula representing the truth value of $\delta$ w.r.t. $\Sigma$. A tuple interpretation is considered to be interesting if, together with $d$, it satisfies $\psi$. The boolean formula for computing $ans(Q, d)$ is then the conjunction of the formulas for each interesting tuple interpretation.

### 3.1 Translation To Boolean Formula

**Domain** The domain, that is the boolean variables encoding an answer in $ans(Q, d)$, is defined straightforwardly as follows: for each schema variable $X \in \{X_1, \ldots, X_n\}$ and each attribute $\bar{A} \in R$, the boolean variable $p^X_{\bar{A}}$ is true whenever $\bar{A} \in \Sigma(X)$.

The following definition explains how the boolean formula is built from the $\mathcal{RL}$ formula. This definition relies on an attribute interpretation. However, this interpretation has no influence on top-level $\mathcal{RL}$ formulas, as they have no free attribute variable.

**Definition 12** *Given a $\mathcal{RL}$ formula $\delta$, a tuple interpretation $\tau$, an attribute interpretation $\sigma$ and a set of attributes $R$, the boolean encoding of $\delta$, denoted by $enc(\delta, \tau, \sigma, R)$, is inductively defined as:*

- *$enc(t.A = \bar{c}, \tau, \sigma, R) =$ **true** if $\tau(t)[\sigma(A)] = \bar{c}$, **false** otherwise*
- *$enc(t.A = s.B, \tau, \sigma, R) =$ **true** if $\tau(t)[\sigma(A)] = \tau(s)[\sigma(B)]$, **false** otherwise*
- *$enc(A = \bar{A}, \tau, \sigma, R) =$ **true** if $\sigma(A) = \bar{A}$*
- *$enc(A = B, \tau, \sigma, R) =$ **true**) if $\sigma(A) = \sigma(B)$, **false** otherwise*
- *$enc(\neg\delta, \tau, \sigma, R) = \neg enc(\delta, \tau, \sigma, R)$*
- *$enc(\delta_1 \wedge \delta_2, \tau, \sigma, R) = enc(\delta_1, \tau, \sigma, R) \wedge enc(\delta_2, \tau, \sigma, R)$*
- *$enc(\forall A(X)\delta, \tau, \sigma, R) = \bigwedge_{\bar{A} \in R}(\ p^X_{\bar{A}} \Rightarrow enc(\delta, \tau, \sigma[A := \bar{A}], R)\ )$*
- *$enc(\exists A(X)\delta, \tau, \sigma, R) = \bigvee_{\bar{A} \in R}(\ p^X_{\bar{A}} \wedge enc(\delta, \tau, \sigma[A := \bar{A}], R)\ )$*

**Definition 13** *A* boolean interpretation *$I$ is a function from boolean variables to $\{\mathbf{true}, \mathbf{false}\}$. It* satisfies *a boolean formula $\gamma$, denoted by $I \models \gamma$, if the formula obtained by replacing each variable $p$ by $I(p)$ is equivalent to* **true** *in the boolean algebra.*

**Property 1** *Let $\Sigma$ be a schema interpretation, $\delta$ a $\mathcal{RL}$ formula, $\tau$ a tuple interpretation, $\sigma$ an attribute interpretation and $R$ a set of attributes. Let $I_\Sigma$ be a boolean interpretation such that for any schema variable $X$ and any attribute $\bar{A}$, $I_\Sigma(p^X_{\bar{A}}) =$ **true** if and only if $\bar{A} \in \Sigma(X)$.*

*Then $(R, \Sigma, \sigma, \tau) \models \delta$ if and only if $I_\Sigma \models enc(\delta, \tau, \sigma, R)$.*

**Proof** By definitions 3 and 12.

**Definition 14** *The boolean encoding of a query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$ w.r.t. a database $d$, denoted by $enc(Q, d)$ is defined as:*

$$\bigwedge_{\tau \in ans(\psi, d)} enc(\delta, \tau, \sigma, R)$$

*where $\sigma$ is any attribute interpretation*[4].

**Property 2** *Let $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$ be a $\mathcal{RLT}$ query, $d$ a database and $\Sigma$ a schema interpretation. Let $I_\Sigma$ be a boolean interpretation such that for any schema variable $X$ and any attribute $\bar{A}$, $I_\Sigma(p^X_{\bar{A}} =$ **true**$)$ if and only if $\bar{A} \in \Sigma(X)$.*

*$\langle \Sigma(X_1), \ldots, \Sigma(X_n)\rangle \in ans(Q, d)$ if and only if $I_\Sigma \models enc(Q, d)$.*

[4] $\sigma$ is not actually used in the encoding since $\delta$ is closed w.r.t. attribute variables

**Proof** By definitions 11, 7 and 14, by property 1 and by remarking that if $\tau_{|\{t_1,\dots,t_k\}} = \tau'_{|\{t_1,\dots,t_k\}}$, then $enc(\delta,\tau,\sigma,R) = enc(\delta,\tau',\sigma,R)$.

## 3.2 Theoretical Complexity

The cost of evaluating a $\mathcal{RLT}$ query using a boolean formula can be evaluated by the size of the formula and its number of boolean variables. Except for quantifiers, each construction of $\mathcal{RL}$ formula generate only a constant amount of additional symbols in the encoding. For each use of a quantifier $\Box A(X)\delta$, if the size of $enc(\delta,\tau,\sigma,R)$ is $n$, then the size of $enc(\Box A(X)\delta,\tau,\sigma,R)$ is in $O(|R| \times n)$.

Let us consider a $\mathcal{RLT}$ query $Q = \{\langle X_1,\dots,X_n\rangle : R \mid \forall t_1 \dots \forall t_k\ \psi \Rightarrow \delta\}$. Let $n_{\forall\exists}$ be the maximal number of quantifiers on a branch of the abstract syntax tree of $\delta$. Then, an upper bound on the size of $enc(Q,d)$ is $O(|ans(\psi,d)| \times |\delta| \times |R|^{n_{\forall\exists}})$.

Let $time_{\psi,d}$ be the time required to evaluate $ans(\psi,d)$. Then an upper bound on the time complexity of the evaluation of a $\mathcal{RLT}$ query is $time_{\psi,d} + O(|ans(\psi,d)| \times |\delta| \times |R|^{n_{\forall\exists}} \times 2^{n\times|R|})$.

# 4 IMPLEMENTATION AND OPTIMIZATION

In this section we present the principles used in the implementation of $\mathcal{RLT}$, as well as a few optimizations that help to drastically reduce the size of generated formulas. In this section, we assume given both $\mathcal{RLT}$ query $Q = \{\langle X_1,\dots,X_n\rangle : R \mid \forall t_1 \dots \forall t_k\ \psi \Rightarrow \delta\}$ and database $d$. Figure 1 presents the different steps used in the computation of answers.

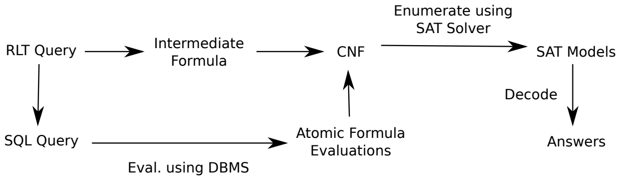

**Figure 1.** Architecture

## 4.1 Naive translation

The first step in generating $enc(Q,d)$ is to evaluate answers $ans(\psi,d)$. Since $\psi$ is an authorized TRC formula, it can evaluated using a SQL engine, at the cost of an attribute renaming to avoid attribute name clashes between tuples. We will see in section 4.3 that in the final implementation this problem disappears. Using SQL allows to easily extend the comparison predicates and expressions used in the TRC formula $\psi$. Then for each tuple combination $\tau$ we generate $enc(\delta,\tau,\sigma,R)$, with $\sigma$ being uninitialized[5].

## 4.2 Getting Answers From Boolean Formula

Because of property 2, the answers $ans(Q,d)$ can be obtained by the boolean interpretations satisfying the formula $enc(Q,d)$. We use a modified SAT-solver [6] based on Minisat [7]. Using a SAT solver requires the formula to be translated into conjunctive normal form (CNF). For this we use a linear translation based on [16]. This translation propagates constants through standard logical equivalences such as $\eta_1 \wedge \textbf{true} \equiv \eta_1$ and $\eta_1 \wedge \textbf{false} \equiv \textbf{false}$. The translation into CNF also introduce new variables. However, the value of these new variables can be deduced from the values of the variables in the original formula. This information can be transmitted to the enumerating SAT solver, allowing it to branch only on variables of the original formula. Moreover the use of modern SAT solvers allows to benefit from efficient propagation techniques, learning [12, 17] and dynamic search heuristics [17, 4]. For an extensive overview of current techniques to solve SAT, the reader is referred to [3].

## 4.3 Caching Subformulas

By looking at definition 12, one can see that the generated formula for each tuple interpretation depend on the structure of $\delta$ and on the evaluation of the atomic sub formulas of $\delta$ for (all) the possible attribute interpretations $\sigma$. That is, the actual value of tuples is not important, only the value they give to atomic formulas in $\delta$ matters.

For each atomic formula $\delta_1$ in $\delta$, we introduce a series of boolean variables $q^{\delta_1}_{\sigma}$ for all possible attribute interpretation $\sigma : \mathcal{A} \rightarrow R$ where $\mathcal{A}$ is the set of attribute variables appearing in $\delta_1$.

**Definition 15** *The* intermediate encoding $enc'(\delta,\sigma,R)$ *is inductively defined as follows:*

- $enc'(t.A = \bar{c},\sigma,R) = q^{t.A=\bar{c}}_{\sigma_{|\{A\}}}$
- $enc'(t.A = s.B,\sigma,R) = q^{t.A=s.B}_{\sigma_{|\{A,B\}}}$
- $enc'(A = \bar{A},\sigma,R) = \textbf{true}$ *if* $\sigma(A) = \bar{A}$
- $enc'(A = B,\sigma,R) = \textbf{true})$ *if* $\sigma(A) = \sigma(B)$*,* **false** *otherwise*
- $enc'(\neg\delta,\sigma,R) = \neg enc'(\delta,\sigma,R)$
- $enc'(\delta_1 \wedge \delta_2,\sigma,R) = enc'(\delta_1,\sigma,R) \wedge enc'(\delta_2,\sigma,R)$
- $enc'(\forall A(X)\delta,\sigma,R) =$ $\bigwedge_{\bar{A}\in R}(\ p^X_{\bar{A}} \Rightarrow enc'(\delta,\sigma[A := \bar{A}],R)\ )$
- $enc'(\exists A(X)\delta,\sigma,R) =$ $\bigvee_{\bar{A}\in R}(\ p^X_{\bar{A}} \wedge enc'(\delta,\sigma[A := \bar{A}],R)\ )$

This definition is similar to definition 12, except for atomic formulas where tuple variables appear. By construction, for a given tuple interpretation $\tau$, $enc(\delta,\tau,\sigma,R)$ can be obtained by replacing each variable $q^{\delta_1}_{\sigma_1}$ in $enc'(\delta,\sigma,R)$ by $enc(\delta_1,\tau,\sigma_1,R)$.

This suggests a change in the generation of $enc(Q,d)$: the SQL engine can be used to evaluate the value of atomic formulas in $\delta$ w.r.t. all relevant attribute interpretations $\sigma$, that is w.r.t. all combination of values for attribute variables appearing in the sub formula. The encoding of the $\mathcal{RL}$ formula for this tuple combination is then obtained by propagating these boolean values in the intermediate encoding.

The benefits of this change are twofold. Firstly, the comparison operator and expressions used in atomic $\mathcal{RL}$ formulas can easily be extended to any operator supported by the SQL engine. Secondly, and more importantly, given two tuple interpretations $\tau_1$ and $\tau_2$, if the evaluation of all atomic formulas w.r.t. all attribute interpretation are the same for $\tau_1$ and $\tau_2$, then $enc(\delta,\tau_1,\sigma,R) = enc(\delta,\tau_2,\sigma,R)$. Since the occurrence of

[5] In fact, we just initialize the data structure for representing the function

these two formulas are used in same conjunction, one of them is useless and can be discarded. This discarding behavior can be obtained, either by the use of the DISTINCT keyword in the SQL query, or more efficiently, by the use of a prefix tree to store and search for atomic formula evaluations. This optimization can be essential for the diminution of the size of the generated formula as shown in section 6.

## 4.4 Attribute Variable Combinatorics

Another way to reduce the size of $enc(Q, d)$ is to try to reduce the number of nested attribute quantifiers in $\delta$. We assume, without loss of generality, that each attribute variable appears exactly in one quantifier in $\delta$. The attribute quantifiers can be "pushed down" towards atomic formulas in $\delta$, by using the following standard logical equivalences:

- $\forall A(X)(\delta_1 \wedge \delta_2) \equiv (\forall A(X)\delta_1) \wedge \delta_2$ if $A$ does not appear in $\delta_2$
- $\exists A(X)\neg\delta_1 \equiv \neg\forall A(X)\delta_1$
- $\forall A(X)\forall B(Y)\delta_1 \equiv \forall B(Y)\forall A(X)\delta_1$

For example, using these equivalences $\exists A(X)\forall B(Y)(t.A = 1 \Rightarrow t.B = 1) \equiv (\forall A(X)t.A = 1) \Rightarrow (\forall B(Y)t.B = 1)$. The size of the generated formula in the second case is $O(|R|)$ smaller than the one generated in the first case.

The use of $\wedge$ commutativity and associativity may allow for more optimizations such as $\forall A(X)\forall B(Y)(\delta_1 \wedge (\delta_2 \wedge \delta_3)) \equiv \forall B(Y)(\delta_2) \wedge \forall A(X)(\delta_1 \wedge \delta_3)$ if $A$ does not appear in $\delta_2$ and $B$ does not appear in $\delta_1$ nor $\delta_3$. From this point of view, this kind of optimization can be brought near rule based optimization in relational queries [1].

# 5 REDUCING RESULT SIZE

As the search space size is $2^{n\times|R|}$, it is interesting to reduce the number of results. For example, it is usual when mining functional dependancies to output a minimal base of rules from which all rules can be inferred using Armstrong's axioms [9]. However, since our language is not supposed to be Armstrong-compliant [2], we express a wider class of queries without knowing a priori whether or not a given property is true (e.g. transitivity or reflexivity).Thus a canonical condensed representation of rules may not exist. Nevertheless, we provide means to end-users to reduce the number of results, while keeping interesting information. These means come in two flavors: firstly constraining the resulting sets of attributes, and secondly output only minimal sets (or maximal) w.r.t. set inclusion for some schema variables.

## 5.1 Constraining schema variables

The following examples illustrate how $\mathcal{RL}$ formulas can be used to constrain schema variables. Assume one wants to constraint two schema variables $X$ and $Y$, such that $\Sigma(X) \cap \Sigma(Y) = \emptyset$. This constraint can be expressed by $\forall A(X)\forall B(Y)\ \neg A = B$. The formula $\exists A(X)$ **true** imposes that $\Sigma(X)$ contains at least one attribute, while the formula $\forall A(X)\forall B(X)\ A = B$ imposes that $\Sigma(X)$ contains at most one attribute.

One can remark that $(R, \Sigma, \sigma, \tau) \models \exists A(X)X = \bar{A}$ if and only if $\bar{A} \in \Sigma(X)$. Therefore $enc(\exists A(X)X = \bar{A}, \tau, \sigma, R) \equiv p^X_{\bar{A}}$[6]. This allows constraining schema variables by using any boolean formula on the variables $p^X_{\bar{A}}$ through the $\mathcal{RL}$ formula of an $\mathcal{RLT}$ query. A consequence of this remark is that given $d$ and $Q$, the problem of determining whether $ans(Q, d) \neq \emptyset$ is NP-Hard.

## 5.2 Minimizing/Maximizing Schema Variables

Another way to reduce the number of results is to minimize or maximize schema interpretation values for some variables.

**Definition 16** *A schema interpretation* $\Sigma$ *is said to be* minimal *(resp.* maximal*) w.r.t. a schema variable* $X$*, a database* $d$ *and* $\mathcal{RLT}$ *query* $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \zeta\}$*, if there is no* $R_X$ *such that* $R_X \subset \Sigma(X)$ *(resp.* $R_X \supset \Sigma(X)$*) and* $(d, R, \Sigma[X := R_X]) \models \zeta$.

$\Sigma$ *is said to be* locally *minimal (resp. maximal) w.r.t.* $X$*,* $d$ *and* $Q$ *if there is no* $R_X$ *such that* $R_X \subset \Sigma(X)$ *with* $|R_X| = |\Sigma(X)| - 1$ *(resp.* $R_X \supset \Sigma(X)$ *with* $|R_X| = |\Sigma(X)| + 1$*) and* $(d, R, \Sigma[X := R_X]) \models \zeta$.

$Q$ *is said to be monotone (resp.* antimonotone*) w.r.t.* $X$ *if for all* $d$*,* $\Sigma$ *and* $R_X$ *such that* $\Sigma(X) \subset R_X \subseteq R$ *(resp.* $R_X \subset \Sigma(X)$*), if* $(d, R, \Sigma) \models \zeta$ *then* $(d, R, \Sigma[X := R_X]) \models \zeta$.

It is clear that if $Q$ is monotone (resp. antimonotone) w.r.t. $X$, then if $\Sigma$ is locally minimal (resp. maximal) w.r.t. $X$, $d$ and $Q$ then it is maximal (resp. minimal) w.r.t. $X$, $d$ and $Q$. We propose the following boolean encoding of the locally minimal/maximal constraint on a schema variable $X$, a database $d$ and a $\mathcal{RLT}$ query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$. Given a boolean formula $\gamma$, we denote by $\gamma[\gamma'/p]$ the formula obtained by replacing each occurrence of $p$ in $\gamma$ by $\gamma'$.

- $enc_{min}(X, d, Q) = \bigwedge_{\bar{A}\in R} p^X_{\bar{A}} \Rightarrow \neg(enc(Q, d)[\textbf{false}/p^X_{\bar{A}}])$
- $enc_{max}(X, d, Q) = \bigwedge_{\bar{A}\in R} \neg p^X_{\bar{A}} \Rightarrow \neg(enc(Q, d)[\textbf{true}/p^X_{\bar{A}}])$

The size of this constraint's boolean encoding is $|R|$ times the size of the original query's boolean encoding.

# 6 PRELIMINARY EXPERIMENTS

The CNF generator has been coded in Java, while the modified MiniSat solver in C++. We have used an embedded DBMS (Derby), since it allows to include the execution of SQL statements in CPU time results. The experiments were conducted on a 2GHz dual core Athlon processor with 3GB of RAM, running Linux.

This section presents a few experimental results on the following $\mathcal{RLT}$ query:

$$\begin{aligned}
&\{\langle X, Y\rangle : R \mid \forall t_1\forall t_2\ r(t_1) \wedge r(t_2) \Rightarrow \\
&\qquad ((\forall A(X)t_1.A = t_2.A) \Rightarrow (\forall B(Y)t_1.B = t_2.B)) \\
&\qquad \wedge(\forall A(X)\forall B(Y)\ \neg A = B) \\
&\qquad \wedge(\exists B(Y)\ \textbf{true}) \wedge (\forall B_1(Y)\forall B_2(Y)\ B_1 = B_2)\}
\end{aligned}$$

This query finds functional dependancies $X \rightarrow Y$ in $r$, $X$ and $Y$ having an empty intersection and $Y$ being a singleton. Moreover $X$ was minimized.

The relation initially contains 2013 tuples and 27 attributes. Figure 2 shows evolution of CPU time w.r.t. the number $m$

[6] This simplification is automatically performed through the propagation of **true** and **false** in the generated boolean formula.

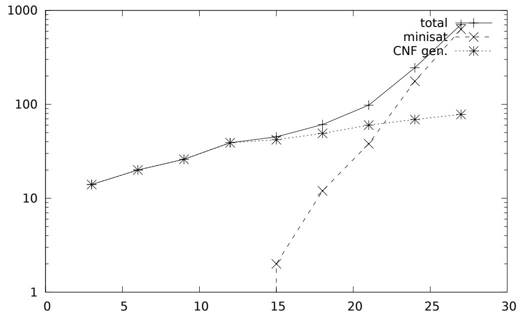

**Figure 2.** CPU time (in sec.) w.r.t. $|R|$

of attributes in $R$, *i.e.* only $m$ attributes were kept in the relation $r$. As expected, the CPU time increases exponentially w.r.t. the number of attributes, as it increases both the search space and the size of the boolean formula. Figure 3 shows the size of the generated CNF (in number of variable occurrences) and the number of attribute interpretations w.r.t. the number of tuples, the query being run on the 27 attributes of the relation. As expected, the number of interpretations grows quadratically as there are two uncorrelated tuple variables in the query. One can remark that the size of the CNF, increases slowly. This is due to the low number of useful additional tuple interpretations. Indeed the size of the generated CNF increases proportionally to the number of useful tuple interpretations. This shows the efficiency of the cache optimization on boolean formula generation.

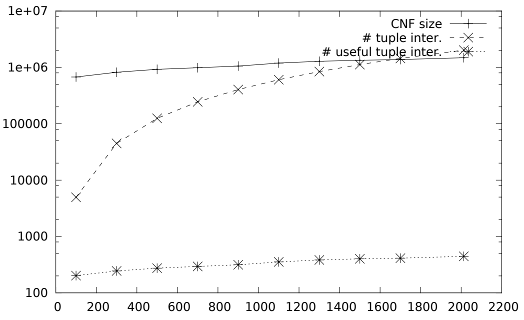

**Figure 3.** cache influence w.r.t. # tuples in $r$

## 7 CONCLUSION

We presented an ongoing work on the query language $\mathcal{RLT}$ for pattern mining. Namely, we presented the semantics of the language, as well as a translation of queries and data into a boolean formula. Implementation techniques used for implementing a query engine were presented and some experimental results show the feasibility of the approach.

Several issues remain to be explored. One the theoretical side, it would be interesting to characterize the complexity of answering $\mathcal{RLT}$ queries (e.g. the complexity of determining the emptiness of a query). One the practical side, performances of the query engine and query optimization techniques have to be investigated through a comprehensive set of databases.The performance of the current implementation could be improved either through high level optimization in order to generate better, more easy to solve, formulas, or through the elaboration of (SAT) engines dedicated to the enumeration of models of boolean formulas. An other direction of improvement is to enrich the language, for example with counting statements to be able to take into account the well-known frequency constraint in data mining. This could be treated using pseudo-boolean constraints such as in [15].

## REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison Wesley, 1995.

[2] Marie Agier, Christine Froidevaux, Jean-Marc Petit, Yoan Renaud, and Jef Wijsen, 'On Armstrong-compliant Logical Query Languages', in *4th International Workshop on Logic in Databases (LID 2011)*, pp. 33–40. ACM, (March 2011).

[3] *Handbook of Satisfiability*, eds., A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009.

[4] Armin Biere, 'Adaptive restart strategies for conflict driven SAT solvers', in *Theory and Applications of Satisfiability Testing*, pp. 28–33, (2008).

[5] Toon Calders and Jef Wijsen, 'On monotone data mining languages', in *8th International Workshop on Database Programming Languages*, (2001).

[6] Emmanuel Coquery, Saïd Jabbour, and Lakhdar Sais, 'A constraint programming approach for enumerating motifs in a sequence', in *Workshop on Declarative Pattern Mining, ICDM Workshops*, pp. 1091–1097, (2011).

[7] Niklas Eén and Niklas Sörensson. Minisat. `http://minisat.se/`.

[8] Fosca Giannotti, Giuseppe Manco, and Franco Turini, 'Towards a logic query language for data mining', in *Database Support for Data Mining Applications, LNCS 2682*, pp. 76–94, (2004).

[9] G. Gottlob and L. Libkin, 'Investigations on Armstrong relations, dependency inference, and excluded functional dependencies', *Acta Cybernetica*, **9**(4), 385–402, (1990).

[10] Tomasz Imielinski and Heikki Mannila, 'A database perspective on knowledge discovery', *Commun. ACM*, **39**(11), 58–64, (1996).

[11] Hong-Cheu Liu, Aditya Ghose, and John Zeleznikow, 'Towards an algebraic framework for querying inductive databases', in *DASFAA (2)*, pp. 306–312, (2010).

[12] Joao P. Marques-Silva and Karem A. Sakallah, 'GRASP - A New Search Algorithm for Satisfiability', in *Proceedings of International Conference on Computer-Aided Design*, pp. 220–227, (1996).

[13] Rosa Meo, Giuseppe Psaila, and Stefano Ceri, 'An extension to sql for mining association rules', *Data Min. Knowl. Discov.*, **2**(2), 195–224, (1998).

[14] Amir Netz, Surajit Chaudhuri, Jeff Bernhardt, and Usama M. Fayyad, 'Integration of data mining with database technology', in *VLDB*, pp. 719–722, (2000).

[15] Luc De Raedt, Tias Guns, and Siegfried Nijssen, 'Constraint programming for itemset mining', in *KDD*, pp. 204–212, (2008).

[16] G.S. Tseitin, 'On the complexity of derivations in the propositional calculus', in *Structures in Constructives Mathematics and Mathematical Logic, Part II*, ed., H.A.O. Slesenko, pp. 115–125, (1968).

[17] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik, 'Efficient conflict driven learning in Boolean satisfiability solver', in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 279–285, (2001).