On rewritings of queries in disjunction-conjunction branching shape of arbitrary depth

Sebastien Bonduelle

Introduction

Existential rules have become a major formalism to represent ontologies in Knowledge Representation (KR). Though they form a subset of First-Order Logics, they originated from two fields of KR: conceptuals graphs [1], and more precisely the formalisation work initiated by Chein, Mugnier and Book, and independently from relational databases, as extension of the Datalog language [2].

In this formalism, the fondamental problem is to check whether or not a query is entailed by a knowledge base , *i.e* a database and an ontology. The database can be seen as a set of ground atoms (as done in this document) encoding existential knowledge, and the ontology is a set of existential rules encoding universal knowledge. As an example of such a rule (in its logical form), let us consider:

$$\forall X \, \forall T((student(X), learns(X,T), topic(T)) \rightarrow \exists Y(has-teacher(X,Y), teacher(Y), knows(Y,T)))$$

asserting that every student learning a topic has a teacher knowing that topic. The existential variable Y introduces a new entity that is not present in the database. This is the main difference between existential rules (also called Datalog+) and Datalog: this possibility, called value invention in KR, is considered very important when writing ontologies.

Finally, the query is usually a conjunctive query (in this document, we will restrict our study to boolean queries, that only answer true or false). This EXISTENTIAL RULES DEDUCTION problem is undecidable (it is indeed a computation model), and a lot of efforts have been brought to exhibit multiple decidable (and sometimes tractable) fragments of existential rules, without sacrificing the expressivity required to write ontologies [3].

Two main subsets of decidable fragments are those for which the forward chaining algorithm halts, and those for which the backward chaining algorithm halts.

The forward chaining algorithm is the simpler to understand: we consider in a first step uniquely the database and the ontology, and use the ontology to saturate the database, adding to it every atom that can be deduced by using rules. As a very basic example, if the database contains $Man(socrates)$ and the ontology $\forall X \, (Man(X) \rightarrow Mortal(X))$, then we add the deduced atom $Mortal(socrates)$ to the database. When this algorithm halts, we obtain a universal model, and the resulting database can be queried in a second step independently of the ontology, for instance with the query $Mortal(socrates)$. In database, this algorithm is known as the chase.

We can obtain the same results by using the backward chaining algorithm. In that case, we consider in the first step uniquely the ontology and the query. The query is rewritten according to the ontology, generating a union of conjunctive queries (UCQ). With our previous and simple example, the rewritten query would be $Man(socrates) \vee Mortal(socrates)$. Then we have only in a second step to evaluate the rewritten query against the database. The usual rewriting mechanism studied in logic programming is made more complex by the presence of existential variables in the conclusion of rules, and we have to rely on piece unifiers [4]. When this algorithm halts, it has several benefits: we do not have to saturate the database, saving a lot of disk space, and if the database changes, we do not have to compute its saturation again. Theoretically, as soon as we are in a decidable fragment of the language, we are also in AC0 in data complexity (a complexity measure often used in databases, where the size of the query and of the ontology are considered as constants).

This miraculous data complexity is, in practical terms, a mirage. Consider the very simple query $A_1(X), \cdots, A_p(X)$, and suppose there is k different ways to prove each A_i (for instance, for $1 \leq i \leq p$, for $1 \leq j \leq k$ there is a rule of form $\forall X \, (A_i^j(X) \rightarrow A_i(X))$). In that case, the rewriting of the query would consist in the union of an exponential number $k0$ of conjunctive queries. It is often the case that an obtained UCQ is too big to be evaluated against a

database. To tackle this problem, [5] suggested to compile some rules. Those compilable rules are not used in the rewriting mechanism, leading to smaller UCQs, and the compilation is used at the evaluation step.

The problem is that compilable rules form only a strict subset of linear rules (with body and head of size 1). Though a great part of ontologies consist in type hierarchies, that are translated to compilable rules, we felt it was possible to factorize the UCQ rewriting.

1 Starting definitions and theorems

1.1 Facts and Queries

Usually the equality symbol is not used, however in this paper we will use it. It is however a separate predicate, that has a fixed interpretation.

1.1.1 Syntax

We begin by defining the vocabulary on which facts, rules and queries will be defined.

Definition 1 (Vocabulary)**.** *A* vocabulary *is a triple $\mathcal{V} = (\mathcal{P}, \mathcal{C})$ where $\mathcal{P}$ is a set of predicate symbols and $\mathcal{C}$ is a set of constant symbols. To each predicate is associated an integer called its* arity*.*

Though it is not part of the vocabulary, we also consider a set $\mathcal{X}$ of *variables*. In this paper, predicate and constants symbols will be in lowerscript letters, while variable names will begin with an upperscript letter. Constants as well as variables are denoted as *terms*.

In what folllows, *atoms* on a vocabulary will be either

- *standard atoms* of the form $p(t_1, \ldots, t_k)$ where p is a predicate of arity k and the t_i are terms
- *equality atoms* of the form $t = t'$ where both t and t' are terms.

The basic building block for facts, queries and rules is called a conjunct. It is essentially a conjunction of atoms, but is traditionally represented as a set of atoms. Note that even though the set representation does not allow for multiple occurences of the same atom in a conjunction, it is without loss of generality.

Definition 2 (Conjunct)**.** *A* conjunct *on a vocabulary $\mathcal{V}$ is a set of atoms on $\mathcal{V}$. The FOL formula $\phi(C)$ associated with a conjunct C is the conjunction of its atoms. We denote by $\texttt{var}(C)$ the set of variables of a conjunct C.*

Remark. *Note that the formula $\phi(C)$ is not necessarily closed.*

In knowledge representation, knowledge is usually split into factual and ontological knowledge. Here, factual knowledge will be encoded by a conjunct that is sometimes called facts base, but that we will simply call a *fact*. Databases require a fact to be grounded (*i.e.,* without variables), but we do not enforce that restriction here (we will see anyway that applying a rule to a grounded fact creates a fact that is not necessarily grounded fact). Moreover, since adding equality to facts would add neither expressivity to our formalism nor effficiency to our algorithms, we can simplify our definitions by forbidding equality atoms to appear in such facts.

Definition 3 (Fact)**.** *A* fact *on a vocabulary $\mathcal{V}$ is a set of standard atoms. We denote by $\mathcal{F}$ the set of facts. The FOL formula $\Phi(F)$ associated with a fact F is the existential closure of the conjunction of its atoms.*

Definition 4. *A* conjunctive query with equality *(or ECQ) is a conjunct (thus possibly with equality atoms). A* conjunctive query *(or CQ) is an ECQ that has no equality atom. To each ECQ Q we associate a FOL formula $\Phi(Q)$ which is the existential closure of the conjunction $\phi(Q)$ of its atoms.*

Note that CQs and Facts have the same syntax and logical form. ECQs only add equality atoms (whose interest will become clearer later in this paper). Note also that since ECQs are closed formulae, they should really be called *boolean* conjunctive queries (we are not interested in the answers to the query, but in the existence of an answer). We are only interested in boolean queries in this paper.

Example 1. *$\{p(X, a), q(Y), p(b, Y), X = Y\}$ and $\{p(X, a), q(Y), p(b, Y)\}$ are two conjuncts on the vocabulary $(\{p(2), q(1)\}, \{a(0), b(0)\}, \{P(2)\})$. The first, containing an equality atom, can only be considered as an ECQ, while the second, without any equality atom, could be considered as a Fact or an ECQ (which would also be a CQ). Their associated FOL formulae are respectively $\exists X\, \exists Y\, p(X, a) \wedge q(Y) \wedge p(b, Y) \wedge X = Y$ and $\exists X\, \exists Y\, p(X, a) \wedge q(Y) \wedge p(b, Y)$.*

1.1.2 Semantic

We begin with the classic notions of interpretations in first order logics, on which is based entailment (or semantic consequence). We will the be able to characterize entailment of a query by a fact using an operation called homomorphism.

Definition 5 (Interpretation)**.** *An* interpretation *of a vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C}, \mathcal{ER})$ is a pair $I = (\Delta, .^I)$ where Δ is a non empty set called the* interpretation domain *and $.^I$ is an* interpretation function *such that:*

1. $\forall c \in \mathcal{C}, c^I \in \Delta$;
2. $\forall p \in \mathcal{P}, p^I \subseteq \Delta^k$ *where k is the arity of p.*

Definition 6 (Value of a term)**.** *Let $I = (\Delta, .^I)$ be an interpretation of a vocabulary $(\mathcal{P}, \mathcal{C})$, and λ an affectation of the variables into Δ. We define the* value *of a term t in I for λ (noted $[\![t]\!]_{I,\lambda}$) by t^I if $t \in \mathcal{C}$ or $\lambda(t)$ if t is a variable.*

Definition 7 (Truth of a formula)**.** *Let $I = (\Delta, .^I)$ be an interpretation of a vocabulary $(\mathcal{P}, \mathcal{C})$, and λ an affectation of the variables into Δ. We define the* truth *of a formula φ in I for λ (noted $I, \lambda \models \varphi$) by induction:*

- $I, \lambda \models p(t_1, \ldots, t_n) \iff ([\![t_1]\!]_{I,\lambda}, \ldots, [\![t_n]\!]_{I,\lambda}) \in p^I$ *if p is a predicate;*
- $I, \lambda \models t_1 = t_2 \iff [\![t_1]\!]_{I,\lambda} = [\![t_2]\!]_{I,\lambda}$;
- $I, \lambda \models \neg\varphi \iff I, \lambda \not\models \varphi$;
- $I, \lambda \models \varphi \wedge \psi \iff I, \lambda \models \varphi$ *and* $I, \lambda \models \psi$;
- $I, \lambda \models \varphi \vee \psi \iff I, \lambda \models \varphi$ *or* $I, \lambda \models \psi$;
- $I, \lambda \models \varphi \to \psi \iff I, \lambda \models \neg\varphi \vee \psi$;
- $I, \lambda \models \forall X \varphi \iff$ *for all* $d \in \Delta, I, \lambda[X \to d] \models \varphi$;
- $I, \lambda \models \exists X \varphi \iff$ *there exists* $d \in \Delta$ *such that* $I, \lambda[X \to d] \models \varphi$.

The previous definition can be simplified for Facts and ECQs.

Proposition 1 (Truth of an ECQ)**.** *The formula $\Phi(Q)$ associated to an ECQ Q is true in an interpretation I iff there exists an affectation λ of the variables of Q to the interpretation domain such that:*

1. *for each standard atom a in Q, $I, \lambda \models a$;*
2. *for each equality atom $t = t'$ in Q, $\lambda(t) = \lambda(t')$.*

Since Facts and CQs have the same syntax and associated formulas as ECQs without equality atoms, their truth value is also characterized by the previous proposition (though the second criteria obviously does not have to be verified).

When restricting ourselves to ECQs, SATISFIABILITY and VALIDITY are not interesting problems: an ECQ is always satisfiable (unless the unique name assumption is made and an equality between two distinct constants can be inferred), and the only valid ECQs contain only equality atoms of form $t = t$. In what follows, we will be interested in the following DEDUCTION PROBLEM: given a fact F and an ECQ Q, does F entails Q ?

1.1.3 Homomorphisms

Entailment It is well known that the existence of an homomorphism from a CQ Q to a fact F characterizes entailment of Q by F. In what follows, we extend that notion to E-homomorphisms (for equality preserving homomorphisms) from an ECQ Q to a fact F.

Definition 8 (E-Homomorphism)**.** *An* equality-preserving homomorphism *(for short* E-homomorphism*) from an ECQ Q to a fact F is a substitution π from the variables of Q ($\mathtt{var}(Q)$) to the terms of F ($\mathtt{term}(F)$) such that:*

1. *for each standard atom $a \in Q$, $\pi(a) \in F$;*

2. *for each equality atom $t = t' \in Q$, $\pi(t) = \pi(t')$.*

When there is such an E-homomorphism, we say that Q E-maps to F.

It is immediate to check that when Q is a CQ, the previous definition becomes "a substitution π such that $\pi(Q) \subseteq F$", and an E-homomorphism from a CQ is thus a homomorphism (and we can say that Q maps to F). The homomorphism theorem then states that, when Q is a CQ, $F \models Q$ iff there is a homomorphism (*i.e.*, an E-homomorphism) from Q to F. It remains now to prove that E-homomorphisms characterize ECQ entailement.

Example 2. *Let $F = \{p(X, a), q(Y), p(b, Y)\}$, $Q = \{p(U, V), q(W)\}$ and $Q' = Q \cup \{V = W\}$ be three conjuncts. Then $\pi = \{(U, X), (V, a), (W, Y)\}$ is an E-homomorphism (and an homomorphism) from Q to F, but it is not an homomorphism nor an E-homomorphism from Q' to F. On the other hand, $\pi' = \{(U, b), (V, Y), (W, Y)\}$ is both an homomorphism from Q to F and an E-homomorphism from Q' to F.*

To prove that E-homomorphisms effectively characterize ECQ entailment, we introduce an operation (which will also be used later in this paper) called *equalization*. Basically, the equalization of an ECQ is a CQ having the same semantics.

Definition 9 (Equalizable ECQs)**.** *An ECQ is* equalizable *when no equivalence class of terms contains two distinct constants.*

Example 3. *The ECQ $\{p(X, Y, Z, T, U), X = a, X = Y, Y = b, Z = T, T = U\}$ contains two equivalence classes of terms, namely $\{X, Y, a, b\}$ and $\{Z, T, U\}$. Since the first equivalence class contains the two constants a and b, the ECQ is non-equalizable.*

Proposition 2. *Let F be a fact and Q be a non-equalizable ECQ. Then $F \not\models Q$*

Proof. See that the isomorphic model of F assigns a distinct element of the domain to each constant. It follows that it is not a model of Q. □

Thus non-equalizable ECQs do not need to be transformed into CQs.

Definition 10 (Equalization)**.** *Let $Q = Q_s \cup Q_e$ be an equalizable ECQ, where Q_s contains only standard atoms and Q_e only equality atoms. An* equalization substitution *of Q is a substitution σ that associates to each variable X of Q_e a term representing its equivalence class $[X]$.*

1. *if $[X]$ contains a (necessarily unique) constant a, then $\sigma(X) = a$;*

2. *otherwise, for each $Y \in [X]$, $\sigma(Y) = Z$, where Z is a variable representing the equivalence class $[X]$: Z is either a variable of $[X]$ or a variable that is not in Q.*

If σ is an equalization substitution of Q, then $\sigma(Q_s)$ is an equalization CQ *of Q.*

Note that an equalization CQ is indeed a CQ. In what follows, and when there is no ambiguity, we will call *equalization* either the substitution or the CQ it can produce. Moreover, we arbitrarily decide to chose as a representant of $[X]$ (unless it contains a constant) the smallest variable ranked by alphabetical order in $[X]$. We can thus obtain *the* equalization of Q.

Example 4. *Let $Q = \{p(U, V), q(W), p(W, T), U = T, V = T\}$ be an ECQ. The only equivalence class induced by its equality atoms is $\{U, V, T\}$, and Q is thus equalizable. Possible equalization substitutions of Q associate the same variable to the variables U, V and T. Though we could choose either one of those variables (or even another variable X, but not W which is in Q), we choose T for* the *equalization substitution of Q.* The *equalization CQ of Q is $\{p(T, T), q(W), p(W, T)\}$.*

Proposition 3. *Let Q be an equalizable ECQ and Q' be an equalization of Q. Then $Q \equiv Q'$.*

Proof. It follows from proposition 1. □

The following proposition prepares for the characterization of ECQ entailment.

Proposition 4. *Let Q be an equalizable ECQ and F be a fact. Let Q' be an equalization of Q. Then there is an E-homomorphism from Q to F iff there is an homomorphism from Q' to F.*

These two propositions allow us to extend the homomorphism theorem to one that links ECQ entailment and E-homomorphisms:

Theorem 1 (E-Homomorphism)**.** *Let F be a fact and Q be an equalizable ECQ. Then there is an E-homomorphism from Q to F if and only if $F \models Q$.*

Note that, as for CQ-ENTAILMENT, ECQ-ENTAILMENT is an NP-complete problem. Indeed, an E-homomorphism is a polynomial certificate, and CQ-ENTAILMENT is a particular case of ECQ-ENTAILMENT. Polynomial subclasses of the CQ-ENTAILMENT problem involve mainly bounding the number of variables in the query or its treewidth. Those polynomial subclasses naturally adapt to ECQ-ENTAILMENT by using equalization.

1.2 $\forall\exists$-rules

1.2.1 Syntax and semantic

We present here the syntax and semantics of rules. There was no need to introduce equality neither in the hypothesis nor the conclusion of rules. Adding equality in the hypothesis would add no expressivity (thanks to equalization), and, contrary to queries, no optimization issue was at stake. Adding equality in the conclusion (as done in EGDs – Equality Generating Dependencies) leads to more complex properties of derivations, and more complex rewritings. It is not considered in this paper.

Definition 11 ($\forall\exists$-rule)**.** *A* $\forall\exists$-rule *(later simply called* rule*) is a pair of conjuncts without any equality atom $R = (H, C)$ (commonly noted $H \to C$). H is called the* hypothesis *(or* body*), noted $\texttt{hyp}(R)$ and C is called the* conclusion *(or* head*), noted $\texttt{conc}(R)$. The* frontier *of R is $\texttt{fr}(R) = \texttt{var}(H) \cap \texttt{var}(C)$. We denote by $\forall\exists$-$\mathcal{R}$ the set of all $\forall\exists$-rules.*

The FOL formula $\Phi(R)$ associated with a rule $R = (H, C)$ is $\forall\bar{X}(\phi(H) \to \exists\bar{Y}\phi(C))$, where $\bar{X}$ contains all variables appearing in H and $\bar{Y}$ those appearing only in C.

As for facts and queries, the general definition of the truth value of a formula in an interpretation can be simplified in the case of rules:

Proposition 5 (Truth of a rule)**.** *The formula $\Phi(R)$ associated to a rule $R = (H, C)$ is true in an interpretation I iff for any affectation λ of the variables of H such that $I, \lambda \models \phi(H)$, there exists an affectation λ' of the variables of C that extends λ (i.e., if X is a frontier variable of R, then $\lambda(X) = \lambda'(X)$) such that $I, \lambda' \models \phi(C)$.*

Now our main problem becomes the ECQ-DEDUCTION problem. Given an ECQ Q and a knowledge base $\mathcal{K}$ containing a fact F and a set of rules $\mathcal{R}$, is Q entailed by $\mathcal{K}$ (*i.e.*, is every interpretation that is a model for F and every rule in $\mathcal{R}$ also a model of Q?). In what follows, we present the two great families of algorithm designed to answer that problem, forward and backward chaining.

1.2.2 Forward chaining

The Forward Chaining algorithm (also called *chase* in databases) enriches incrementally the fact F. The elementary operation of this algorithm is the *rule application* mechanism: every time the hypothesis of a rule is proven true in F (by an homomorphism), its conclusion is added to F. A derivation is a fact created by a succession of such rule applications.

Application of a rule Intuitively, to apply a rule is to know that the hypothesis is true in F (thanks to an homomorphism π), in order to add the conclusion $\pi(H)$ to the facts. This is a very standard mechanism. A subtle difference comes from the presence of existential variables in the conclusion. Suppose a rule $\text{human}(X) \to \text{hasParent}(X, Y)$ and a fact $\{\text{human(bob)}, \text{human(sam)}\}$. By mapping X to *bob*, we have a first proof of the hypothesis in F, and we can add $\text{hasParent}(\text{bob}, Y)$ to the fact. By mapping X to *sam*, we also add $\text{hasParent}(\text{sam}, Y)$

to the fact. Now, *bob* and *sam* have the same parent Y, which was not the intended meaning of the rule. In the following definition, we will take care to replace existential variables with fresh ones (say $Y0$ on the first application, and $Y1$ on the second).

Definition 12 (Application of a rule)**.** *A rule $R = (H, C)$ is said* applicable *to a fact F if there is a homomorphism π from H to F. The fact obtained after application of R is $\alpha(F, R, \pi) = F \cup \pi^{safe}(C)$ where $\pi^{safe}(C)$ is C in which we substituted the variables of $\mathtt{fr}(R)$ by their image by π and we renamed the others with fresh variables. $\alpha(F, R, \pi)$ is called an* immediate derivation *from F.*

Example 5. *Let $F = \{q(U), p(U, V), p(V, W)\}$ and $R = (\{q(X), p(X, Y)\}, \{q(Y), r(Y, W)\})$. R is applicable to F with the homomorphism $\pi = \{(X, U), (Y, V)\}$, and $\alpha(F, R, \pi) = F \cup \pi^{safe}(C) = F \cup \{q(V), r(V, W_0)\} = \{q(U), p(U, V), p(V, W), q(V), r(V, W_0)\}$.*

Definition 13 (Derivation)**.** *An $\mathcal{R}$-*derivation *of a fact F is either F or a fact obtained after application of a rule of $\mathcal{R}$ to an $\mathcal{R}$-derivation of F.*

As in the absence of rules, and for the same reasons, a knowledge base can never entails a non equalizable ECQ. The following theorem characterizes entailment of equalizable ECQs.

Theorem 2. *Let F be a fact and Q be an equalizable ECQ. $F, \mathcal{R} \models Q$ if and only if there exists F' an $\mathcal{R}$-derivation of F such that $F' \models Q$ (i.e. Q E-maps to F').*

Proof. The usual version of that theorem states that $F, \mathcal{R} \models Q$ (where Q is a CQ) iff Q maps to the (possibly infinite) fact built from the union of all facts created along a complete sequence of rule applications (complete meaning that all possible rule applications have been done at some time). Let us call F^* this fact. Since any derivation F is included in F^*, then Q maps to F^* whenever Q maps to F. Conversely, let us number the atoms of F^* according to their appearance in the sequence of rule applications. If π maps Q to F^*, $\pi(Q)$ is a finite subset of F^*, admitting an element with greater numbering N. All these elements will be created in a (finite)derivation using at most N rule applications. Then the homomorphism from Q to F^* is an homomorphism from Q to that derivation.

We conclude for equalizable ECQs by using proposition 4. □

Since rules can simulate Turing Machines, ECQ-DEDUCTION is an undecidable problem (even with a single rule and a vocabulary restricted to a single binary predicate). More precisely, the problem is semi-decidable: if we build our derivation in a *fair* way (for instance with a breadth-first ordering of rule applications, that will not miss any one of them) and test for a mapping of Q at each step, our algorithm will stop whenever $\mathcal{K} \models Q$. When there is no deduction, however, nothing ensures that the algorithm will halt.

To improve the chances of reaching a complete derivation, improvements have been added to the algorithm, by simplifying the fact after each rule application and thus trying to avoid infinite redundant parts. The *restricted chase* tries to *fold* the added conclusion to the previous fact after each rule application. The *core chase* computes the minimal equivalent fact after each rule application: though the simplification is optimal, the overhead cost is tremendous.

Finally, an important research effort has been dedicated to identifying decidable, yet expressive, subclasses of the CQ-DEDUCTION problem. Among them, *finite expansion sets* describe those for which a forward chaining algorithm halts. The reader can refer to [3] for a zoology of those decidable subclasses.

1.3 Backward chaining

Another method to determine if Q is entailed by $(F, \mathcal{R})$ is to rewrite the query until we find a query that is entailed by F. We rewrite Q into a query that, when combined with $\mathcal{R}$, entails Q. That is the backward chaining.

Rewriting of a query Given a query Q and a rule R, we can rewrite Q according to R if there is a part of Q that can be proven by an application of R. We thus produce a query composed of the rule hypothesis and of the part of Q not proven by the rule. To formalize this rewriting, the notion of piece-unifier is needed.

Definition 14 (Piece)**.** *Let Q be an ECQ and $S \subseteq \mathtt{var}(F)$ a* separator *of Q. A* piece *of Q according to S is a minimal non-empty subset Q' of Q such that if an atom of Q' (including equality atoms) contains a variable X that is not in S, then every atom of Q containing X is also in Q'.*

Example 6. *The pieces of $Q_0 = \{p(U,V), q(V,U), p(W,a), r(U,W)\}$ according to $\{U\}$ are $\{p(U,V), q(V,U)\}$ and $\{r(U,W), p(W,a)\}$; and the pieces of Q_0 according to $\{U,W\}$ are $\{p(U,V), q(V,U)\}, \{r(U,W)\}$ and $\{p(W,a)\}$.*

The pieces of $Q_1 = \{p(U,V),\ q(V,U), p(W,a), r(U,W), U = W\}$ according to $\{U\}$ are $\{p(U,V), q(V,U)\}$ and $\{r(U,W), p(W,a), U = W\}$; and the pieces of Q_1 according to $\{U,W\}$ are $\{p(U,V), q(V,U)\}$, $\{r(U,W)\}$, $\{p(W,a)\}$ and $\{U = W\}$.

Definition 15 (Cutpoints)**.** *Let $R = (H,C)$ be a rule. The cutpoints of R are either frontier variables or constants of C, noted $\mathtt{cutp}(R) := \mathtt{fr}(R) \cup \mathtt{const}(C)$.*

We can now introduce the notion of a *piece-unifier*, or unifier. Intuitively, a unifier defines the way some part of a query can be proven by a rule application. Since non equalizable ECQs will never be proven, whatever the rule applications involved, we will only be interested here in unifying equalizable ECQs.

Definition 16 (Piece-unifier)**.** *Let Q be an equalizable ECQ and $R = (H,C)$ be a rule. A* unifier *of Q with R is a tuple $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ where:*

- *T_Q is a (possibly empty) subset of $\mathtt{var}(Q)$, thus defining pieces of Q;*
- *Q' is the reunion of one or more pieces of Q according to T_Q;*
- *σ_R is a substitution of $\mathtt{fr}(R)$ by $\mathtt{cutp}(R) \cup \mathtt{const}(Q')$;*
- *π_Q is an E-homomorphism from Q' to $\sigma_R(C)$ such that for all $t \in T_Q \cap \mathtt{var}(Q')$, there exists $t' \in \mathtt{cutp}(R)$ with $\pi_Q(t) = \sigma_R(t')$;*

Analysis of the definition Q' is the part of Q that the rule R will be able to prove, σ_R is the *specialization*, i.e. it is the substitution that will transform the rule R into another rule R', better suited to the shape of Q', and of course satisfying $R \models R'$.

Finally, π_Q is the homomorphism that shows how Q' is deductible from the rule application. The first condition that comes with it is what is interesting though. Basically, to make pieces according to T_Q means that you only allow atoms to be separated at variables of T_Q, atoms that are linked another way *must* be "glued" in the same piece. The condition on π_Q forces that variables of T_Q that are separating the pieces of Q' from other pieces (all $t \in T_Q \cap \mathtt{var}(Q')$) must be sent on the specialization of a cutpoint of R, or equivalently on a cutpoint of R'. Remark that the cutpoints of a rule are precisely the terms of its conclusion that are not existantially quantified. The idea behind that condition is to make sure that all variables of Q that will be sent on existantially quantified variables must act as "glue" for the atoms of Q. It is necessary because for example, to prove $\exists V[p(U,V) \wedge q(V,U)]$, we can't separate these two atoms since there is the condition that the V is the same in both.

Definition 17 (E-rewriting of a query)**.** *Now that we have formalized the notion of unifier, we can finally formalize the notion of* rewriting *that was introduced at the start of this section.*
Let Q be an ECQ, $R = (H,C)$ a rule and $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ a unifier of Q with R. An E-rewriting of Q according to R and μ *produces an ECQ:*

$$\beta^{=}(Q,R,\mu) = (\pi_Q^{-1})^{safe} \circ \sigma_R(H) \cup Q \setminus Q' \cup E$$

where E is a set of equality predicates such that when there are several antecedents by π_Q, we choose one for π_Q^{-1} and map all the others to the chosen one with equality predicates (E is the choice of a representative for each equivalence class for the relation $X \in \mathtt{var}(Q) \sim Y \in \mathtt{var}(Q) \iff \pi_Q(X) = \pi_Q(Y)$).

The E-rewriting is a step in the process of obtaining the rewriting, that will be necessary when dealing with more complex queries.

Definition 18 (Rewriting of a query)**.** *Let Q be an ECQ, $R = (H,C)$ a rule and $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ a unifier of Q with R. A* rewriting of Q according to R and μ *is an* equalization *of $\beta^{=}(Q,R,\mu)$.*

Example 7. *Let $R = h(X, Y, Z, S) \to p(X, T) \wedge q(T, Y) \wedge p(Z, S)$ and $Q_0 = \{p(U, V), q(V, U), p(W, a), r(U, W)\}$.*

Let us find a piece-unifier of Q_0 with R that will prove $p(U, V)$ in Q_0. We can see that we can send $p(U, V)$ to $p(X, T)$ or $p(Z, S)$.

• We will first try to send it to $p(Z, S)$. We need to have $U \mapsto Z$ and $V \mapsto S$. Both Z and S are frontier variables, so we don't need to glue any atom to $p(U, V)$: thus we can just let $T_Q = \{U, V\}$. We didn't need to specialize the rule, so $\sigma_R = \texttt{id}$. The unifier is complete, and it produces the fact $\{h(X_0, Y_0, U, V), q(V, U), p(W, a), r(U, W)\}$.

• We will then try to send $p(U, V)$ to $p(X, T)$. We need to have $U \mapsto X$ and $V \mapsto T$. X is a frontier variable, but T is not, so we need to take all atoms in Q that contain V, i.e. $q(V, U)$. But now that $q(V, U)$ is also in Q', we need to send it in the conclusion of R. In this case the only choice is to send it to $q(T, Y)$, so we need $V \mapsto T$ and $U \mapsto Y$. $V \mapsto T$ was already chosen, so there is no problem. But we already had $U \mapsto X$. So we need to gather X and Y using the specialization, for example with $\sigma_R \colon Y \mapsto X$. Since Y is a frontier variable, we can stop here the expansion of $p(U, V)$ into a piece. To separate these two atoms from the rest of Q, we can cut at U, and let $T_Q = \{U\}$. This unifier produces the fact $\{h(U, U, Z_0, S_0), p(W, a), r(U, W)\}$.

• However, instead of stopping the expansion of $p(U, V)$ to only $\{p(U, V), q(V, U)\}$, we can try to also incorporate $p(W, a)$ in Q'. We need $W \mapsto Z$, and we need to specialize S to a. Since we can't prove $r(U, W)$ with R, we need to separate these two atoms, so by putting W in T_Q, and we can since it is sent to Z, a frontier variable (and so satisfies the fourth condition). This unifier produces the fact $\{h(U, U, W, a), r(U, W)\}$.

In these examples, we didn't need to equalize variables since π_Q was always injective. Let us show an example where it is needed, with $R = h(X) \to p(X, X)$ and $Q_0 = \{p(U, V), q(V, U)\}$. We want to send $p(U, V)$ to $p(X, X)$, so we need $U \to X$ and $V \to X$. X is a frontier variable so $\{p(U, V)\}$ is a piece. We didn't need to specialize the rule so $\sigma_R = \texttt{id}$. Since π_Q maps both U and V to X, we need to choose the value of $\pi_Q^{-1}(X)$, for example U. The unifier is complete and the equalizable-rewriting produces the equalizable query $(\{h(U)\} \cup \{q(V, U)\}, \{(V, U)\})$. The rewriting produces the equalized of this, so the query $\{h(U), q(U, U)\}$.

As announced in the introduction of this section, we have the following proposition:

Proposition 6. *Let F and Q be a fact and an ECQ, and R be a rule. Then Q E-maps to an application of R to F if and only if Q E-maps to F or there is an E-rewriting of Q according to R that E-maps to F (if and only if there is a rewriting of Q according to R that maps to F).*

Like with the application of rules, we can define the equivalent of the derivation for the backward chaining:

Definition 19 (Rewriting sequence)**.** *Let Q and Q' be two ECQs and $\mathcal{R}$ a set of rules. We say that Q' is an* $\mathcal{R}$-E-rewriting of Q *if there exists a finite sequence $Q = Q_0, \ldots, Q_k = Q'$ such that for all $0 \leq i \leq k - 1$, there exists a rule R_i and a unifier μ_i of Q_i with R_i such that $Q_{i+1} = \beta^{=}(Q_i, R_i, \mu_i)$.*

With a quick recursion, we obtain the following theorem:

Theorem 3. *Let F and Q be a fact and an ECQ and $\mathcal{R}$ be a set of rules. Then Q E-maps to an $\mathcal{R}$-derivation of F if and only if there is an $\mathcal{R}$-E-rewriting of Q that E-maps to F. With the theorem 2, we can obtain that: $F, \mathcal{R} \models Q$ if and only if there is an $\mathcal{R}$-E-rewriting of Q that E-maps to F.*

On an algorithmic standpoint, we want to build rewritings of Q until we find one that E-maps to F, but to prevent the considered set of rewritings to get to big, we only keep in it the most general rewritings (i.e. we prune the ones that are entailed by another rewriting).

2 Rewritings of disjunctive queries

2.1 Interest of the problem

As we have seen in the previous section, the algorithmic way to use the rewritings is to start from the original query and apply rewritings to the queries in the set of already found rewritings. The fact that we do this search breadth-first means that we need to keep the queries that are rewritten. For example, if a query Q can be separated in $Q_1 \wedge Q_2$ with Q_2 being the part rewritten into Q_3 according to a rule, we keep both the original query $(Q_1 \wedge Q_2)$ and the rewritten one $(Q_1 \wedge Q_3)$. This means that at each step of rewriting, the size of the set of queries can up to double, leading to a very large set of queries to test (we can see this set of queries as an union of conjuncive

queries — an UCQ). Since these queries come from the mechanism described above, we can think about factoring that UCQ to reduce its size, but it would be too time-expensive.

But since there is an intuitive factoring available in the rewriting process, we want to try to make use of it to control the size of the set during the rewriting rather than after.

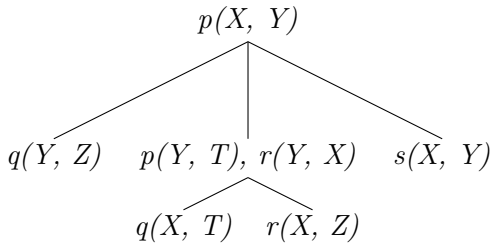
This factoring means that the rewriting process will have to be able to take as input queries that are in disjunction-conjunction shape, and this will be the object of what follows.

2.2 Conjunctive-Disjunctive Queries

2.2.1 Syntax

Definition 20 (ECDQ)**.** *As introduced, we define a* Conjunctive-Disjunctive Query with equality (or ECDQ)*: it is a (finite) tree where the nodes are conjuncts. ECDQs without equality are the CDQs. To represent trees, we use the prefix notation:* $(root, subtree_1, \ldots, subtree_n)$*, where nodes with no subtrees are leaves. We denote by* $\mathcal{ECDQ}$ *the set of all the ECDQs and* $\mathcal{CDQ}$ *the set of all the CDQs.*

Example 8. $\mathcal{Q}_0 = (Q_1, Q_2, (Q_3, Q_5, Q_6), Q_4)$ *where* $Q_1 = \{p(X,Y)\}$*,* $Q_2 = \{q(Y,Z)\}$*,* $Q_3 = \{p(Y,T), r(Y,X)\}$*,* $Q_4 = \{s(X,Y)\}$*,* $Q_5 = \{q(X,T)\}$ *and* $Q_6 = \{r(X,Z)\}$ *is a CDQ. It is represented below:*



2.2.2 Semantic

Definition 21 (Formula of an ECDQ)**.** *Let* Q *be an ECDQ, and* T *a set of variables. The formula of* Q *according to* T*, noted* $\phi(Q,T)$*, is defined by induction as follows:*

- $\phi(Q,T) = \underset{X \in \mathtt{var}(Q) \setminus T}{\exists X} Q$ *if* Q *is a conjunct (a leaf);*
- $\phi(Q,T) = \underset{X \in \mathtt{var}(R) \setminus T}{\exists X} [R \wedge [\bigvee_{1 \leq i \leq n} \phi(S_i, T \cup \mathtt{var}(R))]]$ *if* $Q = (R, S_1, \ldots, S_n)$.

The semantic of an ECDQ is its formula according to $\emptyset$ ($\phi(Q,\emptyset)$, also simply noted $\phi(Q)$).

Example 9. *Let us build the formula of $\mathcal{Q}_0$ according to $\emptyset$ step by step:*

$$\begin{aligned}
\mathcal{Q}_0 &= (Q_1, Q_2, (Q_3, Q_5, Q_6), Q_4) \text{ so} \\
\phi(\mathcal{Q}_0, \emptyset) &= \underset{X \in \mathtt{var}(Q_1)}{\exists X} [Q_1 \wedge [\phi(Q_2, \mathtt{var}(Q_1)) \vee \phi((Q_3, Q_5, Q_6), \mathtt{var}(Q_1)) \vee \phi(Q_4, \mathtt{var}(Q_1))]] \\
&= \exists X \exists Y [Q_1 \wedge [\phi(Q_2, \{X, Y\}) \vee \phi((Q_3, Q_5, Q_6), \{X, Y\}) \vee \phi(Q_4, \{X, Y\})]]
\end{aligned}$$

$$Q_2 = \{q(Y, Z)\} \text{ so } \phi(Q_2, \{X, Y\}) = \exists Z q(Y, Z)$$

$$Q_4 = \{s(X, Y)\} \text{ so } \phi(Q_4, \{X, Y\}) = s(X, Y)$$

$$\begin{aligned}
\phi((Q_3, Q_5, Q_6), \{X, Y\}) &= \underset{X \in \mathtt{var}(Q_3) \setminus \{X,Y\}}{\exists X} [Q_3 \wedge [\phi(Q_5, \{X, Y\} \cup \mathtt{var}(Q_3)) \vee \phi(Q_6, \{X, Y\} \cup \mathtt{var}(Q_3))]] \\
&= \underset{X \in \{X,Y,T\} \setminus \{X,Y\}}{\exists X} [Q_3 \wedge [\phi(Q_5, \{X, Y, T\}) \vee \phi(Q_6, \{X, Y, T\})]] \\
&= \exists T [Q_3 \wedge [\phi(Q_5, \{X, Y, T\}) \vee \phi(Q_6, \{X, Y, T\})]]
\end{aligned}$$

$$Q_5 = \{q(X, T)\} \text{ so } \phi(Q_5, \{X, Y, T\}) = q(X, T)$$

$$Q_6 = \{r(X, Z)\} \text{ so } \phi(Q_6, \{X, Y, T\}) = \exists Z r(X, Z)$$

We have thus $\phi((Q_3, Q_5, Q_6), \{X, Y\}) = \exists T [p(Y, T) \wedge r(Y, X) \wedge [q(X, T) \vee \exists Z r(X, Z)]]$

$$\begin{aligned}
\textit{Finally, } \phi(\mathcal{Q}_0, \emptyset) = \exists X \exists Y [p(X, Y) \wedge [&\exists Z q(Y, Z) \\
&\vee \exists T [p(Y, T) \wedge r(Y, X) \wedge [q(X, T) \vee \exists Z r(X, Z)]] \\
&\vee s(X, Y)]]
\end{aligned}$$

Remark. *Two occurences of the same variable that do not have a parental link in the ECDQ (in this case Z in Q_2 and Q_6) are independant, we can rename one without changing the semantic of the ECDQ.*

2.2.3 ECDQ flattening

To link ECDQs with UCQs (or rather with UECQs), we rearrange a ECDQ (of course while retaining an equivalent semantic) until we produce an UECQ. Let us examine a rearranging of an ECDQ such that the semantic is unchanged modulo an equivalence:

Definition 22 (Flattening of an ECDQ)**.** *Let Q be an ECDQ. The* flattening *of Q, noted $\mathtt{Flat}(Q)$ is defined by induction:*

- $\mathtt{Flat}(Q) = Q$ *if Q is a conjunct (a leaf);*
- $\mathtt{Flat}(Q) = (\emptyset, S'_1, \ldots, S'_n)$ *if $Q = (R, S_1, \ldots, S_n)$ with $R \neq \emptyset$, where S'_i is obtained by joining R to the root of S_i. Formally, if $S_i = (R_i, S_{i,1}, \ldots, S_{i,n_i})$, $S'_i = (R \cup R_i, S_{i,1}, \ldots, S_{i,n_i})$ (with of course $n_i = 0$ if S_i is a conjunct).*
- $\mathtt{Flat}(Q) = (\emptyset, \mathtt{Flat}(S_1), \ldots, \mathtt{Flat}(S_n))$ *if $Q = (\emptyset, S_1, \ldots, S_n)$*

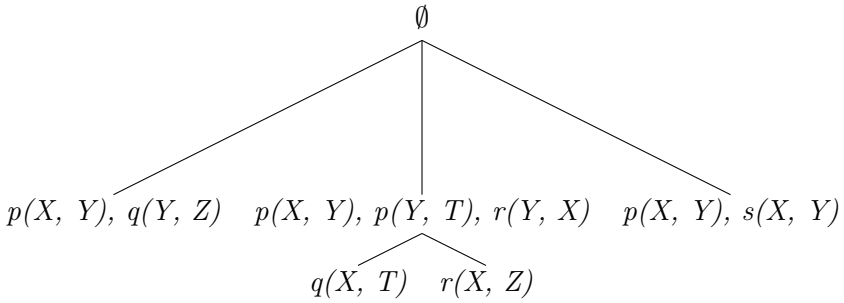
When we consecutively apply the flattening process to an ECDQ, we obtain an ECDQ where all internal nodes are empty, which we can see as an UECQ.

Definition 23 (Complete flattening)**.** *Let Q be an ECDQ. The* complete flattening *of Q, noted $\mathtt{Flat}^*(Q)$, is the result of consecutive applications of the $\mathtt{Flat}$ operator to Q until all internal nodes are empty. It is a finite process (bounded by the height of Q, so $\mathtt{Flat}^*(Q) = \mathtt{Flat}^{height(Q)}(Q)$).*

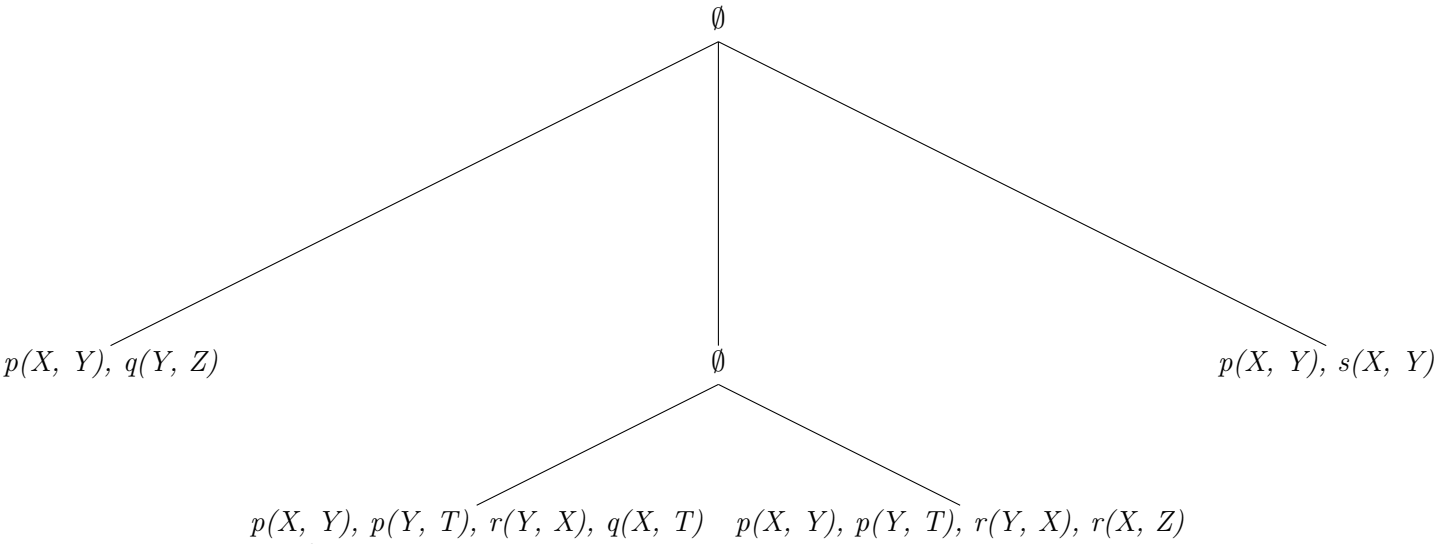
Since the result of a complete flattening is a tree where all internal nodes are empty, when we compute its semantic, we obtain a simple overlapping of conjunctions of conjuncts, which is equivalent to a conjunction of conjuncts, i.e. a tree of height 1 with an empty root whose sons are the leaves of the complete flattening. That tree of height 1 is trivially equivalent to an UECQ.

Example 10. *To illustrate the complete flattening, we will apply it on* $\mathcal{Q}_0$*:*

$\texttt{Flat}(\mathcal{Q}_0)$ *is:*



$\texttt{Flat}(\texttt{Flat}(\mathcal{Q}_0))$ *is:*



Thus $\texttt{Flat}^*(\mathcal{Q}_0) = \texttt{Flat}^2(\mathcal{Q}_0)$.

Proposition 7. *Let* Q *be an ECDQ. Then* $\phi(Q) \equiv \phi(\texttt{Flat}(Q))$.

Proof. The proof is done by induction.

- If Q is a conjunct, $\texttt{Flat}(Q) = Q$ so the result is trivial.
- If $Q = (R \neq \emptyset, S_1, \ldots, S_n)$, $\texttt{Flat}(Q) = (\emptyset, S'_1, \ldots, S'_n)$ with the S'_i as described above. We have thus:
  $\phi(Q) = \underset{X \in \texttt{var}(R)}{\exists X}[R \wedge [\bigvee_{1 \leq i \leq n} \phi(S_i, \texttt{var}(R))]]$ and
  $\phi(\texttt{Flat}(Q)) = \bigvee_{1 \leq i \leq n} \phi(S'_i)$

  First direction Let I be an interpretation in which $\phi(Q)$ is true. Then there exists λ an affectation of $\texttt{var}(R)$ such that $I, \lambda \models R$ and there is $1 \leq i \leq n$ such that $I, \lambda \models \phi(S_i, \texttt{var}(R))$.

  Let us note $S_i = (R_i, S_{i,1}, \ldots, S_{i,n_i})$. We have thus:
  $\phi(S_i, \texttt{var}(R)) = \underset{X \in \texttt{var}(R_i) \setminus \texttt{var}(R)}{\exists X}[R_i \wedge [\bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \texttt{var}(R) \cup \texttt{var}(R_i))]]$.

  Since $I, \lambda \models \phi(S_i, \texttt{var}(R))$, we can extend λ to an affectation of $\texttt{var}(R) \cup \texttt{var}(R_i)$ λ' such that $I, \lambda' \models R_i$ and $I, \lambda' \models \bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \texttt{var}(R) \cup \texttt{var}(R_i)) = \bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \texttt{var}(R \cup R_i))$.

  From the notation of S_i, we have $S'_i = (R \cup R_i, S_{i,1}, \ldots, S_{i,n_i})$.

So $\phi(S'_i) = \underset{X \in \mathtt{var}(R \cup R_i)}{\exists X} [(R \cup R_i) \wedge [\bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \mathtt{var}(R \cup R_i))]]$.

We have $I, \lambda' \models R$ and $I, \lambda' \models R_i$ so $I, \lambda' \models (R \cup R_i)$, and $I, \lambda' \models \bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \mathtt{var}(R \cup R_i))$. Since λ' is an affectation of $\mathtt{var}(R) \cup \mathtt{var}(R_i) = \mathtt{var}(R \cup R_i)$, we have proven $I \models \phi(S'_i)$, so $I \models \phi(\mathtt{Flat}(Q))$.

Second direction Let I be an interpretation in which $\phi(\mathtt{Flat}(Q))$ is true. Then there exists $1 \leq i \leq n$ such that $I \models \phi(S'_i)$.

Let us note $S_i = (R_i, S_{i,1}, \ldots, S_{i,n_i})$. We have thus $S'_i = (R \cup R_i, S_{i,1}, \ldots, S_{i,n_i})$.

So $\phi(S'_i) = \underset{X \in \mathtt{var}(R \cup R_i)}{\exists X} [(R \cup R_i) \wedge [\bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \mathtt{var}(R \cup R_i))]]$.

So there exists λ an affectation of $\mathtt{var}(R \cup R_i)$ such that $I, \lambda \models R \cup R_i$ and $I, \lambda \models \bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \mathtt{var}(R \cup R_i))$.

Let us note λ' the restriction of λ to $\mathtt{var}(R)$. Then we have $I, \lambda' \models R$. We still need to prove that $I, \lambda' \models \phi(S_i, \mathtt{var}(R))$.

$\phi(S_i, \mathtt{var}(R)) = \underset{X \in \mathtt{var}(R_i) \setminus \mathtt{var}(R)}{\exists X} [R_i \wedge [\bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \mathtt{var}(R) \cup \mathtt{var}(R_i))]]$.

By extending λ' to λ, an affectation of $\mathtt{var}(R) \cup \mathtt{var}(R_i)$, we have $I, \lambda \models R_i$ and $I, \lambda \models \bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \mathtt{var}(R \cup R_i)) = \bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \mathtt{var}(R) \cup \mathtt{var}(R_i))$.

We have $I, \lambda \models R_i$ and $I, \lambda \models \bigvee_{1 \leq j \leq n_i} \phi(S_{i,j}, \mathtt{var}(R) \cup \mathtt{var}(R_i))$ so $I, \lambda' \models \phi(S_i, \mathtt{var}(R))$.

We have $I, \lambda' \models R$ and $I, \lambda' \models \phi(S_i, \mathtt{var}(R))$ and λ' is an affectation of $\mathtt{var}(R)$, so we have proven $I \models \phi(Q)$.

Thus $\phi(Q) \equiv \phi(\mathtt{Flat}(Q))$.

- If $Q = (\emptyset, S_1, \ldots, S_n)$, $\mathtt{Flat}(Q) = (\emptyset, \mathtt{Flat}(S_1), \ldots, \mathtt{Flat}(S_n))$. We have thus:
  $\phi(Q) = \bigvee_{1 \leq i \leq n} \phi(S_i)$ and $\phi(\mathtt{Flat}(Q)) = \bigvee_{1 \leq i \leq n} \phi(\mathtt{Flat}(S_i))$.
  With the induction hypothesis, we have that $\forall i \in [\![1; n]\!] *, \phi(S_i) \equiv \phi(\mathtt{Flat}(S_i))$, so $\phi(Q) \equiv \phi(\mathtt{Flat}(Q))$.

□

Corollary 1. *Let Q be an ECDQ. Then $\phi(Q) \equiv \phi(\mathtt{Flat}^*(Q))$.*

2.3 ECD-homomorphisms

Definition 24 (ECD-homomorphism)**.** *Let Q be an ECDQ and F be a fact. Let P be a* complete path *of Q (we define by that a path in Q starting at its root and ending at one of its leaves). We identify a complete path of Q to an ECQ by considering the reunion of the conjuncts at its nodes.*

An ECD-homomorphism *from Q to F is a E-homomorphism from a complete path of Q to F.*

If there exists an ECD-homomorphism from an ECDQ to a fact F, we say that that ECDQ ECD-maps *to F.*

Remark. *Let Q be an ECQ and F be a fact. We can see Q as an ECDQ that is simply a leaf. An ECD-homomorphism from Q to F is exactly an E-homomorphism from Q to F and vice versa.*

Remark. *We can notice that for an ECDQ Q, a leaf of $\mathtt{Flat}^*(Q)$ is equivalent to a complete path of Q. This proves the following property (but there is an alternative proof in annex).*

Proposition 8. *Let Q be an ECDQ and F be a fact. Then Q ECD-maps to F if and only if $\mathtt{Flat}^*(Q)$ ECD-maps to F.*

Theorem 4 (ECD-homomorphism)**.** *Let F be a fact and Q be an ECDQ. Then there is an ECD-homomorphism from Q to F if and only if $F \models \phi(Q)$.*

Proof. We proceed by consecutive equivalences:

- We begin by noticing that with corollary 1, we have $F \models \phi(Q) \iff F \models \phi(\texttt{Flat}^*(Q))$. Indeed, for each interpretation I such that $I \models F$, $I \models \phi(\texttt{Flat}^*(Q)) \iff I \models \phi(Q)$.
- Since we can see $\texttt{Flat}^*(Q)$ as an UECQ, we can use theorem 1 to interpret $F \models \phi(\texttt{Flat}^*(Q))$ as "there exists an E-homomorphism from an ECQ of $\texttt{Flat}^*(Q)$ to F". But that ECQ is simply a leaf of $\texttt{Flat}^*(Q)$, so with the previous remark, it is also a complete path of Q. Thus that homomorphism is an ECD-homomorphism from Q to F. This means that $F \models \texttt{Flat}^*(Q)$ if and only if Q ECD-maps to F.

□

Theorem 5. *We can give the equivalent of theorem 2 for ECDQs. Let Q be an ECDQ, F be a fact and $\mathcal{R}$ be a set of rules. Then $F, \mathcal{R} \models \phi(Q)$ if and only if there is an $\mathcal{R}$-derivation F' of F such that Q ECD-maps to F'.*

Proof. The proof follows the same structure as the one for theorem 4

- $F, \mathcal{R} \models \phi(Q) \iff F, \mathcal{R} \models \phi(\texttt{Flat}^*(Q))$
- Since $\texttt{Flat}^*(Q)$ is an UECQ, with theorem 2 we have $F, \mathcal{R} \models \phi(\texttt{Flat}^*(Q))$ if and only if there exists F' an $\mathcal{R}$-derivation of F and C an ECQ of $\texttt{Flat}^*(Q)$ such that C E-maps to F'. That C is a leaf of $\texttt{Flat}^*(Q)$ so a complete path of Q, so $F, \mathcal{R} \models \phi(\texttt{Flat}^*(Q))$ if and only if there exists F' an $\mathcal{R}$-derivation of F such that Q ECD-maps to F'.

□

2.4 ECD-rewritings

We want to define a mechanism to rewrite ECDQs into ECDQs that will have the same results that E-rewriting has.

2.4.1 Correct ECD-rewriters

Definition 25 (Correct ECD-rewriter)**.** *Let $\mathcal{Q}$ be a set of ECDQs and $\mathcal{R}$ a set of rules. A* correct ECD-rewriter of $\mathcal{Q}$ according to $\mathcal{R}$ *is a function CD-$\beta^=$ from $\mathcal{Q} \times \mathcal{R}$ to $\mathcal{P}(\mathcal{Q})$ that upholds the two following properties:*

(CECDR i) For any fact F, rule $R \in \mathcal{R}$ applicable to F and ECDQ $Q \in \mathcal{Q}$, if Q ECD-maps to an application of R to F then Q ECD-maps to F or there is $Q' \in$ CD-$\beta^=(Q, R)$ that ECD-maps to F.

(CECDR ii) For any fact F, rule $R \in \mathcal{R}$ and ECDQ $Q \in \mathcal{Q}$, if there is $Q' \in$ CD-$\beta^=(Q, R)$ that ECD-maps to F, then Q ECD-maps to F or to an application of R to F.

For $Q \in \mathcal{Q}$ and $R \in \mathcal{R}$, the elements of CD-$\beta(Q, R)^=$ are called ECD-rewritings of Q according to R. *For $Q \in \mathcal{Q}$ and $\mathcal{R}' \subseteq \mathcal{R}$, a* $\mathcal{R}'$-ECD-rewriting of Q *is an ECDQ obtained by a sequence of ECD-rewritings according to rules of $\mathcal{R}'$ that starts at Q.*

The two (CECDR) properties allow the following theorem on correct ECD-rewriters:

Theorem 6. *Let $\mathcal{R}$ be a set of rules, $\mathcal{Q}$ a set of ECDQs and CD-β a correct ECD-rewriter of $\mathcal{Q}$ according to $\mathcal{R}$. Then for any fact F, set of rules $\mathcal{R}' \subseteq \mathcal{R}$ and ECDQ $Q \in \mathcal{Q}$, Q ECD-maps to an $\mathcal{R}'$-derivation of F if and only if there is a $\mathcal{R}'$-ECD-rewriting of Q that ECD-maps to F.*

Proof. The proof is done with a recursion for both directions.

First direction Let us prove that $\mathcal{P}(n)$: "If Q ECD-maps to an $\mathcal{R}'$-derivation of F of length n, then there is a $\mathcal{R}'$-ECD-rewriting of Q that ECD-maps to F" is true for $n \in \mathbb{N}$.

- Base case: if $n = 0$, suppose that Q ECD-maps to F (any $\mathcal{R}'$-derivation of F of length 0 is F). Then since Q is a $\mathcal{R}'$-ECD-rewriting of Q that maps to F, we have $\mathcal{P}(0)$.
- Heredity: let $n \in \mathbb{N}$ such that $\mathcal{P}(n)$. Suppose that Q ECD-maps to an $\mathcal{R}'$-derivation of F of length $n+1$. Then there is F' an immediate derivation of F according to a rule $R \in \mathcal{R}'$ such that Q ECD-maps to an $\mathcal{R}'$-derivation of F' of length n. Since we have $\mathcal{P}(n)$, we know that there is Q' a $\mathcal{R}'$-ECD-rewriting of Q that ECD-maps to F'.

  With the property (CECDR i), we have that either Q' ECD-maps to F or there is $Q'' \in \text{CD-}\beta^{=}(Q', R)$ that ECD-maps to F. Since $R \in \mathcal{R}'$ and Q' is a $\mathcal{R}'$-ECD-rewriting of Q, Q'' is als() $\mathcal{R}'$-ECD-rewriting of Q.

  We have proven $\mathcal{P}(n+1)$, so $\mathcal{P}(n)$ is true for all n.

Second direction Let us prove that $\mathcal{P}'(n)$: "If there is a $\mathcal{R}'$-ECD-rewriting of Q of length n that ECD-maps to F, then Q ECD-maps to an $\mathcal{R}'$-derivation of F" is true for $n \in \mathbb{N}$.

- Base case: if $n = 0$, suppose that Q ECD-maps to F (any $\mathcal{R}'$-ECD-rewriting of Q of length 0 is Q). Then since F is an $\mathcal{R}'$-derivation of F, Q ECD-maps to an $\mathcal{R}'$-derivation of F, so $\mathcal{P}'(0)$.
- Heredity: let $n \in \mathbb{N}$ such that $\mathcal{P}(n)$. Suppose that there is a $\mathcal{R}'$-ECD-rewriting of Q of length $n+1$ that ECD-maps to F. Then there is Q' an ECD-rewriting of Q according to a rule $R \in \mathcal{R}'$ such that there is a $\mathcal{R}'$-ECD-rewriting of Q' of length n that ECD-maps to F. Since we have $\mathcal{P}'(n)$, then Q' ECD-maps to F' an $\mathcal{R}'$-derivation of F.

  With the property (CECDR ii), we have that Q ECD-maps to F' or to F'' an application of R to F'. Since $R \in \mathcal{R}'$ and F' is a $\mathcal{R}'$-derivation of F, F'' is also a $\mathcal{R}'$-derivation of F.

  We have proven $\mathcal{P}'(n+1)$, so $\mathcal{P}'(n)$ is true for all n.

□

The goal will be to find correct ECD-rewriters of $\mathcal{ECDQ}$ according to $\forall\exists$-$\mathcal{R}$.

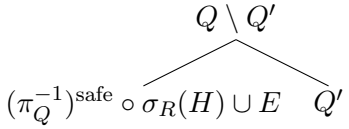
Remark. *With proposition 6 and the remark that for ECQs, ECD-homomorphisms and E-homomorphisms are the same, we have that $\tilde{\beta}$ is a correct ECD-rewriter of $\mathcal{ECQ}$ according to $\forall\exists$-$\mathcal{R}$ where $\tilde{\beta}$ is defined as follows:*

$$\begin{aligned}\tilde{\beta}\colon \mathcal{ECQ} \times \forall\exists\text{-}\mathcal{R} &\to \mathcal{P}(\mathcal{ECQ}) \\ (Q, R) &\mapsto \{\beta^{=}(Q, R, \mu) | \mu \textit{ is a unifier of } Q \textit{ with } R\}\end{aligned}$$

2.4.2 Construction of a CECDR of the ECDQs according to the $\forall\exists$-rules

We want to build CD-$\beta^{=}$ a correct ECD-rewriter of $\mathcal{ECDQ}$ according to $\forall\exists$-$\mathcal{R}$. Let us begin by defining CD-$\beta^{=}$ on $\mathcal{ECQ} \times \forall\exists$-$\mathcal{R}$.

Let Q, R be in $\mathcal{ECQ} \times \forall\exists$-$\mathcal{R}$. For each $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ unifier of Q with R, we build an ECDQ CD-$\beta^{=}(Q, R, \mu) = (Q \setminus Q', \beta^{=}(Q, R, \mu), Q')$, depicted below:



We then define CD-$\beta^{=}(Q, R) = \{\text{CD-}\beta^{=}(Q, R, \mu) | \mu \text{ is a unifier of } Q \text{ with } R\}$. We can prove (CECDR i) and (CECDR ii) for that ECD-rewriter on this domain restriction:

Proof. Let F be a fact, R a rule applicable to F and Q an ECQ.

(CECDR i) Suppose that Q ECD-maps to an application of R to F. Since Q is an ECQ, it E-maps to that application of R to F. According to proposition 6, either Q E-maps to F or there is a E-rewriting of Q according to R and μ that E-maps to F, with μ an unifier of Q with R. In the first case, we have Q ECD-maps to F since Q is an ECQ. In the second case, for that $\mu = (T_Q, Q', \sigma_R, \pi_Q)$, there is CD-$\beta^=(Q, R, \mu) \in$ CD-$\beta^=(Q, R)$ that ECD-maps to F. Indeed, one of its complete paths is $Q \setminus Q' \cup (\pi_Q^{-1})^{\text{safe}} \circ \sigma_R(H) \cup E = \beta^=(Q, R, \mu)$, which is the E-rewriting that E-maps to F that the proposition 6 produced.

(CECDR ii) Suppose that there is $Q' \in$ CD-$\beta^=(Q, R)$ that ECD-maps to F. Since $Q' \in$ CD-$\beta^=(Q, R)$, there exists $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ an unifier of Q with R such that $Q' =$ CD-$\beta^=(Q, R, \mu) = (Q \setminus Q', (\pi_Q^{-1})^{\text{safe}} \circ \sigma_R(H) \cup E, Q')$. Q' ECD-maps to F, so we have two cases:

- $Q \setminus Q' \cup (\pi_Q^{-1})^{\text{safe}} \circ \sigma_R(H) \cup E$ E-maps to F. Then we have $\beta^=(Q, R, \mu)$ that E-maps to F so with proposition 6, Q E-maps to an application of R to F.
- $Q \setminus Q' \cup Q'$ E-maps to F. Then Q E-maps to F.

In both cases, since Q is an ECQ, Q ECD-maps to F or to an application of R to F. □

Let us now define CD-$\beta^=$ on $\mathcal{ECDQ} \times \forall\exists\text{-}\mathcal{R}$. For that we need the notion of CD-unifier:

Definition 26 (CD-unifier)**.** *Let Q, R be in $\mathcal{ECDQ} \times \forall\exists\text{-}\mathcal{R}$. A* CD-unifier of Q with R *is a unifier of a complete path of Q with R.*

Definition 27 (ECD-rewriting of an ECDQ, part I)**.** *Now that we have the notion of CD-unifier we can define an* ECD-rewriting. *Let Q, R be in $\mathcal{ECDQ} \times \forall\exists\text{-}\mathcal{R}$ and $\mu = (T_P, P', \sigma_R, \pi_P)$ a CD-unifier of Q with R (P is a complete path of Q). We first consider the case where the rewritten part of Q, P' is contained within a single node, Q_0. In this case, the* ECD-rewriting of Q according to R and μ *is an ECDQ constructed as follows:*

- *We separate Q_0 by deleting P' from its root and inserting it a single son whose root is P' and whose sons are its previous sons. After that, we have an ECDQ equivalent to Q that has a single node containing all and exactly P'.*
- *We can now include the rewriting by giving to P' a brother whose root is $(\pi_Q)^{-1} \circ \sigma_R(H) \cup E$ and whose sons are a copy of the sons of P'.*

Definition 28 (ECD-rewriting of an ECDQ, part II)**.** *In the general case, we transform Q to get to the first case:*

- *Along the path P, we identify the first and last nodes (noted Q_{start} and Q_{end}) whose intersection with P' is not empty.*
- *We build from Q a new, equivalent ECDQ by flattening the parts of P' from the subtree whose root is Q_{start} along P down to Q_{end}. That is to say, we apply a partial* `Flat` *operator (a* `Flat` *that instead of lowering the whole root, only lowers the part of the root that is in P') on the nodes of P between Q_{start} (included) and Q_{end} (not included), from top to bottom.*
- *Once this has been done, we have an ECDQ equivalent to Q where P' (the part to be rewritten) is gathered in a single node (Q_{end}).*

3 Algorithms

3.1 Evaluation (ECD-Homomorphisms enumeration)

Algorithm 1: Get, Define, In

def *Get*(h, U, t) **:**
  return h[*Find(U, t)*]
def *Define*(h, U, t, v) **:**
  $h[\text{Find}(U, t)] = v$
def *In*(h, U, t) **:**
  return *Find*$(U, t) \in h$

Algorithm 2: VarNec-ECDQ(Q)

Data: An ECDQ $Q = (C, Q_1, \ldots, Q_n)$
Result: Q pointed with the set of joint variables for each conjunct.

Algorithm 3: ECDQ-Answers($Q, U, AV, F, \mathcal{V}, H, h_a, h_j$)

Data: A pointed ECDQ $Q = ((C, JV), \mathcal{X}, Q_1, \ldots, Q_n)$, a UnionFind structure U, a set of Answer Variables AV, a fact F on a vocabulary $\mathcal{V}$, a SearchTree structure H, a partial homomorphism (h_a, h_j) separated between the answer variables and the joint variables.
Result: Add to H the enumeration of the possible expansions of (h_a, h_j) to an ECD-homomorphism from Q to F. Furthermore, if h_a is total (defines all answer variables), returns $True$ if and only if there we find a proof for (h_a, h_j) in Q.

```
C_s, C_e = Split(C)
if not UpdateHom(Merge(h_a, h_j), U, C_e) then
    return False
UpdateU(Merge(h_a, h_j), U, C_e)
if AllMarked(C_s) then
    if n == 0 then
        AddH(H, h_a)
        return True
    else
        if h_a not total then
            for Q' ∈ Q.sons do
                ECDQ-Answers(Q', U, AV, F, V, H, h_a, h_j)
        else
            if InH(H, h_a) then
                return True
            for Q' ∈ Q.sons do
                if ECDQ-Answers(Q', U, AV, F, V, H, h_a, h_j) then
                    return True
            return False
else
    a = ChooseAndMark(C_s)
    if h_a not total then
        for (h'_a, h'_j) ∈ Filter(F, V, a, h_a, h_j, U, AV, JV) do
            ECDQ-Answers(Q, U, AV, F, V, H, h'_a, h'_j)
        Unmark(a)
    else
        if InH(H, h_a) then
            Unmark(a)
            return True
        for (h'_a, h'_j) ∈ Filter(F, V, a, h_a, h_j, U, AV, JV) do
            if ECDQ-Answers(Q, U, AV, F, V, H, h'_a, h'_j) then
                Unmark(a)
                return True
        Unmark(a)
        return False
```

Algorithm 4: Filter($F, \mathcal{V}, a, h_a, h_j, U, AV, JV$)

Data: A fact F on a vocabulary $\mathcal{V}$, an standard atom a, a partial homomorphism (h_a, h_j), a UnionFind structure U, a set of Answer Variables AV and of Joint Variables JV
Result: An enumeration of the (h'_a, h'_j) that are an extension (that agrees with U) from (h_a, h_j) to a partial homomorphism from a to an standard atom of F or $\mathcal{V}$

Algorithm 5: UpdateHom(h, U, E)

Data: A partial homomorphism h, a UnionFind structure U and a set of equality predicates E

Result: For each equality predicates in E where at least one term is known by h, adds to h the other term (or breaks if incompatible). If it breaks, it returns $False$, otherwise returns $True$.

```
for a ∈ E do
    (t = t') = a
    if In(h, U, t) ∧ In(h, U, t') then
        if Get(h, U, t)! = Get(h, U, t') then
            return False
    else
        if In(h, U, t) then
            Define(h, U, t', Get(h, U, t))
        else
            if In(h, U, t') then
                Define(h, U, t, Get(h, U, t'))
return True
```

Algorithm 6: UpdateU(h, U, E)

Data: A partial homomorphism h, a UnionFind structure U and a set of equality predicates E.

Result: Updates the UnionFind with every equality predicates where no term is known by h.

```
for a ∈ E do
    (t = t') = a
    if not In(h, U, t)∧ not In(h, U, t') then
        Union(U, t, t')
```

3.2 Rewriting (CD-Unifiers enumeration)

Algorithm 7: ECDQ-Unifiers$(Q, U, R, \mathcal{V}, h, \sigma, \mathcal{X}, \mathcal{F}, \mathcal{EL})$

Data: An ECDQ $Q = (C, Q_1, \ldots, Q_n)$, a UnionFind structure U, a rule $R = (B, H, fr)$, a vocabulary $\mathcal{V}$, a partial homomorphism h, a specialization σ, $\mathcal{X}$ the set of variables sent by h outside fr, a set of variables that must be sent to fr $\mathcal{F}$ and a set of atoms to be evaluated later $\mathcal{EL}$.

Result: Enumerates the minimal unificators created from (h, σ) according to U between Q (plus the atoms of $\mathcal{EL}$) and R that assign variables of $\mathcal{F}$ only to fr.

```
C_s, C_e = Split(C)
if not UpdateHom(h, U, C_e) then
    return
UpdateU(h, U, C_e)
if AllMarked(C_s) then
    if n == 0 then
        yield (h, σ)
    else
        for Q' ∈ Q.sons do
            yield from ECDQ-Unifiers(Q', U, R, V, h, σ, X, F, EL)
else
    a = ChooseAndMark(C_s)
    if h = ∅ then
        // We skip this atom
        U' = Copy(U)
        F' = Copy(F) ∪ var(a) // It means we can't assign variables from this atom to non-frontier variables
        yield from ECDQ-Unifiers(Q, U', R, V, ∅, ∅, ∅, F', ∅)
        // We begin the piece with this atom
        for (h', σ', X') ∈ Filter'(H, V, a, ∅, ∅, U, ∅, ∅) do
            if X' = ∅ then
                yield (h', σ') // If the first atom of the piece does not send any variable outside the frontier, the piece is complete
            else
                U' = Copy(U)
                F' = Copy(F)
                yield from ECDQ-Unifiers(Q, U', R, V, h', σ', X', F', ∅)
    else
        if X ∩ var(a) = ∅ then
            yield from ECDQ-Unifiers(Q, U, R, V, h, σ, X, F, EL ∪ {a}) // If we have no immediate need to take the atom, it is sent to the "evaluated later" set
        else
            // Here is the tricky part: we need to evaluate this atom because it contains a variable sent outside the frontier. Since evaluating an atom may send additional variables outside the frontier, we need to recursively search the "evaluated later" set and evaluate the atoms that have to. That is what ECDQ-Evaluator should do. Importantly, there is not a unique separation in EL between the atoms to evaluate and the others. Indeed, depending on the evaluation of an atom, another may or may not need to be evaluated itself.
            for (h', σ', X') ∈ Filter'(H, V, a, h, σ, U, X, F) do
                EN', EL' = Separate(EL, X') // Separates EL into a set of atoms to be "evaluated now" (because they contain a variable of X') and the others.
                U' = Copy(U)
                F' = Copy(F)
                yield from ECDQ-Evaluator(EN', EL', H, V, h', σ', X ∪ X', F', Q, U', R)
        Unmark(a)
```

Algorithm 8: Filter'$(H, fr, \mathcal{V}, a, h, \sigma, U, \mathcal{X}, \mathcal{F})$

Data: A conjunct H, a set of variables fr, a vocabulary $\mathcal{V}$, an atom a, a partial homomorphism h, a specialization σ, a UnionFind structure U and two sets of variables $\mathcal{X}$ and $\mathcal{F}$

Result: An enumeration of the $(h', \sigma', \mathcal{X}')$ where (h', σ') is an extension (that agrees with U) from (h, σ) to a partial homomorphism from a to an standard atom of H or $\mathcal{V}$; this homomorphism must not send a variable of $\mathcal{F}$ to a variable outside fr and $\mathcal{X}'$ will record any new variable (that is, not in $\mathcal{X}$) sent outside fr.

Algorithm 9: ECDQ-Evaluator$(Q, U, R, H, \mathcal{V}, h, \sigma, \mathcal{X}, \mathcal{F}, \mathcal{EN}, \mathcal{EL})$

Data: An ECDQ $Q = (C, Q_1, \ldots, Q_n)$, a UnionFind structure U, a rule $R = (B, H, fr)$, a vocabulary $\mathcal{V}$, a partial homomorphism h, a specialization σ, $\mathcal{X}$ the set of variables sent by h outside fr, a set of variables that must be sent to fr $\mathcal{F}$, a set of atoms to be evaluated now $\mathcal{EN}$ and a set of atoms to be evaluated later $\mathcal{EL}$.

Result: Has the same function as ECDQ-Unifier, but restricted to evaluating everything necessary from $\mathcal{EL}$.

Remaining work

Implementation of ECDQ-Evaluator

This function will proceed by separating $\mathcal{EL}$ in two sets while $\mathcal{EN}$ is not empty.

Implementation of ECD-rewriting

We have implemented the search of unifiers, but there is still a need to decorate the algorithm to extract more information (mostly which nodes of the tree are visited, and what atoms were evaluated); and to make another algorithm that creates the rewrited ECDQ.

Unification of the two algorithms

The algorithm for Unifiers presents a particuliar tree traversal, provided with a way to evaluate things on a later time. It would be interesting to use this method for the Answers algorithm.

Answer variables

As it is written here, we only focus on boolean queries, but we have begun to consider allowing users to specify answer variables (it is visible in the algorithms). However, doing so would make certain rewritings impossible, so precautions should be taken when implementing ECD-rewriting.

References

[1] John F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, 1984.

[2] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 228–242. IEEE Computer Society, 2010.

[3] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[4] Eric Salvat and Marie-Laure Mugnier. Sound and complete forward and backward chainingd of graph rules. In Peter W. Eklund, Gerard Ellis, and Graham Mann, editors, *Conceptual Structures: Knowledge Representation as Interlingua, 4th International Conference on Conceptual Structures, ICCS '96, Sydney, Australia, August 19-22, 1996, Proceedings*, volume 1115 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 1996.

[5] Mélanie König, Michel Leclère, and Marie-Laure Mugnier. Query rewriting for existential rules with compiled preorder. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3106–3112. AAAI Press, 2015.

Annex

Alternative proof of proposition 8

Proof. We will prove this by induction.

- If Q is an ECQ, then $\texttt{Flat}(Q) = Q$ so the result is trivial.
- If $Q = (R \neq \emptyset, S_1, \ldots, S_n)$, $\texttt{Flat}(Q) = (\emptyset, S'_1, \ldots, S'_n)$.

  We have that for $1 \leq i \leq n$, $P_i = (R, R_i, \ldots end \ldots)$ is a complete path of Q if and only if $P'_i = (\emptyset, R \cup R_i, \ldots end \ldots)$ is a complete path of $\texttt{Flat}(Q)$, and if they are, they both have the same union of the conjuncts at their nodes.

  Thus P_i ECD-maps to F if and only if P'_i ECD-maps to F, and this is true for all $1 \leq i \leq n$. Since a complete path of Q is of the form P_i and a complete path of $\texttt{Flat}(Q)$ is of the form P'_i, we have that Q ECD-maps to F if and only if $\texttt{Flat}(Q)$ ECD-maps to F.
- If $Q = (\emptyset, S_1, \ldots, S_n)$ with $S_1, \ldots, S_n$ for which the property holds, $\texttt{Flat}(Q) = (\emptyset, \texttt{Flat}(S_1), \ldots, \texttt{Flat}(S_n))$.

  We have that for $1 \leq i \leq n$, $P_i = (\emptyset, root(\texttt{Flat}(S_i)), \ldots end \ldots)$ is a complete path of $\texttt{Flat}(Q)$ if and only if $P'_i = (root(\texttt{Flat}(S_i)), \ldots end \ldots)$ is a complete path of $\texttt{Flat}(S_i)$, and if they are, they both have the same union of the conjuncts at their nodes.

  Thus P_i ECD-maps to F if and only if P'_i ECD-maps to F, and this is true for all $1 \leq i \leq n$. Since a complete path of $\texttt{Flat}(Q)$ is of the form P_i and a complete path of $\texttt{Flat}(S_i)$ is of the form P'_i, we have that $\texttt{Flat}(Q)$ ECD-maps to F if and only if there is a i such that $\texttt{Flat}(S_i)$ ECD-maps to F. By induction hypothesis, $\texttt{Flat}(Q)$ ECD-maps to F if and only if there is a i such that S_i ECD-maps to F.

  By using the same relation between complete paths of Q and S_i (which is the presence/absence of $\emptyset$ at the start), we obtain that Q ECD-maps to F if and only if there is a i such that S_i ECD-maps to F.

  With these two results, we have that: $\texttt{Flat}(Q)$ ECD-maps to F if and only if Q ECD-maps to F.

We only need a basic recursion to have the goal result. □