Ideal Refinement of Descriptions in $\mathcal{AL}$-Log

Francesca A. Lisi and Donato Malerba

Dipartimento di Informatica, University of Bari, Italy
{lisi,malerba}@di.uniba.it

Abstract. This paper deals with learning in $\mathcal{AL}$-log, a hybrid language that merges the function-free Horn clause language DATALOG and the description logic $\mathcal{ALC}$. Our application context is descriptive data mining. We introduce $\mathcal{O}$-queries, a rule-based form of unary conjunctive queries in $\mathcal{AL}$-log, and a generality order $\succeq_{\mathcal{B}}$ for structuring spaces of $\mathcal{O}$-queries. We define a (downward) refinement operator $\rho_{\mathcal{O}}$ for $\succeq_{\mathcal{B}}$-ordered spaces of $\mathcal{O}$-queries, prove its ideality and discuss an efficient implementation of it in the context of interest.

1 Introduction

Hybrid systems are a special class of knowledge representation systems which are constituted by two or more subsystems dealing with distinct portions of a knowledge base and specific reasoning procedures [11]. The characterizing feature of hybrid systems is that the whole system is in charge of a single knowledge base, thus combining knowledge and reasoning services of the different subsystems in order to answer user questions. Indeed the motivation for building hybrid systems is to improve on two basic features of knowledge representation formalisms, namely *representational adequacy* and *deductive power*. Among hybrid systems, languages such as CARIN [14] and $\mathcal{AL}$-log [8] are particularly interesting because they bridge the gap between description logics (DLs) and Horn clausal logic (notoriously incomparable with respect to expressive power [3]). E.g., $\mathcal{AL}$-log combines DATALOG [5] and $\mathcal{ALC}$ [23]. Whereas learning pure DLs has been quite widely investigated [6,13,1], there are very few attempts at learning in DL-based hybrid languages. In [22] the chosen language is CARIN-$\mathcal{ALN}$, therefore example coverage and subsumption between two hypotheses are based on the existential entailment algorithm of CARIN. Following [22], Kietz studies the learnability of CARIN-$\mathcal{ALN}$, thus providing a pre-processing method which enables ILP systems to learn CARIN-$\mathcal{ALN}$ rules [12]. Closely related to DL-based hybrid systems are the proposals arising from the study of many-sorted logics, where a first-order language is combined with a sort language which can be regarded as an elementary DL [9]. In this respect the study of sorted downward refinement [10] can be also considered a contribution to learning in hybrid languages.

In this paper we deal with learning in $\mathcal{AL}$-log. This language merges DATALOG [5] and $\mathcal{ALC}$ [23] by using concept assertions essentially as *type constraints* on variables. For constrained DATALOG clauses we have defined the relation of $\mathcal{B}$-subsumption and the subsequent generality order $\succeq_{\mathcal{B}}$, and provided a decidable procedure to check $\succeq_{\mathcal{B}}$ on the basis of constrained SLD-resolution [17].

T. Horváth and A. Yamamoto (Eds.): ILP 2003, LNAI 2835, pp. 215–232, 2003.

This work presents a case study for $\mathcal{B}$-subsumption in the context of *descriptive data mining*. As opposite to prediction, description focuses on finding human-interpretable patterns describing a data set $\mathbf{r}$. Among descriptive tasks, frequent pattern discovery aims at the extraction of all patterns whose cardinality exceeds a user-defined threshold. Indeed each pattern is considered as an intensional description (expressed in a given language $\mathcal{L}$) of a subset of $\mathbf{r}$. We propose a variant of this task which takes concept hierarchies into account during the discovery process, thus yielding descriptions of $\mathbf{r}$ at multiple granularity levels. More formally, given

- a data set $\mathbf{r}$ including a taxonomy $\mathcal{T}$ where a reference concept and task-relevant concepts are designated,
- a set $\{\mathcal{L}^l\}_{1 \leq l \leq maxG}$ of languages
- a set $\{minsup^l\}_{1 \leq l \leq maxG}$ of support thresholds

the problem of *frequent pattern discovery at l levels of description granularity*, $1 \leq l \leq maxG$, is to find the set $\mathcal{F}$ of all the patterns $P \in \mathcal{L}^l$ frequent in $\mathbf{r}$, namely P's with support s such that (i) $s \geq minsup^l$ and (ii) all ancestors of P w.r.t. $\mathcal{T}$ are frequent. An ILP approach to this problem requires the specification of a language $\mathcal{L}$ of hypotheses and a generality relation $\succeq$ for $\mathcal{L}$. To this aim we introduce $\mathcal{O}$-queries, a rule-based form of unary conjunctive queries in $\mathcal{AL}$-log, and study $\succeq_{\mathcal{B}}$-ordered spaces of $\mathcal{O}$-queries. Descriptive data mining problems are characterized by hypothesis spaces with *high solution density*. Ideal refinement operators are usually suggested to search spaces of this kind [2]. Unfortunately for clausal languages ordered by θ-subsumption or stronger orders, ideal operators have been proven not to exist [20]. Yet they can be defined in some restricted yet meaningful cases. The main contribution of this paper is the definition of an ideal downward refinement operator $\rho_{\mathcal{O}}$ for $\succeq_{\mathcal{B}}$-ordered spaces of $\mathcal{O}$-queries in the context of frequent pattern discovery at multiple levels of description granularity.

The paper is organized as follows. Section 2 introduces the basic notions of $\mathcal{AL}$-log. Section 3 defines the space of $\mathcal{O}$-queries organized according to $\mathcal{B}$-subsumption. Section 4 presents the refinement operator $\rho_{\mathcal{O}}$, proves its ideality and discusses an efficient implementation of it in the context of interest. Section 5 concludes the paper with final remarks.

2 $\mathcal{AL}$-Log in a Nutshell

The language $\mathcal{AL}$-log [8] combines the representation and reasoning means offered by DATALOG and $\mathcal{ALC}$. Indeed it embodies two subsystems, called *relational* and *structural*. We assume the reader to be familiar with DATALOG, therefore we focus on the structural subsystem and hybridization of the relational subsystem.

2.1 The Structural Subsystem

The structural subsystem of $\mathcal{AL}$-log allows for the specification of structural knowledge in terms of *concepts*, *roles*, and *individuals*. Individuals represent objects in the domain of interest. Concepts represent classes of these objects, while roles represent binary relations between concepts. Complex concepts can be defined by means of constructs, such as $\sqcap$ and $\sqcup$. The structural subsystem is itself a two-component system. The *intensional* component $\mathcal{T}$ consists of concept hierarchies spanned by is-a relations between concepts, namely *inclusion statements* of the form $C \sqsubseteq D$ (read "C is included in D") where C and D are two arbitrary concepts. The *extensional* component $\mathcal{M}$ specifies instance-of relations, e.g. *concept assertions* of the form $a : C$ (read "the individual a belongs to the concept C") and *role assertions* of the form aRb (read "the individual a is related to the individual b by means of the role R").

In $\mathcal{ALC}$ knowledge bases, an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *interpretation function* of $\mathcal{I}$). E.g., it maps concepts to subsets of $\Delta^{\mathcal{I}}$ and individuals to elements of $\Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (see *unique names* assumption [21]). We say that $\mathcal{I}$ is a *model* for $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, for $a : C$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and for aRb if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

The main reasoning mechanism for the structural component is the satisfiability check. The *tableau calculus* proposed in [8] starts with the tableau branch $S = \mathcal{T} \cup \mathcal{M}$ and adds assertions to S by means of *propagation rules* such as

- $S \rightarrow_{\sqcup} S \cup \{s : D\}$ if
  1. $s : C_1 \sqcup C_2$ is in S,
  2. $D = C_1$ and $D = C_2$,
  3. neither $s : C_1$ nor $s : C_2$ is in S
- $S \rightarrow_{\forall} S \cup \{t : C\}$ if
  1. $s : \forall R.C$ is in S,
  2. sRt is in S,
  3. $t : C$ is not in S
- $S \rightarrow_{\sqsubseteq} S \cup \{s : C' \sqcup D\}$ if
  1. $C \sqsubseteq D$ is in S,
  2. s appears in S,
  3. C' is the NNF concept equivalent to $\neg C$
  4. $s : \neg C \sqcup D$ is not in S
- $S \rightarrow_{\perp} \{s : \perp\}$ if
  1. $s : A$ and $s : \neg A$ are in S, or
  2. $s : \neg\top$ is in S,
  3. $s : \perp$ is not in S

until either a contradiction is generated or an interpretation satisfying S can be easily obtained from it.

2.2 Hybridization of the Relational Subsystem

The relational part of $\mathcal{AL}$-log allows one to define DATALOG programs enriched with *constraints* of the form $s : C$ where s is either a constant or a variable, and C is an $\mathcal{ALC}$-concept. Note that the usage of concepts as typing constraints applies only to variables and constants that already appear in the clause. The symbol & separates constraints from DATALOG atoms in a clause.

Definition 1. *A* constrained DATALOG clause *is an implication of the form* $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_m \& \gamma_1, \ldots, \gamma_n$ *where* $m \geq 0$, $n \geq 0$, α_i *are* DATALOG *atoms and* γ_j *are constraints. A* constrained DATALOG program Π *is a set of constrained* DATALOG *clauses.*

An $\mathcal{AL}$*-log knowledge base* $\mathcal{B}$ is the pair $\langle \Sigma, \Pi \rangle$ where Σ is an $\mathcal{ALC}$ knowledge base and Π is a constrained DATALOG program. For a knowledge base to be acceptable, it must satisfy the following conditions:

- The set of DATALOG predicate symbols appearing in Π is disjoint from the set of concept and role symbols appearing in Σ.
- The alphabet of constants in Π coincides with the alphabet $\mathcal{O}$ of the individuals in Σ. Furthermore, every constant in Π appears also in Σ.
- For each clause in Π, each variable occurring in the constraint part occurs also in the DATALOG part.

These properties allow for the extension of terminology and results related to the notion of *substitution* from DATALOG to $\mathcal{AL}$-log in a straightforward manner.

Example 1. As a running example, we consider an $\mathcal{AL}$-log knowledge base $\mathcal{B}$ obtained from the N$\frac{\text{ORTHWIN}}{\text{TRADERS}}$D database. The structural subsystem Σ should reflect the E/R model underlying the N$\frac{\text{ORTHWIN}}{\text{TRADERS}}$D database. To serve our illustrative purpose we focus on the concepts (entities) `Order`, `Product` and `Customer`. The intensional part of Σ encompasses inclusion statements such as `DairyProduct` $\sqsubset$ `Product` and `EuroCustomer`$\equiv$`Customer`$\sqcap\exists$`LivesIn.EuroCountry` that define two taxonomies, one for `Product` and the other one for `Customer`. The extensional part of Σ contains assertions like `order10248:Order`, `product11:DairyProduct`, `'VINET'LivesIn'France'` and `'France':EuroCountry`. The relational subsystem Π expresses the N$\frac{\text{ORTHWIN}}{\text{TRADERS}}$D database as a constrained DATALOG program. We restrict ourselves to the relations Order and OrderDetail. The extensional part of Π consists of facts such as `order(order10248,'VINET',...)` whereas the intensional part defines two views on `order` and `orderDetail`:

```
item(OrderID,ProductID) ← orderDetail(OrderID,ProductID,_,_,_)
                          & OrderID:Order, ProductID:Product
purchaser(OrderID,CustomerID) ← order(OrderID,CustomerID,_,...,_)
                          & OrderID:Order, CustomerID:Customer
```

that, when triggered on $\mathcal{B}$, can deduce facts such as `item(order10248, product11)` and `purchaser(order10248,'VINET')`.

The interaction between the structural and the relational part of an $\mathcal{AL}$-log knowledge base is also at the basis of a model-theoretic semantics for $\mathcal{AL}$-log. We call Π_D the set of DATALOG clauses obtained from the clauses of Π by deleting their constraints. We define an *interpretation* $\mathcal{J}$ for $\mathcal{B}$ as the union of an $\mathcal{O}$-interpretation $\mathcal{I}_\mathcal{O}$ for Σ (i.e. an interpretation compliant with the unique names assumption) and an Herbrand interpretation $\mathcal{I}_\mathcal{H}$ for Π_D. An interpretation $\mathcal{J}$ is a *model* of $\mathcal{B}$ if $\mathcal{I}_\mathcal{O}$ is a model of Σ, and for each ground instance $\bar{\alpha}'\&\gamma'_1, \ldots, \gamma'_n$ of each clause $\bar{\alpha}\&\gamma_1, \ldots, \gamma_n$ in Π, either there exists one γ'_i, $i \in \{1, \ldots, n\}$, that is not satisfied by $\mathcal{J}$, or $\bar{\alpha}'$ is satisfied by $\mathcal{J}$. The notion of *logical consequence* paves the way to the definition of answer set for queries. *Queries* to $\mathcal{AL}$-log knowledge bases are special cases of Definition 1. An *answer* to the query Q is a ground substitution σ for the variables in Q. The answer σ is *correct* w.r.t. a $\mathcal{AL}$-log knowledge base $\mathcal{B}$ if $Q\sigma$ is a logical consequence of $\mathcal{B}$ ($\mathcal{B} \models Q\sigma$). The *answer set* of Q in $\mathcal{B}$ contains all the correct answers to Q w.r.t. $\mathcal{B}$.

Reasoning for $\mathcal{AL}$-log knowledge bases is based on *constrained SLD-resolution* [8], i.e. an extension of SLD-resolution to deal with constraints. In particular, the constraints of the resolvent of a query Q and a constrained DATALOG clause E are recursively simplified by replacing couples of constraints $t : C$, $t : D$ with the equivalent constraint $t : C \sqcap D$. The one-to-one mapping between constrained SLD-derivations and the SLD-derivations obtained by ignoring the constraints is exploited to extend known results for DATALOG to $\mathcal{AL}$-log. Note that in $\mathcal{AL}$-log a derivation of the empty clause with associated constraints does not represent a refutation. It actually infers that the query is true in those models of $\mathcal{B}$ that satisfy its constraints. Therefore in order to answer a query it is necessary to collect enough derivations ending with a constrained empty clause such that every model of $\mathcal{B}$ satisfies the constraints associated with the final query of at least one derivation.

Definition 2. *Let $Q^{(0)}$ be a query $\leftarrow \beta_1, \ldots, \beta_m\&\gamma_1, \ldots, \gamma_n$ to a $\mathcal{AL}$-log knowledge base $\mathcal{B}$. A* constrained SLD-refutation *for $Q^{(0)}$ in $\mathcal{B}$ is a finite set $\{d_1, \ldots, d_s\}$ of constrained SLD-derivations for $Q^{(0)}$ in $\mathcal{B}$ such that:*

1. *for each derivation d_i, $1 \leq i \leq s$, the last query $Q^{(n_i)}$ of d_i is a constrained empty clause;*
2. *for every model $\mathcal{J}$ of $\mathcal{B}$, there exists at least one derivation d_i, $1 \leq i \leq s$, such that $\mathcal{J} \models Q^{(n_i)}$*

Constrained SLD-refutation is a complete and sound method for answering *ground* queries. An answer σ to a query Q is a *computed answer* if there exists a constrained SLD-refutation for $Q\sigma$ in $\mathcal{B}$ ($\mathcal{B} \vdash Q\sigma$). The set of computed answers is called the *success set* of Q in $\mathcal{B}$. Furthermore, given *any* query Q, the success set of Q in $\mathcal{B}$ coincides with the answer set of Q in $\mathcal{B}$. This provides an operational means for computing correct answers to queries. Indeed, it is straightforward to see that the usual reasoning methods for DATALOG allow us to collect in a finite number of steps enough constrained SLD-derivations for Q in $\mathcal{B}$ to construct a refutation - if any. Derivations must satisfy both conditions of Definition 2. In particular, the latter requires some reasoning on the structural

component of $\mathcal{B}$. This is done by applying the tableau calculus as shown in the following example.

Example 2. Following Example 1, we compute a correct answer to

$Q = \leftarrow$ `purchaser(order10248,Y)` & `order10248:Order`, `Y:EuroCustomer`

w.r.t. $\mathcal{B}$. A refutation for $Q = Q^{(0)}$ consists of the following single constrained SLD-derivation. Let $E^{(1)}$ be

```
purchaser(OrderID,CustomerID)← order(OrderID,CustomerID,_,...,_)
                                & OrderID:Order, CustomerID:Customer
```

A resolvent for $Q^{(0)}$ and $E^{(1)}$ with substitution $\sigma^{(1)} = \{$`OrderID/ order10248`, `CustomerID/ Y`$\}$ is the query

$Q^{(1)} = \leftarrow$ `order(order10248,Y,_,...,_)`
& `order10248:Order`, `Y:EuroCustomer`

Let $E^{(2)}$ be `order(order10248,'VINET',_,...,_)`. A resolvent for $Q^{(1)}$ and $E^{(2)}$ with substitution $\sigma^{(2)} = \{$`Y/ 'VINET'`$\}$ is the constrained empty clause

$Q^{(2)} = \leftarrow$ & `order10248:Order`, `'VINET':EuroCustomer`

What we need to check is that $\Sigma \cup \{$`order10248:Order`, `'VINET': EuroCustomer`$\}$ is satisfiable. This check amounts to two unsatisfiability checks to be performed by applying the tableau calculus. The first check operates on the initial tableau $S^{(0)} = \Sigma \cup \{$`order10248:`$\neg$`Order`$\}$. The application of the propagation rule $\rightarrow_{\bot}$ to $S^{(0)}$ produces the tableau $S^{(1)} = \{$`order10248:`$\bot\}$. Computation stops here because no other rule can be applied to $S^{(1)}$. Since $S^{(1)}$ is complete and contains a clash, the initial tableau $S^{(0)}$ is unsatisfiable. The second check operates on the initial tableau $S^{(0)} = \Sigma \cup \{$`'VINET':`$\neg$`EuroCustomer`$\}=\Sigma$ $\cup \{$`'VINET':`$\neg$`Customer`$\sqcup\forall$ `LivesIn.(`$\neg$`EuroCountry)`$\}$. By applying $\rightarrow_{\sqcup}$ w.r.t. $\forall$`LivesIn.(`$\neg$`EuroCountry`) to $S^{(0)}$ we obtain $S^{(1)}=\Sigma \cup \{$`'VINET':`$\forall$ `LivesIn.(`$\neg$`EuroCountry)`$\}$. The only propagation rule applicable to $S^{(1)}$ is $\rightarrow_{\forall}$ which yields the tableau $S^{(2)} = \Sigma \cup \{$`'VINET':(`$\neg$`EuroCountry)`$\}$. It presents a contradiction. Indeed the application of $\rightarrow_{\bot}$ to $S^{(2)}$ produces the final tableau $S^{(3)}$ $= \{$`'VINET':`$\bot\}$.

These two results together prove the satisfiability of $\Sigma \cup \{$`order10248:Order`, `'VINET':EuroCustomer`$\}$, then the correcteness of $\sigma=\{$`Y/ 'VINET'`$\}$ as an answer to Q w.r.t. $\mathcal{B}$.

3 The $\succeq_{\mathcal{B}}$-Ordered Space of $\mathcal{O}$-Queries

In this section we propose $\mathcal{AL}$-log as the starting point for the definition of a knowledge representation and reasoning framework in the context of interest. The main feature of this framework is the extension of the *unique names assumption* from the semantic level to the syntactic one. We would like to remind the reader that this assumption holds in $\mathcal{ALC}$. Also it holds naturally for ground constrained DATALOG clauses because the semantics of $\mathcal{AL}$-log adopts Herbrand

models for the DATALOG part and $\mathcal{O}$-models for the constraint part. Conversely it is not guaranteed in the case of non-ground constrained DATALOG clauses, e.g. different variables can be unified. In particular we resort to the bias of *Object Identity* [24]: In a formula, terms denoted with different symbols must be distinct, i.e. they represent different entities of the domain. This bias yields to a restricted form of substitution whose bindings avoid the identification of terms: A substitution σ is an *OI-substitution* w.r.t. a set of terms T iff $\forall t_1, t_2 \in T\colon\ t_1 \neq t_2$ yields that $t_1\sigma \neq t_2\sigma$. From now on, we assume that substitutions are OI-compliant. See [16] for an investigation of OI in the case of DATALOG queries.

In this framework descriptions are represented as $\mathcal{O}$-queries, a rule-based form of unary conjunctive queries whose answer set contains individuals of an $\mathcal{ALC}$ concept $\hat{C}$ of reference.

Definition 3. *Given a reference concept $\hat{C}$, an* $\mathcal{O}$-query Q *to an $\mathcal{AL}$-log knowledge base $\mathcal{B}$ is a constrained* DATALOG *clause of the form*

$$Q = q(X) \leftarrow \alpha_1, \ldots, \alpha_m \& X : \hat{C}, \gamma_2, \ldots, \gamma_n$$

where X is the distinguished variable *and the remaining variables occurring in the body of Q are the* existential variables. *We denote by key(Q) the key constraint $X : \hat{C}$ of Q. A* trivial $\mathcal{O}$*-query is a constrained empty clause of the form $q(X) \leftarrow \& X : \hat{C}$.*

We impose $\mathcal{O}$-queries to be linked and connected (or range-restricted) constrained DATALOG clauses. The language $\mathcal{L}$ of descriptions for a given learning problem is implicitly defined by a set $\mathcal{A}$ of atom templates, a key constraint $\hat{\gamma}$, and an additional set Γ of constraint templates. An atom template α specify name and arity of the predicate and mode of its arguments. An instantiation of α is a DATALOG atom with predicate and arguments that fulfill the requirements specified in α. Constraint templates specify the concept name for concept assertions and determine the granularity level l of descriptions.

Example 3. Following Example 1, suppose that we want to perform sales analysis by finding associations between the category of ordered products and the geographic location of the customer within orders. Here the entity `Order` is the reference concept, and the entities `Product` and `Customer` are task-relevant concepts. The language $\mathcal{L}$ must be defined so that it can generate descriptions of orders with respect to products and customers. To this aim let $\mathcal{A}$={`item(+,-)`, `purchaser(+,-)`} and $\hat{\gamma}$ be the key constraint built on the concept `Order`. Suppose that we are interested in descriptions at two different granularity levels. Thus $\mathcal{T}$ consists of the two layers $\mathcal{T}^1$={`Product`, `Customer`} and $\mathcal{T}^2$={`Beverage`, `Condiment`, `Confection`, `DairyProduct`, `GrainsCereals`, `MeatPoultry`, `Produce`, `SeaFood`, `EuroCustomer`, `NorthAmericanCustomer`, `SouthAmericanCustomer`} from which the sets Γ^1 and Γ^2 of constraints are derived. Examples of $\mathcal{O}$-queries belonging to this language are:

Q_0= `q(X)` ← & `X:Order`
Q_1= `q(X)` ← `item(X,Y)` & `X:Order`
Q_2= `q(X)` ← `purchaser(X,Y)` & `X:Order`
Q_3= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Product`
Q_4= `q(X)` ← `purchaser(X,Y)` & `X:Order`, `Y:Customer`
Q_5= `q(X)` ← `item(X,Y)`, `item(X,Z)` & `X:Order`, `Y:Product`
Q_6= `q(X)` ← `item(X,Y)`, `purchaser(X,Z)` & `X:Order`, `Y:Product`
Q_7= `q(X)` ← `item(X,Y)`, `item(X,Z)` & `X:Order`, `Y:Product`, `Z:Product`
Q_8= `q(X)` ← `item(X,Y)`, `purchaser(X,Z)` & `X:Order`, `Y:Product`, `Z:Customer`
Q_9= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:DairyProduct`
Q_{10}= `q(X)` ← `purchaser(X,Y)` & `X:Order`, `Y:EuroCustomer`
Q_{11}= `q(X)` ← `item(X,Y)`, `item(X,Z)` & `X:Order`, `Y:DairyProduct`
Q_{12}= `q(X)` ← `item(X,Y)`, `purchaser(X,Z)` & `X:Order`, `Y:DairyProduct`
Q_{13}= `q(X)` ← `item(X,Y)`, `item(X,Z)`
& `X:Order`, `Y:DairyProduct`, `Z:GrainsCereals`
Q_{14}= `q(X)` ← `item(X,Y)`, `purchaser(X,Z)`
& `X:Order`, `Y:DairyProduct`, `Z:EuroCustomer`

In particular, Q_0 and Q_1 are valid for both $\mathcal{L}^1$ and $\mathcal{L}^2$, Q_3 and Q_5 belong to $\mathcal{L}^1$, and Q_9 belongs to $\mathcal{L}^2$. Note that all of them are linked and connected.

An *answer* to an $\mathcal{O}$-query Q is a ground substitution θ for the distinguished variable of Q. The aforementioned conditions of well-formedness guarantee that the evaluation of $\mathcal{O}$-queries is sound according to the following notions of answer set and success set.

Definition 4. *Let $\mathcal{B}$ be a $\mathcal{AL}$-log knowledge base. An answer θ to an $\mathcal{O}$-query Q is a* correct (resp. computed) answer *w.r.t. $\mathcal{B}$ if there exists at least one correct (resp. computed) answer to body$(Q)\theta$ w.r.t. $\mathcal{B}$.*

Example 4. Following Example 2 and 3, the substitution θ = {`X/order10248`} is a correct answer to Q_{10} w.r.t. $\mathcal{B}$ because there exists a correct answer σ={`Y/'VINET'`} to *body*$(Q_{10})\theta$ w.r.t. $\mathcal{B}$.

The definition of a generality order for structuring the space of $\mathcal{O}$-queries can not disregard the nature of $\mathcal{O}$-queries as a special case of constrained DATALOG clauses as well as the availability of an $\mathcal{AL}$-log knowledge base with respect to which these $\mathcal{O}$-queries are to be evaluated. For constrained DATALOG clauses we have defined the relation of $\mathcal{B}$-subsumption [17]. It adapts generalized subsumption [4] to the $\mathcal{AL}$-log framework.

Definition 5. *Let P, Q be two constrained* DATALOG *clauses and $\mathcal{B}$ an $\mathcal{AL}$-log knowledge base. We say that P* $\mathcal{B}$-subsumes *Q, $P \succeq_{\mathcal{B}} Q$, if for every model $\mathcal{J}$ of $\mathcal{B}$ and every ground atom α such that Q covers α under $\mathcal{J}$, we have that P covers α under $\mathcal{J}$.*

We have proved that $\succeq_{\mathcal{B}}$ is a quasi-order for constrained DATALOG clauses and provided a decidable procedure to check $\succeq_{\mathcal{B}}$ on the basis of constrained SLD-resolution [15]. Note that the underlying reasoning mechanism of $\mathcal{AL}$-log makes $\mathcal{B}$-subsumption more powerful than generalized subsumption.

Theorem 1. *Let P, Q be two constrained* DATALOG *clauses, $\mathcal{B}$ an $\mathcal{AL}$-log knowledge base and σ a Skolem substitution for Q with respect to $\{P\} \cup \mathcal{B}$. We say that $P \succeq_{\mathcal{B}} Q$ iff there exists a substitution θ for P such that (i) $head(P)\theta = head(Q)$ and (ii) $\mathcal{B} \cup body(Q)\sigma \vdash body(P)\theta\sigma$ where $body(P)\theta\sigma$ is ground.*

Theorem 2. *Checking $\succeq_{\mathcal{B}}$ in $\mathcal{AL}$-log is decidable.*

Example 5. Following Example 3, we illustrate the test procedure of Theorem 1 on the pair Q_8, Q_{14} of $\mathcal{O}$-queries to check whether $Q_8 \succeq_{\mathcal{B}} Q_{14}$ holds. Let $\sigma = \{\texttt{X/a}, \texttt{Y/b}, \texttt{Z/c}\}$ a Skolem substitution for Q_{14} with respect to $\mathcal{B} \cup \{Q_8\}$ and θ the identity substitution for Q_8. The condition (i) is immediately verified. It remains to verify that (ii) $\mathcal{B} \cup \{\texttt{item(a,b)}, \texttt{purchaser(a,c)} \,\&\, \texttt{a:Order}, \texttt{b:DairyProduct}, \texttt{c:EuroCustomer}\} \models \texttt{item(a,b)}, \texttt{purchaser(a,c)} \,\&\, \texttt{a:Order}, \texttt{b:Product}, \texttt{c:Customer}$. We try to build a constrained SLD-refutation for

$$Q^{(0)} = \leftarrow \texttt{item(a,b)}, \texttt{purchaser(a,c)} \,\&\, \texttt{a:Order}, \texttt{b:Product}, \texttt{c:Customer}$$

in $\mathcal{B}' = \mathcal{B} \cup \{\texttt{item(a,b)}, \texttt{purchaser(a,c)}, \texttt{b:DairyProduct}, \texttt{c:EuroCustomer}, \texttt{a:Order}\}$. Once the constrained empty clause has been obtained by means of classical SLD-resolution, we need to check whether $\Sigma' \cup \{\texttt{a:Order}, \texttt{b:Product}, \texttt{c:Customer}\}$ is satisfiable. The first unsatisfiability check operates on the initial tableau $S^{(0)} = \Sigma' \cup \{\texttt{a:}\neg\texttt{Order}\}$. The application of the propagation rule $\rightarrow_{\bot}$ to $S^{(0)}$ produces the tableau $S^{(1)} = \{\texttt{a:}\bot\}$. Computation stops here because no other rule can be applied to $S^{(1)}$. Since $S^{(1)}$ is complete and contains a clash, the initial tableau $S^{(0)}$ is unsatisfiable. The second unsatisfiability check operates on the initial tableau $S^{(0)} = \Sigma' \cup \{\texttt{b:}\neg\texttt{Product}\}$. The only propagation rule applicable to $S^{(0)}$ is $\rightarrow_{\sqsubseteq}$ with respect to the assertion $\texttt{DairyProduct} \sqsubseteq \texttt{Product}$. It produces the tableau $S^{(1)} = \Sigma \cup \{\texttt{b:}\neg\texttt{Product}, \texttt{b:}\neg\texttt{DairyProduct} \sqcup \texttt{Product}\}$. By applying $\rightarrow_{\sqcup}$ to $S^{(1)}$ with respect to the concept `Product` we obtain $S^{(2)} = \Sigma \cup \{\texttt{b:}\neg\texttt{Product}, \texttt{b:Product}\}$ which presents an evident contradiction. Indeed the application of $\rightarrow_{\bot}$ to $S^{(2)}$ produces the final tableau $S^{(3)} = \{\texttt{b:}\bot\}$. The third unsatisfiability check operates on the initial tableau $S^{(0)} = \Sigma' \cup \{\texttt{b:}\neg\texttt{Customer}\}$. Remember that Σ' contains the assertion $\texttt{c:Customer} \sqcap \exists\ \texttt{LivesIn.EuroCountry}$. By applying $\rightarrow_{\sqcap}$ to $S^{(0)}$ we obtain $S^{(1)}$ which encompasses the contradiction $\{\texttt{c:}\neg\texttt{Customer}, \texttt{c:Customer}\}$. Indeed the application of $\rightarrow_{\bot}$ to $S^{(1)}$ produces the final tableau $S^{(2)} = \{\texttt{b:}\bot\}$.

Having proved the satisfiability of $\Sigma' \cup \{\texttt{a:Order}, \texttt{b:Product}, \texttt{c:Customer}\}$, we have proved the existence of a constrained SLD-refutation for $Q^{(0)}$ in $\mathcal{B}'$. Therefore we can say that $Q_8 \succeq_{\mathcal{B}} Q_{14}$.

4 The Refinement Operator $\rho_{\mathcal{O}}$

The space $(\mathcal{L}, \succeq_{\mathcal{B}})$ is a quasi-ordered set, therefore it can be searched by refinement operators [20]. In the application context of interest, the refinement operator being defined must enable the search through multiple spaces, each of which corresponds to a different level of description granularity. Furthermore we restrict our investigation to downward refinement operators because the search towards finer-grained descriptions is more efficient.

Definition 6. *Let $\gamma_1 = t_1 : C$ and $\gamma_2 = t_2 : D$ two $\mathcal{ALC}$ constraints. We say that γ_1 is* at least as strong as *γ_2, denoted as $\gamma_1 \succeq \gamma_2$, if and only if $t_1 = t_2$ and $C \sqsubseteq D$. Furthermore γ_1 is* stronger than *γ_2, denoted as $\gamma_1 \succ \gamma_2$, if and only if $t_1 = t_2$ and $C \sqsubset D$.*

Definition 7. *Let $\mathcal{L} = \{\mathcal{L}^l\}_{1 \leq l \leq maxG}$ be a language of $\mathcal{O}$-queries. A (downward) refinement operator $\rho_{\mathcal{O}}$ for $(\mathcal{L}, \succeq_{\mathcal{B}})$ is defined such that, for a given $\mathcal{O}$-query $P = q(X) \leftarrow \alpha_1, \ldots, \alpha_m \& X : \hat{C}, \gamma_2, \ldots, \gamma_n$ in $\mathcal{L}^l$, $l < maxG$, the set $\rho_{\mathcal{O}}(P)$ contains all $Q \in \mathcal{L}$ that can be obtained by applying one of the following refinement rules:*

$\langle Atom \rangle$ *$Q = q(X) \leftarrow \alpha_1, \ldots, \alpha_m, \alpha_{m+1} \& X : \hat{C}, \gamma_2, \ldots, \gamma_n$ where α_{m+1} is an instantiation of an atom template in $\mathcal{A}$ such that $\alpha_{m+1} \notin body(P)$.*

$\langle Constr \rangle$ *$Q = q(X) \leftarrow \alpha_1, \ldots, \alpha_m \& X : \hat{C}$, γ_2, …, γ_n, γ_{n+1} where γ_{n+1} is an instantiation of a constraint template in Γ^l such that γ_{n+1} constrains an unconstrained variable in $body(P)$.*

$\langle \forall C \rangle$ *$Q = q(X) \leftarrow \alpha_1, \ldots, \alpha_m \& X : \hat{C}, \gamma'_2, \ldots, \gamma'_n$ where each γ'_j, $2 \leq j \leq n$, is an instantiation of a constraint template in Γ^{l+1} such that $\gamma'_j \succeq \gamma_j$ and at least one $\gamma'_j \succ \gamma_j$.*

The rules $\langle Atom \rangle$ and $\langle Constr \rangle$ help moving within the space $\mathcal{L}^l$ (*intra-space search*) whereas the rule $\langle \forall C \rangle$ helps moving from $\mathcal{L}^l$ to $\mathcal{L}^{l+1}$ (*inter-space search*). Both rules are correct, i.e. the Q's obtained by applying any of these rules to $P \in \mathcal{L}^l$ are such that $P \succeq_{\mathcal{B}} Q$. This can be proved intuitively by observing that they act only on $body(P)$. Thus condition (i) of Theorem 1 is satisfied. Furthermore, it is straightforward to notice that the application of $\rho_{\mathcal{O}}$ to P reduces the number of models of P in both cases. In particular, as for $\langle \forall C \rangle$, this intuition follows from the definition of $\mathcal{O}$-model. So condition (ii) also is fulfilled.

From now on we call *k-patterns* those patterns $Q \in \rho^k_{\mathcal{O}}(P)$ that have been generated after k refinement steps starting from the trivial $\mathcal{O}$-query $\widehat{P}$ in $\mathcal{L}^l$ and applying either $\langle Atom \rangle$ or $\langle Constr \rangle$.

Example 6. Each edge in Figure 1 indicates the application of only one of the rules defined for $\rho_{\mathcal{O}}$ to $\mathcal{O}$-queries listed in Example 3. E.g., $\rho_{\mathcal{O}}(Q_1)$ is the set

Q'_1= `q(X)` ← `item(X,Y)`, `item(X,Z)` & `X:Order`
Q'_2= `q(X)` ← `item(X,Y)`, `purchaser(X,Z)` & `X:Order`
Q'_3= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Product`
Q'_4= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Customer`
Q'_5= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Beverage`
Q'_6= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Condiment`
Q'_7= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Confection`
Q'_8= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:DairyProduct`
Q'_9= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:GrainsCereals`
Q'_{10}= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:MeatPoultry`
Q'_{11}= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Produce`
Q'_{12}= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:SeaFood`

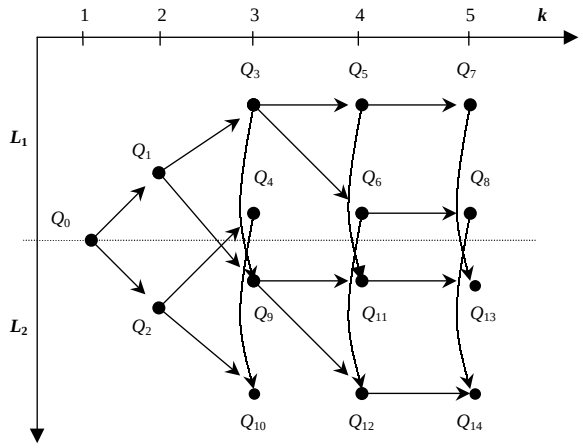

Fig. 1. Portion of the refinement graph of $\rho_{\mathcal{O}}$ in $\mathcal{L}$.

Q'_{13}= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:EuroCustomer`
Q'_{14}= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:NorthAmericanCustomer`
Q'_{15}= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:SouthAmericanCustomer`

where Q'_1 and Q'_2 are generated by means of $\langle Atom \rangle$, Q'_3 and Q'_4 by means of $\langle Constr \rangle$, and the $\mathcal{O}$-queries from Q'_5 to Q'_{15} also by means of $\langle Constr \rangle$ (but considering Q_1 as belonging to $\mathcal{L}^2$). Note that Q'_4, Q'_{13}, Q'_{14}, and Q'_{15} will turn out to be infrequent. Yet they are generated. What matters while searching $(\mathcal{L}, \succeq_{\mathcal{B}})$ is to find patterns that are more specific than a given P under $\mathcal{B}$-subsumption. Conversely, $\rho_{\mathcal{O}}(Q_3)$ is the set

Q''_1= `q(X)` ← `item(X,Y)`, `item(X,Z)` & `X:Order`, `Y:Product`
Q''_2= `q(X)` ← `item(X,Y)`, `purchaser(X,Z)` & `X:Order`, `Y:Product`
Q''_3= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Beverage`
Q''_4= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Condiment`
Q''_5= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Confection`
Q''_6= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:DairyProduct`
Q''_7= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:GrainsCereals`
Q''_8= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:MeatPoultry`
Q''_9= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:Produce`
Q''_{10}= `q(X)` ← `item(X,Y)` & `X:Order`, `Y:SeaFood`

where the $\mathcal{O}$-queries Q''_1 and Q''_2 are generated by means of $\langle Atom \rangle$, and the $\mathcal{O}$-queries from Q''_3 to Q''_{10} by means of $\langle \forall C \rangle$. Note that the query Q_9 can be obtained by applying either $\langle Constr \rangle$ to Q_1 (here Q_1 is considered as belonging to $\mathcal{L}^2$) or $\langle \forall C \rangle$ to Q_3. Actually each node in $\mathcal{L}^2$ can be reached starting from either another node in $\mathcal{L}^2$ or a node in $\mathcal{L}^1$. This can be exploited to speed up the search at levels of finer granularity as shown later.

4.1 Reaching Ideality

Descriptive data mining problems are characterized by hypothesis spaces with *dense solutions*. Ideal refinement operators are usually suggested to search spaces of this kind [2]. They satisfy the following properties.

Definition 8. *Let ρ a downward refinement operator for a quasi-ordered set $(\mathcal{L}, \succeq)$. Denoted with ρ^* the* transitive closure *of ρ:*

- *ρ is* locally finite *iff $\forall P \in \mathcal{L} : \rho(P)$ is finite and computable;*
- *ρ is* proper *iff $\forall P \in \mathcal{L} \forall Q \in \rho(P)$: $Q \not\sim P$;*
- *ρ is* complete *iff $\forall P, Q \in \mathcal{L}$ if $P \succ Q$ then $\exists Q' \in \rho^*(P)$: $Q' \sim Q$.*

Unfortunately, for clausal languages ordered by θ-subsumption or stronger orders such as generalized subsumption, ideal operators have been proven not to exist [20]. They can be approximated by dropping the requirement of properness or by bounding the language. We choose to follow the latter option. It guarantees that, if $(\mathcal{L}, \succeq)$ is a quasi-ordered set, $\mathcal{L}$ is finite and $\succeq$ is decidable, then there always exists an ideal refinement operator for $(\mathcal{L}, \succeq)$. In our case, since $\succeq_{\mathcal{B}}$ is a decidable quasi-order, we only need to bound $\mathcal{L}$ in a suitable manner.

The expressive power of $\mathcal{AL}$-log requires several bounds to be imposed on $\mathcal{L}$ in order to guarantee its finiteness. First, it is necessary to set a maximum level $maxG$ of description granularity, so that the problem of finiteness of $\mathcal{L}$ can be boiled down to the finiteness of each $\mathcal{L}^l$, $1 \leq l \leq maxG$. Second, it is necessary to introduce a complexity measure for $\mathcal{O}$-queries, as a pair of two different coordinates. Note that the complexity of an $\mathcal{O}$-query resides in its body. Therefore, given an $\mathcal{O}$-query P, the former coordinate is the size of the biggest literal in $body(P)$, while the latter is the number of literals in $body(P)$. Constraints do count as literals.

Definition 9. *Let L be either a* DATALOG *atom or an $\mathcal{ALC}$ constraint. Then $size(L)$ is equal to the difference between the number of symbol occurrences in L and the number of distinct variables in L.*

Example 7. Let L_1, L_2 be the literals `item(X,Y)` and `X:DairyProduct`. Then $size(L_1) = 3 - 2 = 1$ and $size(L_2) = 2 - 1 = 1$.

Definition 10. *Let P an $\mathcal{O}$-query, $maxsize(P)$ the maximum of $\{size(L) | L \in body(P)\}$, and $|P|$ the number of literals in P. We call $\mathcal{O} - size(P)$ the pair $(maxsize(body(P)), |P|)$.*

Definition 11. *Let P an $\mathcal{O}$-query, and $maxS$,$maxD$ be natural numbers. We say that P is* bounded *by $(maxS, maxD)$ if $maxsize(body(P)) \leq maxS$ and $|body(P)| \leq maxD$.*

Example 8. The language $\mathcal{L} = \mathcal{L}^1 \cup \mathcal{L}^2$ partially reported in Example 3 and illustrated in Example 6 contains $(1, 5)$-bounded $\mathcal{O}$-queries. Note that the bound $maxS = 1$ derives from the fact that $\mathcal{A}$ contains only binary predicates and Γ contains only assertions of primitive concepts.

Proposition 1. *Given a language $\mathcal{L}$ of $\mathcal{O}$-queries and two integers $maxS$, $maxD > 0$, the set $\{P \in \mathcal{L} | P$ is bounded by $(maxS, maxD)\}$ is finite up to variants.*

Proof. We will only sketch the idea behind this. The language $\mathcal{L}$ has finitely many constants and predicate symbols because it is built starting from the alphabets $\mathcal{A}$ and Γ. Furthermore it has no functors because it is a fragment of $\mathcal{AL}$-log. Suppose we are given $(maxS, maxD)$. It is not difficult to see that the set of literals (either DATALOG *atoms or constraints) with $size \leq maxS$ is finite up to variants. Let v be the maximum of the set $\{n|$ there is a literal $L \in P$ with $size \leq maxS$ that contains n distinct variables$\}$. Because a $(maxS, maxD)$-bounded $\mathcal{O}$-query can contain at most $maxD$ literals in the body, each of which can contain at most v distinct variables, a $(maxS, maxD)$-bounded $\mathcal{O}$-query can contain at most $maxDv$ distinct variables. Let us fix distinct variables $X_1, \ldots, X_{maxDv}$. Now let $\mathcal{K}$ be the finite set of all literals of $size \leq maxS$ that can be constructed from the predicate symbols and constants in $\mathcal{A}$, the concept symbols in Γ and variables $X_1, \ldots, X_{maxDv}$. Since each $\mathcal{O}$-query that is bounded by $(maxS, maxD)$ must be (a variant of) a subset of $\mathcal{K}$, there are only finitely many such $\mathcal{O}$-queries, up to variants.*

We can prove that $\rho_{\mathcal{O}}$ is an ideal refinement operator for $(\mathcal{L}, \succeq_{\mathcal{B}})$ that maps reduced $(maxS, maxD)$-bounded $\mathcal{O}$-queries into reduced $(maxS, maxD)$-bounded $\mathcal{O}$-queries under the OI bias.

Theorem 3. *Let $(maxS, maxD)$ be a pair of natural numbers, and $\mathcal{L}$ be the language $\{\mathcal{L}^l\}_{1 \leq l \leq maxG}$ such that each $\mathcal{L}^l$ contains reduced $(maxS, maxD)$-bounded $\mathcal{O}$-queries. The downward refinement operator $\rho_{\mathcal{O}}$ is ideal for $(\mathcal{L}, \succeq_{\mathcal{B}})$.*

Proof. Let $P, Q \in \mathcal{L}$.

Local Finiteness. *Suppose $P \in \mathcal{L}^l$. The alphabets $\mathcal{A}$ and Γ underlying $\mathcal{L}$ are finite sets. Furthermore, each of the three refinement rules consists of instructions that can be completed in a finite number of steps. Therefore $\rho_{\mathcal{O}}(P)$ is finite and computable.*

Properness. *Suppose $P \in \mathcal{L}^l$. In the case of either $\langle Atom \rangle$ or $\langle Constr \rangle$, Q is strictly longer than P because the OI bias avoids the identification of literals. Therefore $P \succ Q$. In the case of $\langle \forall C \rangle$, the occurrence of a strictly stronger constraint in Q assures that $P \succ Q$.*

Completeness. *Suppose $P \succ_{\mathcal{B}} Q$. Then either Q is a downward cover of P, in which case there is an $R \in \rho_{\mathcal{O}}(P)$ such that $Q \sim_{\mathcal{B}} R$, or there is an $R \in \rho_{\mathcal{O}}(P)$ such that $P \succ_{\mathcal{B}} R \succ_{\mathcal{B}} Q$. In the latter case, we can find an $S \in \rho_{\mathcal{O}}(R)$ such that $P \succ_{\mathcal{B}} R \succ_{\mathcal{B}} S \succeq_{\mathcal{B}} Q$, etc. Since $\mathcal{L}$ is finite and $\rho_{\mathcal{O}}$ is proper, we must eventually find a ρ-chain from P to a member of the equivalence class of Q, so $\rho_{\mathcal{O}}$ is complete.*

Ideal refinement operators are mainly of theoretical interest, because in practice it is often very inefficient to find downward (resp. upward) covers for every $P \in \mathcal{L}$. Thus more constructive - though possibly improper - refinement operators are usually to be preferred over ideal ones. In the following we show that efficient algorithms can be designed to implement $\rho_{\mathcal{O}}$ in the context of interest.

4.2 Making Ideality Something Real

The operator $\rho_{\mathcal{O}}$ has been implemented in $\mathcal{AL}$-QuIn ($\mathcal{AL}$-log QUery INduction) [15], an ILP system that - according to Mannila's levelwise method [19] for frequent pattern discovery - searches the space $(\mathcal{L}, \succeq_{\mathcal{B}})$ breadth-first by alternating candidate generation and candidate evaluation phases. In particular, *candidate generation* consists of a refinement step followed by a pruning step. The former applies one of the three rules of $\rho_{\mathcal{O}}$ to patterns previously found frequent by preserving the properties of linkedness and safety. The pruning step allows some infrequent patterns to be detected and discarded prior to evaluation thanks to the following property [15]: Under the assumption that $minsup^l \leq minsup^{l-1}$, $1 < l < maxG$, a k-pattern Q in $\mathcal{L}^l$ is infrequent if it is $\mathcal{B}$-subsumed w.r.t. an $\mathcal{AL}$-log knowledge base $\mathcal{B}$ by either (i) an infrequent $(k-1)$-pattern in $\mathcal{L}^l$ or (ii) an infrequent k-pattern in $\mathcal{L}^{l-1}$. Note that (i-ii) require a high number of subsumption checks to be performed. This makes candidate generation computationally expensive. Appropriate complementary data structures can help mitigating the computational effort. Our implementation of $\rho_{\mathcal{O}}$ uses a graph of backward pointers to be updated while searching in order to keep track of both intra-space and inter-space search stages. Figure 2 gives an example of such graph for the portion of space reported in Figure 1. Here nodes, dotted edges and dashed edges represent patterns, intra-space parenthood and inter-space parenthood, respectively. We shall illustrate the benefits of using this data structure by going into details of the procedure generateCandidates() reported in Figure 3.

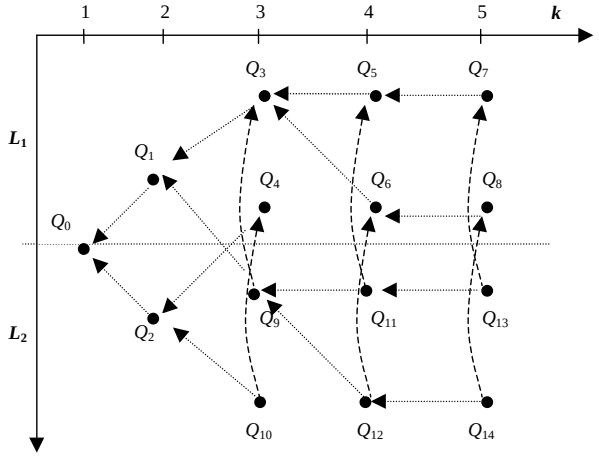

Fig. 2. Graph of backward pointers.

For a given level l of description granularity, $1 \leq l \leq maxG$, procedure generateCandidates() builds the set $\mathcal{C}_k^l$ of candidate k-patterns starting from the set $\mathcal{F}_{k-1}^l$ of frequent $(k-1)$-patterns and the language $\mathcal{L}^l$ by taking the set $\mathcal{I}^l$ of infrequent patterns into account. It consists of two computation branches. The

```
Procedure generateCandidates(F^l_{k-1}, L^l, var I^l)
1. C^l_k ← ∅;
2. if l = 1 then /* search in L^1 */
3.   foreach pattern P in F^l_{k-1} do
4.       Q ← intraRefine(P, L^l);
5.       Q ← prune(Q, I^l);
6.       foreach pattern Q in Q do
             /* set the intra-space edge from Q to P */
7.           setIntraSpaceEdge(Q, P)
8.       endforeach
9.       C^l_k ← C^l_k ∪ Q
10.  endforeach
11. else /* search in L^l, l > 1*/
12.  foreach pattern P in F^l_{k-1} do
13.      P' ← getInterSpaceParent(P);
14.      Q' ← getIntraSpaceChildren(P');
15.      foreach pattern Q' in Q' do
16.          Q ← interRefine(Q');
17.          Q ← prune(Q, I^l);
18.          foreach pattern Q in Q do
                 /* set the inter-space edge from Q to Q' */
19.              setInterSpaceEdge(Q, Q');
                 /* set the intra-space edge from Q to P
20.              setIntraSpaceEdge(Q, P);
21.          endforeach
22.          C^l_k ← C^l_k ∪ Q
23.      endforeach
24.  endforeach
return C^l_k
```

Fig. 3. Implementation of $\rho_{\mathcal{O}}$ in the candidate generation phase of $\mathcal{AL}$-QuIn

former concerns the case of search in $\mathcal{L}^1$. It applies either $\langle Atom \rangle$ or $\langle Constr \rangle$ (procedure intraRefine()), performs the pruning step (procedure prune()) and inserts an intra-space backward pointer for each retained candidate (procedure setIntraSpaceEdge()).

Example 9. As an illustrative example of candidate generation at the first level of description granularity we report the computation of $\mathcal{C}_2^1$ with reference to Example 6. The set $\mathcal{F}_1^1$ contains the trivial query Q_0 only. The procedure call intraRefine(Q_0, $\mathcal{L}^1$) returns the set of possible refinements of Q_0 according to directives of $\mathcal{L}^1$, namely the queries Q_1 and Q_2 obtained by adding the atoms `item(X,Y)` and `purchaser(X,Y)` to the body of Q_0 respectively. In both refinements it is not necessary to insert inequality atoms. Furthermore the procedure call prune($\mathcal{Q}$, $\mathcal{I}^1$) does affect neither $\mathcal{Q}$ nor $\mathcal{I}^1$. Before returning control to the main procedure of $\mathcal{AL}$-QuIn, generateCandidates($\mathcal{F}_1^1$, $\mathcal{L}^1$, $\mathcal{I}^1$) updates the graph of backward pointers by inserting intra-space edges for Q_1 and Q_2 as shown in

Figure 2. Assuming that $\mathcal{F}_2^1$ contains Q_1, we carry on this example by focusing on the generation of candidate patterns starting from Q_1. Possible refinements according to directives of $\mathcal{L}^1$ are the queries Q'_1, Q'_2, Q'_3 and Q'_4 reported in Example 6. Since Q'_1 and Q'_2 do not fulfill the requirement on the maximum number of unconstrained variables, they are generated then pruned. So Q'_3, namely Q_3, and Q'_4 are the only to survive the pruning step. Since Q'_4 will not pass the candidate evaluation phase, it does not appear as child of Q_1 in Figure 2.

The other branch of generateCandidates() concerns the case of search at deeper levels of description granularity. One can expect that it is enough to simply replace the procedure intraRefine() with a procedure interRefine() which implements the refinement rule $\langle \forall C \rangle$. But things are more complicated. Let us suppose that the current space to be searched is $\mathcal{L}^l$, $l > 1$. On one side, searching $\mathcal{L}^l$ with only $\langle Atom \rangle$ or $\langle Constr \rangle$ implies to restart from scratch. Rather we would like to capitalize on the computational effort made when searching $\mathcal{L}^{l-1}$ and minimize the number of inter-space subsumption checks. On the other side, searching $\mathcal{L}^l$ indirectly, i.e. by applying $\langle \forall C \rangle$ to $\mathcal{O}$-queries found frequent in $\mathcal{L}^{l-1}$, implies the loss of the useful information that could be collected if $\mathcal{L}^l$ was searched directly. E.g., when generating $\mathcal{C}_k^l$, $l, k > 1$, it happens that intraRefine($\mathcal{F}_{k-1}^l$)$\subseteq$ interRefine($\mathcal{F}_k^{l-1}$). This means that a blind application of $\langle \forall C \rangle$ causes an increment of intra-space subsumption checks. It is necessary to find a compromise between these two apparently irreconcilable solutions. Our solution requires that $\mathcal{C}_k^l$, $l, k > 1$, is computed taking both $\mathcal{F}_{k-1}^l$ and $\mathcal{F}_k^{l-1}$ into account. In particular, the expansion of a node P in $\mathcal{F}_{k-1}^l$ is done as follows:

- retrieve the inter-space parent node P' of P by following the inter-space backward pointer (step (13));
- retrieve the set $\mathcal{Q}' \subseteq \mathcal{F}_k^{l-1}$ of intra-space children nodes of P' by navigating intra-space backward pointers in reverse sense (step (14));
- generate the set $\mathcal{Q}$ of $\mathcal{O}$-queries obtained by applying $\langle \forall C \rangle$ to each Q' in $\mathcal{Q}'$ (step (16))

This not only avoids a blind application of the refinement rule $\langle \forall C \rangle$ but also lightens the computational load during the pruning step.

Example 10. Again with reference to Example 6 the expansion of Q_1 in the case of Q_1 considered as belonging to $\mathcal{L}^2$ is done indirectly by accessing search stages of success in $\mathcal{L}^1$ departing from Q_1. In Example 9 we have seen that Q_3 is the only child of Q_1. Possible refinements of Q_3 by means of $\langle \forall C \rangle$ are the queries Q''_3 up to Q''_{10} listed in Example 6. Note that these are equivalent to the queries Q'_5 up to Q'_{12} in $\rho_{\mathcal{O}}(Q_1)$. Furthermore the queries Q'_{13} up to Q'_{15}, though possible refinements of Q_1, are not generated because their ancestor Q'_4 turned out to be infrequent. This avoids the generation of certainly infrequent patterns in $\mathcal{L}^2$.

5 Conclusions

Hybrid languages supply expressive *and* deductive power which have no counterpart in Horn clausal logic. This makes them appealing for challenging applications such as descriptive data mining in application domains that require a

uniform treatment of both relational and structural features of data. Also this poses several research issues to the ILP community, e.g. the definition of more sophisticated refinement operators than the ones typically used in ILP. The main contribution of this paper is the definition of an ideal (downward) refinement operator $\rho_{\mathcal{O}}$ to search spaces of descriptions at multiple granularity levels in the $\mathcal{AL}$-log framework. In particular, ideality has been approximated by bounding the language and assuming the OI bias. Though this result is mainly of theoretical interest, it is worthy having it because searching spaces with dense solutions is peculiar to descriptive data mining tasks. With respect to the context of interest we have provided an efficient implementation of $\rho_{\mathcal{O}}$. Note that the resulting algorithm is not an attempt at deriving an optimal operator from $\rho_{\mathcal{O}}$ but aims at lightening the computational load of inter-space subsumption checks.

We would like to emphasize that the choice of an application context and the investigation of ILP issues within the chosen context make a substantial difference between our work and related work on learning in hybrid languages. Indeed our broader goal is the definition of an ILP setting for mining object-relational databases [15]. We claim that $\mathcal{AL}$-log supports a simple yet significant object-relational data model. This provides - among the other things - an application-driven motivation for assuming the OI bias and concentrating on concept hierarchies in our $\mathcal{AL}$-log framework. Differences between $\mathcal{AL}$-QuIn and the ILP system Warmr [7] for mining "pure" relational patterns are discussed in [18]. We intend to carry on the work presented in this paper by following this approach of reconciling theory and practice. In particular, besides the already investigated application to spatial data mining [15], the Semantic Web seems to be a promising source of new interesting tasks for $\mathcal{AL}$-QuIn.

Acknowledgments

We would like to thank the anonymous reviewers for their useful comments and Nicola Fanizzi for the fruitful discussions.

References

1. L. Badea and S.-W. Nienhuys-Cheng. A refinement operator for description logics. In J. Cussens and A. Frisch, editors, *Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 40–59. Springer-Verlag, 2000.
2. L. Badea and M. Stanciu. Refinement operators can be (weakly) perfect. In S. Džeroski and P. Flach, editors, *Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 21–32. Springer, 1999.
3. A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.
4. W. Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.
5. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer, 1990.
6. W. Cohen and H. Hirsh. Learning the CLASSIC description logic: Thoretical and experimental results. In *Proc. of the 4th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 121–133. Morgan Kaufmann, 1994.

7. L. Dehaspe and H. Toivonen. Discovery of frequent DATALOG patterns. *Data Mining and Knowledge Discovery*, 3:7–36, 1999.
8. F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. $\mathcal{AL}$-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.
9. A. Frisch. The substitutional framework for sorted deduction: Fundamental results on hybrid reasoning. *Artificial Intelligence*, 49:161–198, 1991.
10. A. Frisch. Sorted downward refinement: Building background knowledge into a refinement operator for inductive logic programming. In S. Džeroski and P. Flach, editors, *Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 104–115. Springer, 1999.
11. A. Frisch and A. Cohn. Thoughts and afterthoughts on the 1988 workshop on principles of hybrid reasoning. *AI Magazine*, 11(5):84–87, 1991.
12. J.-U. Kietz. Learnability of description logic programs. In S. Matwin and C. Sammut, editors, *Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 117–132. Springer, 2003.
13. J.-U. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(1):193–217, 1994.
14. A. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165–209, 1998.
15. F.A. Lisi. *An ILP Setting for Object-Relational Data Mining*. Ph.D. Thesis, Department of Computer Science, University of Bari, Italy, 2002.
16. F.A. Lisi, S. Ferilli, and N. Fanizzi. Object Identity as Search Bias for Pattern Spaces. In F. van Harmelen, editor, *ECAI 2002. Proceedings of the 15th European Conference on Artificial Intelligence*, pages 375–379, Amsterdam, 2002. IOS Press.
17. F.A. Lisi and D. Malerba. Bridging the Gap between Horn Clausal Logic and Description Logics in Inductive Learning. In A. Cappelli and F. Turini, editors, *AI*IA 2003: Advances in Artificial Intelligence*, volume ? of *Lecture Notes in Artificial Intelligence*, Springer, 2003. to appear.
18. F.A. Lisi and D. Malerba. Towards Object-Relational Data Mining. In S. Flesca, S. Greco, D. Saccà, and E. Zumpano, editors, *Proc. of the 11th Italian Symposium on Advanced Database Systems*, pages 269–280. Rubbettino Editore, Italy, 2003.
19. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
20. S. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence*. Springer, 1997.
21. R. Reiter. Equality and domain closure in first order databases. *Journal of ACM*, 27:235–249, 1980.
22. C. Rouveirol and V. Ventos. Towards Learning in CARIN-$\mathcal{ALN}$. In J. Cussens and A. Frisch, editors, *Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 191–208. Springer, 2000.
23. M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
24. G. Semeraro, F. Esposito, D. Malerba, N. Fanizzi, and S. Ferilli. A logic framework for the incremental inductive synthesis of Datalog theories. In N. Fuchs, editor, *Proc. of 7th Int. Workshop on Logic Program Synthesis and Transformation*, volume 1463 of *Lecture Notes in Computer Science*, pages 300–321. Springer, 1998.