Paper ID: 226

Approximate Query Translation Across Heterogeneous Information Sources

Chen-Chuan K. Chang[1] Héctor García-Molina
Computer Science Department
Stanford University
{kevin, hector}@db.stanford.edu

Paper ID: 226
Category: Research

---

[1]Contact Author: Chen-Chuan K. Chang
Email: kevin@db.stanford.edu
Mail: Gates Hall 431
Computer Science Department
Stanford University
Stanford, CA 94305
USA
Phone: (650) 725-3368

Paper ID: 226

Approximate Query Translation Across Heterogeneous Information Sources

Abstract

In this paper we present a mechanism for approximately translating Boolean query constraints across heterogeneous information sources. Achieving the best translation is challenging because sources support different constraints for formulating queries, and often these constraints cannot be precisely translated. For instance, a query [score $>$ 8] might be "perfectly" translated as [rating $>$ 0.8] at some site, but can only be approximated as [grade = A] at another. Unlike other work, our general framework adopts a customizable "closeness" metric for the translation that combines both precision and recall. Our results show that for query translation we need to handle interdependencies among both query conjuncts as well as disjuncts. As the basis, we identify the essential requirements of a rule system for users to encode the mappings for atomic semantic units. Our algorithm then translates complex queries by rewriting them in terms of the semantic units. We show that, under practical assumptions, our algorithm generates the best approximate translations with respect to the closeness metric of choice. We also present a case study to show how our technique may be applied in practice. *Category: Research*

1 Introduction

To enable interoperability, mediator systems [1, 2] must integrate heterogeneous information sources with different data representations and search capabilities. A mediator presents a unified context for uniform information access, and consequently must translate *original* user queries from the unified context to a *target* source for native execution. This translation problem has become more critical now that the wide range of disparate sources are just "one click away" across the Internet. Achieving the best translation is challenging because sources use different constraints for formulating queries, and often these constraints cannot be precisely translated. This paper presents a framework that finds perfect mappings if possible, or in general the "closest" approximations, taking into account differences in attribute names, operators, and data formats.

Example 1: Consider a mediator that integrates online shopping sites for books, audio, and videos. (This example is based on our case study in Section 6.) In particular, the mediator presents a unified view `Media`(name, format, $\cdots$). Suppose a user wants to find the "VHS" items by some actor "Harrison." Let us consider translating the corresponding constraints v = [format = `vhs`] and n = [name contains `Harrison`].

The mediator will find perfect mappings whenever possible (*e.g.*, it will translate n to [au contains `Harrison`] for source `fatbrain.com`, and it will map v as is for `amazon.com`). However, in many cases such perfect mappings simply do not exist. For instance, for source *EB* at `www.-evenbetter.com`, neither v nor n can be translated precisely.

Consequently, some schemes focus on "minimal-superset" mappings [3], which will return all the potential answers but with as few unwanted answers as possible. In particular, the mediator will map v to [type = movies] (*i.e.*, searching the "movies" category) for *EB*, returning VHS as well as DVD items. Unfortunately, for n the only superset mapping at *EB* is *True* (*i.e.*, returning the entire source database), which is often unacceptable.

However, in many cases, good approximations do exist, and they may be more favorable. For instance, *EB* can approximate n as [star = "Harrison"] to match Harrison as a last name. (Note that *EB* requires at least a last name for star.) It will miss those answers with Harrison as the first name, *e.g.*, Ford, Harrison. However, since most users will actually mean last names (e.g., Harrison, George) in such a name query, this mapping may be better than *True*.

In fact, even v may need a different approximation, say if [type = movies] returns a huge number of DVDs and very few VHS items. Alternatively, mapping [desc contains vhs] simply looks for vhs in the textual descriptions. This mapping may return a lot less data than [type = movies], but may perhaps miss a few VHS items (that do not have the term vhs in their description). If the "false negatives" are acceptable, then the alternative mapping may be more attractive. ■

We can view a query as a Boolean expression of constraints of the *selection* form [attr1 op value] or the *join* form [attr1 op attr2]. (While not discussed here, we stress that our approach can generally handle the join constraints as well; see [4].) These constraints constitute the query "vocabulary," and must be transformed to "native" constraints understood by the target source. For example, constraint [score > 8] may have to be mapped to [grade = A]. In this process, attributes have to be mapped (*e.g.*, score to grade), values have to be converted (*e.g.*, score 8 to grade A), and operators have to be transformed (*e.g.*, ">" to "="). Reference [3] provides more details on how we generally model this constraint-mapping problem in the common mediation architecture [1, 2].

After we first studied query translation in an earlier work [3], and implemented that machinery, we soon realized that *approximate translation* is critical for "real-world" applications. Our earlier work focused on minimal-superset mappings as the "correct" translations, because the *exact* results can be recovered by post-processing their supersets. As just illustrated, in many cases only approximations exist, and they might be even more practical than the strictly correct ones. (Analogously, a concurrent system with strict serializability may result in undesirable low concurrency.) In fact, in our case study of a "real-world" scenario (Section 6), we informally estimated that 70% of the translations must rely on approximation.

Furthermore, different mediation applications need different "correctness" or *closeness* criteria for mappings. It is thus essential for a translation system to flexibly support a wide range of closeness metrics. This paper presents such a framework, where the best approximate translations can be found under virtually *any* reasonable metric. In particular, the framework supports minimal-superset, maximal-subset (when extra-answers must not be returned), and other "hybrid" criteria in between. Our customizable criteria allows one to quantify "false positives" and "false negatives" that are expected to occur in a translation, in an analogous fashion to how the conventional IR parameters of precision and recall quantify "errors" that occur in executing a single query.

Our results show that, under such flexible metrics, one must cope with *interdependencies* among both query conjuncts and disjuncts (Section 4). It is thus critical to note that query mapping is not simply a matter of translating each constraint separately. Some interrelated constraints can form a "semantic unit" that must be handled together. This discovery is surprising since our previous study [3] showed that query disjunctions can be translated separately, significantly simplifying the translation process. Now, in an approximate translation scenario, interrelation depends on the particular closeness metric, as we next illustrate.

Example 2: Let us continue our movie search example. Suppose that the user is looking for both VHS and DVD formats with the query $Q = v \vee d$, where $d =$ [format = dvd]. (Recall that $v =$ [format = vhs].) Let us denote the closest mapping (for some closeness metric) of query Q as $\mathcal{S}(Q)$.

Suppose that the mediator adopts the minimal-superset metric, under which it will generate $\mathcal{S}(v)$ and $\mathcal{S}(d)$ both as [type = movies] (Example 1). In this case, to translate Q, the mediator can separately map the disjuncts v and d, *i.e.*, $\mathcal{S}(Q) = \mathcal{S}(v) \vee \mathcal{S}(d) =$ [type = movies], which indeed precisely translates Q, *i.e.*, $Q \equiv \mathcal{S}(Q)$.

To contrast, assume that the mediator is concerned about large result sizes, so as illustrated earlier, uses the mappings $\mathcal{S}(v) =$ [desc contains vhs] and $\mathcal{S}(d) =$ [desc contains dvd]. (That is, given the mediator's closeness metric, these are the best approximate translations.) Now $\mathcal{S}(v) \vee \mathcal{S}(d) =$ [desc contains vhs] $\vee$ [desc contains dvd]. This mapping is not as good as [type = movies], which in our example exactly gets all VHS and DVD titles. Thus, for the closeness metric in use, translating $\mathcal{S}(v)$ and $\mathcal{S}(d)$ separately leads to a suboptimal mapping, and hence disjunction Q is not "separable." ∎

Query translation must rely on human expertise to define what constraints may be interrelated, and how to translate basic semantic units. For instance, in Example 2 we need a rule for translating the single-constraint pattern [format = F] such as v and d. But do we need a rule for composite queries, *e.g.*, $(v \vee d)$? What kind of queries must constitute such "semantic units"? In this paper we will answer these questions, identifying the essential requirements for a translation rule system.

Based on rules, our challenge is to translate arbitrary queries as Boolean expressions of constraints (we currently do not handle negation). Our approach is to *divide-and-conquer*. We present Algorithm *NFB* to "decompose" an original query into its semantic units, which can then be translated by the given rules. Note that there are many decompositions, but not all of them will lead to the closest mapping. In our running example, suppose that we are given translation rules for the semantic units (n), (v), (d), and $(v \vee d)$, and we wish to translate query $nv \vee nd$. We can decompose the query as $(n)(v) \vee (n)(d)$, or with some rewriting, as $(n)(v \vee d)$. On which expression should we apply the rules to obtain the best mapping? Is the best solution unique? How is the optimality of translation guaranteed? Again, we will answer these questions in this paper.

In summary, we make the following main contributions for approximate query translation:

- We propose a general *framework*, and we define the notion of translation closeness. Our framework can adopt different closeness metrics for different applications.

- We present an *algorithm* for systematically finding the best translation with respect to a given closeness metric. Algorithm *NFB* will find a *unique* best-mapping in the practical cases when semantic units do not "interlock."

- We develop fundamental theorems on the separability of query components, and safeness of decompositions. These results are critical for the development of any algorithm that attempts approximate query translation.
- We study how to estimate the precision and recall parameters of a translation, and we show that reasonable formulas do exist for such estimation.

We briefly discuss related work in Section 2, and then start in Section 3 by defining closeness criteria that combine precision and recall. Section 4 studies a basic assumption on compositional monotonicity and our results on compositional separability. In Section 5 we present our framework and Algorithm *NFB*. Finally, Section 6 concludes with a case study to show how our approach may be applied in practice. Note that, due to space limitations, we leave the details of some important results (that support but are not directly used by our algorithms) to Appendix. Furthermore, interest readers may refer to an extended report [5] for the formal proof of our results.

2 Related Work

Information integration has been an active research area [1, 2, 6, 7]; however, we believe that our focus on the *constraint mapping* problem is unique. Many integration systems have dealt with source capabilities, *e.g.*, Information Manifold [8, 9], TSIMMIS [10, 11], Infomaster [12, 13], Garlic [14, 15], DISCO [16], and others [17, 18, 19]. These efforts have mainly focused on generating query plans that observe the *grammar* restrictions of native queries (such as allowing conjunctions of two constraints, or disallowing disjunctions).

Our work complements these efforts by addressing the semantic mapping of constraints, or analogously the translation of *vocabulary* (of native constraints). In particular, the output of our semantic mapping (which uses the constraint vocabulary understood by the target source) can be the input to the capability mapping that others have analyzed. See reference [3] for additional details of what distinguish our focus on the constraint mapping problem from other integration efforts and how our approach can be applied in the common mediation architecture [1, 2].

There has also been much work on data translation and schema integration. The main focus of these related efforts (*e.g.*, [20, 21, 22, 23, 24, 25]) is to unify data representations across mismatched domains by transforming data to a unified context, where queries can be performed. In contrast, our complementary goal is to map queries to the native domain where data reside. We believe our approach is especially well suited for autonomous sources containing large volumes of data, such as found on the Web (where it is not economical or feasible to transform all data). In addition, note that in our constraint mapping problem we must consider both data conversion and query capability mapping (as Section 1 discusses). Furthermore, we consider translation errors and closeness, which as far as we know are not considered in traditional schema and data translation work.

Surprisingly, while approximation is critical for query mapping (Section 1), we have seen virtually no translation efforts that stress this notion. However, approximation has been studied for query processing: First, some work aims to reduce processing cost through approximation. For instance, references [26, 27] study the approximate fixpoints of Datalog predicates, and [28] uses

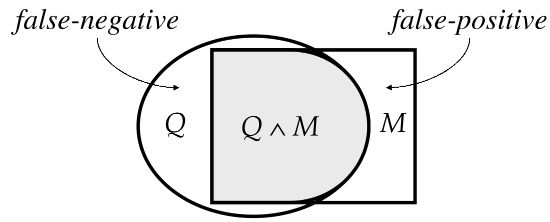


Figure 1: Venn diagram of a query Q and its mapping M.

approximate predicates as filters for expensive ones. Second, several researchers have explored accelerated but approximated query answering to reduce response time [29, 30, 31, 32]. Third, reference [33] develops a framework for representing approximate complex-objects and supporting queries over them. Finally, CoBase [34] explored query relaxation for approximate answering.

We define our translation metrics based on the parameters of precision and recall. Both classic notions have been commonly used for quantifying respectively false-positives and false-negatives, most notably for information retrieval [35, 36]. In addition, some single-valued measures for IR effectiveness have also been proposed, such as the well-known E-measure [35] (see Section 3).

Finally, the approximate translation discussed in this paper was motivated by our previous work [3]. As Section 1 mentioned, our earlier model of "exact" mappings significantly simplifies the translation process, but unfortunately cannot accommodate general closeness metrics. In contrast, this paper specifically explores the notion of *approximation*, and deals with mappings under virtually *any* reasonable closeness metric.

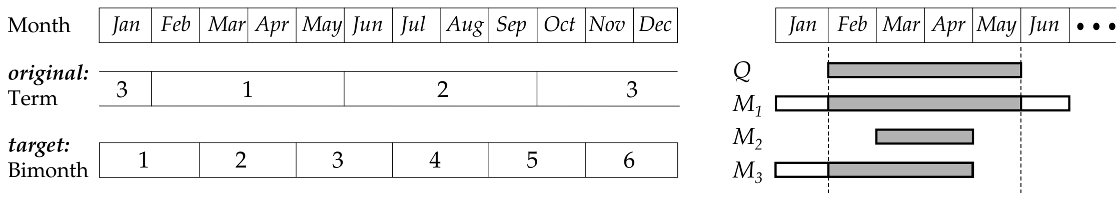
3 Query Approximation: Accounting for Precision and Recall

Our goal for query mapping is to find the closest translation for an original query, which may not be fully expressible at the target. To quantify how closely a mapping M approximates the original query Q, we use a *closeness criterion* $\mathcal{F}[M, Q]$ that returns a normalized "rating" in $[0:1]$ as the *closeness* between M and Q. The higher the rating is, the closer M approximates Q. Our framework allows a wide variety of closeness functions (we will discuss some intuitive and important ones). We say that a mapping M is the *closest mapping* for Q with respect to the closeness criterion $\mathcal{F}$, if for any other mapping M' of Q, $\mathcal{F}[M, Q] \geq \mathcal{F}[M', Q]$ (with ties broken arbitrarily). We denote the closest mapping of Q by $\mathcal{S}(Q)$.

An approximation may erroneously introduce *false-positives* or *false-negatives*, as compared to the original query. Figure 1 illustrates these errors using a Venn diagram for the result sets of a query Q and its mapping M. To quantify (and ultimately minimize) these errors, we define the following metrics. The *precision* measures the proportion of the mapping results that are correct:

$$\mathcal{P}[M, Q] = \frac{|Q \wedge M|}{|M|}. \tag{1}$$

(We denote the size of the result set of query X by $|X|$.) This parameter captures the false-positive component in the approximation error. As Figure 1 suggests, precision will increase as we reduce the number of false-positives.



(a) Correspondence of term and bimonth. (b) Mappings for $Q = [\text{term} = 1]$.

Figure 2: Mapping term constraints to bimonth constraints.

In contrast, *recall* measures the proportion of the correct results that are retrieved by the mapping, *i.e.*,

$$\mathcal{R}[M, Q] = \frac{|Q \wedge M|}{|Q|}. \tag{2}$$

As the dual of precision, recall captures the false-negatives, *i.e.*, higher recall corresponds to fewer false-negatives. Note that both the $\mathcal{P}$ and $\mathcal{R}$ parameters are normalized in $[0 : 1]$.

Example 3 (Precision & Recall): Consider translating between two different calendar systems. As the time unit, the original context uses the term attribute, while the target uses bimonth. Figure 2(a) shows the correspondence. In the original context a year consists of three terms (*i.e.*, trimesters); *e.g.*, constraint $[\text{term} = 1]$ represents *Feb* through *May*. In contrast, the target divides a year into six bimonths; *e.g.*, $[\text{bimonth} = 1]$ matches *Jan* and *Feb*. We illustrate some mappings for query $Q = [\text{term} = 1]$ (see Figure 2(b)).

First, consider M_1:$[\text{bimonth} = 1{:}3]$ (bimonth 1 to 3). Note that Q covers (*Feb*, *Mar*, *Apr*, *May*), while M_1 covers *Jan* and *Jun* in addition. Thus, M_1 incurs no false-negatives, but does have false-positives. By Eq. 2, as $Q \wedge M_1 = Q$, the recall is *perfect*, *i.e.*, $\mathcal{R}[M_1, Q] = 1$. Furthermore, according to Eq. 1, we can estimate $\mathcal{P}[M_1, Q] = 4/6 = .67$ since $M_1 \wedge Q$ covers four out of the six months of M_1 (assuming that each month has equal likelihood).

In contrast, M_2:$[\text{bimonth} = 2]$ is a subset of Q ($M_2 \subseteq Q$). As Figure 2(b) shows, $\mathcal{P}[M_2, Q] = 2/2 = 1$; as a dual of the superset mapping, a subset mapping implies a perfect precision (*i.e.*, no false-positives). The high precision comes at the cost of a lowered recall, *i.e.*, $\mathcal{R}[M_2, Q] = 2/4 = .5$.

Finally, a mapping may have neither perfect precision nor perfect recall. For M_3:$[\text{bimonth} = 1{:}2]$, we can similarly compute $\mathcal{P}[M_3, Q] = 3/4 = .75$ and $\mathcal{R}[M_3, Q] = 3/4 = .75$. Note that M_3 incurs both false-positives (*Jan* is extra) as well as false-negatives (*May* is missed). ■

To quantify translation closeness, a reasonable metric must account for the two *competing* goals of precision and recall. We thus define our closeness criterion $\mathcal{F}[M, Q]$ as a function of the precision and recall between M and Q. For instance, some applications may want to focus on precision while requiring a recall threshold. We denote this important class of closeness functions as *RThresh*. Given a threshold θ, we define *RThresh*(θ) as follows:

$$\textit{RThresh}(\theta): \quad \mathcal{F}(\mathcal{P}, \mathcal{R})[M, Q] = \mathcal{F}(\mathcal{P}[M, Q], \mathcal{R}[M, Q]) = \begin{cases} \mathcal{P}[M, Q] & \text{if } \mathcal{R}[M, Q] \geq \theta \\ \textit{undefined} & \text{otherwise} \end{cases} \tag{3}$$

Example 4: Consider the mappings in Example 3 for $\mathcal{F}=$ *RThresh*$(.7)$. The closeness for M_1 is $\mathcal{F}[M_1, Q] = \mathcal{F}(\mathcal{P} = .67, \mathcal{R} = 1) = .67$. Similarly $\mathcal{F}[M_3, Q] = \mathcal{F}(.75, .75) = .75$. Since M_2 has an unqualified recall ($.5 < .7$), its closeness is *undefined*, *i.e.*, M_2 is as an *invalid* mapping. For our illustration, assume that M_1, M_2 and M_3 represent *all* the relevant mappings for Q. M_3 is thus the best mapping, *i.e.*, $\mathcal{S}(Q) = M_3$, because it has the highest closeness *w.r.t.* $\mathsf{RThresh}(.7)$. ■

We similarly define $\mathsf{PThresh}(\theta)$ as follows:

$$\mathsf{PThresh}(\theta): \quad \mathcal{F}(\mathcal{P},\mathcal{R})[M,Q] = \mathcal{F}(\mathcal{P}[M,Q],\mathcal{R}[M,Q]) = \begin{cases} \mathcal{R}[M,Q] & \text{if } \mathcal{P}[M,Q] \geq \theta \\ \textit{undefined} & \text{otherwise} \end{cases} \tag{4}$$

The $\mathsf{PThresh}$ and $\mathsf{RThresh}$ classes represent many intuitive and important closeness metrics. We stress two special instances typically adopted for query mapping, namely $\mathsf{RThresh}(1)$ and $\mathsf{PThresh}(1)$. First, some applications may require perfect recall and hence $\mathsf{RThresh}(1)$, where M *subsumes* Q, *i.e.*, $M \supseteq Q$. The goal here is to find the most precise mapping (with the highest $\mathcal{P}$) that subsumes the query (with $\mathcal{R} = 1$), usually referred to as the *minimal subsuming mapping* [3] or the *tight upper envelope* [26, 27]. We designate $\mathsf{RThresh}(1)$ as MinSup, since M will retrieve a *minimal superset* of what Q does.

As the dual, other applications may instead require that a mapping return only precise answers, *i.e.*, $M \subseteq Q$. We can implement this closeness criterion as $\mathsf{PThresh}(1)$ with perfect precision. Unlike MinSup, the goal now is to find the *maximal subsumed mapping* or the *tight lower envelope* [26, 27]. We thus refer to $\mathsf{PThresh}(1)$ as MaxSub.

Example 5: Different closeness criteria will determine different mappings as the closest. Example 4 showed that in our calendar application $\mathcal{S}(Q) = M_3$ *w.r.t.* $\mathcal{F}= \mathsf{RThresh}(.7)$. To contrast, consider $\mathcal{F}= \mathsf{MinSup}$: We obtain $\mathcal{F}[M_1,Q] = \mathcal{F}(.67,1) = .67$, $\mathcal{F}[M_2,Q] = \mathcal{F}(1,.5) = \textit{undefined}$, and $\mathcal{F}[M_3,Q] = \mathcal{F}(.75,.75) = \textit{undefined}$. Thus instead $\mathcal{S}(Q) = M_1$ under MinSup. Furthermore, if we adopt MaxSub, both $\mathcal{F}[M_1,Q]$ and $\mathcal{F}[M_3,Q]$ will be *undefined*, and thus $\mathcal{S}(Q) = M_2$. ■

In addition to the above intuitive metrics, many other reasonable criteria are possible. For instance, if we need a function that is defined for every P and R, we can adopt the averages such as the arithmetic average $\mathcal{F}(\mathcal{P},\mathcal{R}) = (\mathcal{P}+\mathcal{R})/2$ (corresponding to the error measure of [27]) or the harmonic mean $\mathcal{F}(\mathcal{P},\mathcal{R}) = 2\mathcal{P}\mathcal{R}/(\mathcal{P}+\mathcal{R})$. The latter actually corresponds to the *E-measure* [35], a conventional single-valued measure for information retrieval.

Finally, we stress that our general framework (Section 5) does not assume particular metrics. However, we do require that the closeness criterion be *monotonic*. That is, if $P_1 \leq P_2$ and $R_1 \leq R_2$ then $\mathcal{F}(P_1,R_1) \leq \mathcal{F}(P_2,R_2)$. In other words, if mapping M_2 (with parameters P_2 and R_2) is better than M_1 (with P_1 and R_1) in both parameters, then M_2 must be an overall better mapping. Because precision and recall capture both false-positive and false-negative errors, clearly any *reasonable* closeness metric (such as the sample functions just discussed) must satisfy monotonicity. (Monotonicity supports our framework through the theorems in Appendix A.)

4 Query Compositions

In this section we address two fundamental questions, whose answers will help us build our approximate query translation machinery. The first question (Section 4.1) is about compositional monotonicity. For instance, if we wish to translate query $Q = A \wedge B$ as $\mathcal{S}(A) \wedge \mathcal{S}(B)$, can we compute $\mathcal{S}(A)$, the best translation for A, independently from that for B? Or will somehow the best translation for A depend on the fact that it will be eventually intersected with $\mathcal{S}(B)$? Furthermore, the second question (Section 4.2) is whether in general it is possible to find the best translation for a query like $Q = A \wedge B$ by separately translating its components A and B.

4.1 Compositional Monotonicity

Consider a query composition $Q = Q_1 \odot \cdots \odot Q_n$, where operator $\odot$ is either $\wedge$ or $\vee$. The following assumption reduces our search space when looking for a best translation.

Assumption 1 (Compositional Monotonicity): For a query composition $Q = Q_1 \odot \cdots \odot Q_n$, let $\mathcal{S}(Q_i)$ be the closest mapping of Q_i with respect to some closeness criterion $\mathcal{F}$. For every mapping $\mathcal{M}_i$ of Q_i:

$$\mathcal{F}[\mathcal{M}_1 \odot \cdots \odot \mathcal{M}_n, Q] \leq \mathcal{F}[\mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n), Q]. \quad \blacksquare$$

This assumption tells us that, if we wish to search for the best way to decompose a query, we can focus on using the "local optimals" as the building blocks, *i.e.*, the mapping $\mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n)$, with the best mappings for each Q_i. In other words, the search for the best translation for each Q_i will not be affected because Q_i appears with other terms in Q. Note, however, that this assumption does *not* tell us if decomposition is the right strategy, *i.e.*, it does not tell us if $\mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n)$ is as good as $\mathcal{S}(Q)$. (We study this "separability" issue in Section 4.2.) It only says that $\mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n)$ is the best of the mappings for Q that use decomposition.

Under certain closeness metrics, such as *MinSup* and *MaxSub*, we can formally verify Assumption 1 (see [5]). We do not have a proof for the general case, but we believe it holds in all cases where we need to use the assumption. That is, when Q_i's are "semantically independent," their individual best-mappings should lead to an overall better mapping, and Assumption 1 should be valid. Otherwise, when Q_i's are indeed interrelated, the closest mapping $\mathcal{S}(Q)$ probably cannot be constructed by separating the components. For such "inseparable" compositions (Section 4.2), our algorithm will not use Assumption 1 and thus it is harmless. Finally, we stress that, even for the rare exceptional cases, $\mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n)$ clearly remains at least a *good* mapping for Q.

4.2 Compositional Separability

When translating a query composition $Q = Q_1 \odot \cdots \odot Q_n$, can we handle the subqueries separately? We say that Q is *separable* if $\mathcal{S}(Q) = \mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n)$, in which case we can obtain the overall mapping simply by translating the components individually. It turns out that separability depends on the particular closeness metric chosen. Our results indicate that disjunctions are *always* separable *for and only for MinSup* (*i.e.*, *RThresh*(1), which requires a perfect recall threshold). As a dual result, conjunctions are *always* separable *for and only for MaxSub* (*i.e.*, *PThresh*(1), which requires a perfect precision threshold). Due to space limitations, please refer to Appendix A for our formal results (Theorem 2 and 3). Here we simply illustrate with an example.

$\mathcal{F}(\mathcal{P},\mathcal{R})$	$\mathcal{S}(t_1)$	$\mathcal{S}(t_2)$	$\mathcal{S}(t_1 \vee t_2)$	$\mathcal{S}(t_1 \wedge t_2)$
MinSup	[bimonth = 1:3] $(\mathcal{P},\mathcal{R}) = (.67, 1)$	[bimonth = 3:5] $(\mathcal{P},\mathcal{R}) = (.67, 1)$	[bimonth = 1:5] $(\mathcal{P},\mathcal{R}) = (.8, 1)$	*False* $(\mathcal{P},\mathcal{R}) = (1, 1)$
MaxSub	[bimonth = 2] $(\mathcal{P},\mathcal{R}) = (1, .5)$	[bimonth = 4] $(\mathcal{P},\mathcal{R}) = (1, .5)$	[bimonth = 2:4] $(\mathcal{P},\mathcal{R}) = (1, .75)$	*False* $(\mathcal{P},\mathcal{R}) = (1, 1)$

Figure 3: Example closest mappings for t_1:[term = 1] and t_2:[term = 2].

Example 6 (Separability): Consider query t_1:[term = 1] and t_2:[term = 2] (for the calendar systems in Example 3). We will compare whether their disjunction $(t_1 \vee t_2)$ and conjunction $(t_1 \wedge t_2)$ are separable or not under *MinSup* and *MaxSub*.

(a) *MinSup*: The *MinSup*-row in Figure 3 shows the closest mappings $\mathcal{S}(t_1)$, $\mathcal{S}(t_2)$, $\mathcal{S}(t_1 \vee t_2)$, and $\mathcal{S}(t_1 \wedge t_2)$ under *MinSup* (*e.g.*, Example 5 shows how we determined $\mathcal{S}(t_1)$). It turns out that for *MinSup* disjunctions are separable, but not conjunctions: We can verify that $\mathcal{S}(t_1 \vee t_2) = \mathcal{S}(t_1) \vee \mathcal{S}(t_2)$ (*i.e.*, [bimonth = 1:5] = [bimonth = 1:3] $\vee$ [bimonth = 3:5]). In contrast, $\mathcal{S}(t_1 \wedge t_2) \neq \mathcal{S}(t_1) \wedge \mathcal{S}(t_2)$, because $\mathcal{S}(t_1 \wedge t_2) = \mathit{False}$, while $\mathcal{S}(t_1) \wedge \mathcal{S}(t_2)$ = [bimonth = 3].

(b) *MaxSub*: We obtain the opposite results: First, the conjunction is separable, since $\mathcal{S}(t_1 \wedge t_2) = \mathcal{S}(t_1) \wedge \mathcal{S}(t_2) = \mathit{False}$ (see Figure 3). Second, the disjunction is not separable: $\mathcal{S}(t_1) \vee \mathcal{S}(t_2)$ = [bimonth = 2] $\vee$ [bimonth = 4], but $\mathcal{S}(t_1 \vee t_2)$ = [bimonth = 2:4]. ■

In general, for any metric other than *MinSup* or *MaxSub*, neither conjunctions nor disjunctions are always separable. Therefore, a general framework for more flexible approximation metrics must cope with the potential inseparability for both types of compositions, as we will discuss next.

5 Approximate Mapping: Framework and Algorithm

This section presents our framework and the associated algorithm for approximate translation. Section 5.1 first defines a translation rule system for codifying the mappings of basic semantic units. Based on the given rules, our algorithm will rewrite an original query using the semantic units to construct the closest mapping. As we just discussed, the rewriting must respect compositional separability to ensure mapping optimality; we present such an algorithm (*NFB*) in Section 5.2.

5.1 Semantic Translation Rules

Query translation must be based on human expertise to resolve semantic heterogeneity. This section identifies the essential requirements of a rule system that codifies human expertise. We will illustrate with a "reference rule system," which is based on our mechanism designed earlier specifically for minimal-superset mapping [3]. We adapt this mechanism (to handle semantic units that can be complex queries) for general approximate translation.

We stress that our contribution is *not* the rule system itself, but its integration with a general query mapping scheme. The "reference" rule system is rather simple (*e.g.*, it has no recursion and negation). However, note that our algorithm can work with any rule mechanism that satisfies our

R_1)	[format = F] $\mapsto$ *emit*: [desc contains F]	// $(\mathcal{P}, \mathcal{R}) = (1.0, 0.8)$
R_2)	[format = F1] $\vee$ [format = F2]; `FormatPair(F1,F2)` $\mapsto$ T = `TypeOfPair(F1,F2)`; *emit*: [type = T]	// $(\mathcal{P}, \mathcal{R}) = (1.0, 1.0)$
R_3)	[term = T] $\mapsto$ (B1,B2) = `TermToBimonth(T)` ; *emit*: [bimonth = B1:B2]	// $(\mathcal{P}, \mathcal{R}) = (.75, .75)$
R_4)	[term = T1] $\vee$ [term = T2] $\mapsto$ (B1,B2) = `TermToBimonth(T1,T2)` ; *emit*: [bimonth = B1:B2]	// $(\mathcal{P}, \mathcal{R}) = (0.8, 1.0)$
R_5)	[fn = F] $\mapsto$ *emit*: [review contains F]	// $(\mathcal{P}, \mathcal{R}) = (0.9, 0.7)$
R_6)	[ln = L] $\mapsto$ A = `LnFnToName(L, "*")`; *emit*: [name = A]	// $(\mathcal{P}, \mathcal{R}) = (1.0, 1.0)$
R_7)	[ln = L] $\wedge$ [fn = F] $\mapsto$ A = `LnFnToName(L, F)`; *emit*: [name = A]	// $(\mathcal{P}, \mathcal{R}) = (1.0, 1.0)$
R_8)	[price in P1:P2] $\mapsto$ *emit*: [price $\geq$ P1] $\wedge$ [price $\leq$ P2]	// $(\mathcal{P}, \mathcal{R}) = (1.0, 1.0)$
R_9)	[subject = S] $\mapsto$ K = `SubjKwds(S)`; *emit*: [review contains K]	// $(\mathcal{P}, \mathcal{R}) = (0.9, 0.7)$
R_{10})	[title = T] $\mapsto$ W = `WordsIn(T)`; *emit*: [title contains W]	// $(\mathcal{P}, \mathcal{R}) = (0.9, 1.0)$

Figure 4: Example mapping specification K_{med} with respect to $\mathcal{F}=$ *RThresh*(.7).

soundness and completeness requirements (see later). For instance, if necessary, our frameowrk can adopt more sophisticated rules that support recursive query patterns (*e.g.*, a conjunction of arbitrary number of conjuncts). Nevertheless, we believe that our simple system is well suited to most query translation tasks, as we will demonstrate through a case study in Section 6.

Figure 4 shows a *mapping specification* K_{med} consisting of rules $R_1, \ldots, R_{10}$ for translating queries that search for media items of books, audios, and videos (based on a real scenario that Section 6 will study). Our discussion assumes $\mathcal{F}=$ *RThresh*(.7). Each rule defines the closest mapping (with respect to $\mathcal{F}$) of the matching query patterns, as Example 7 below illustrates. Figure 4 also shows the estimated $(\mathcal{P}, \mathcal{R})$ for the partitcular mappings. We stress that our algorithm will not require these numeric values to compute the best mappings. However, if we want to quantify the actual closeness of an output mapping, we can estimate it based on the $\mathcal{P}$ and $\mathcal{R}$ of the rules (using the technique in Appendix C).

Example 7 (Mapping Rules): We illustrate rule R_1 and R_2 for translating media format. Suppose that the original context expects formats `hardcover` and `softcover` (for books), `cassette` and `disc` (for audios), and `vhs` and `dvd` (for videos). In contrast, the target accepts media type of `book`, `audio`, and `video`.

First, consider a format constraint such as v = [format = `vhs`]. As an atomic constraint, it needs a rule to define its mapping. To illustrate, we have at least two choices: First, consider M_1 = [type = `video`]. Since M_1 will access both VHS and DVD titles, it has $(\mathcal{P}, \mathcal{R}) = (.5, 1)$ (assuming VHS accounts for 50% of videos). With $\mathcal{F}=$ *RThresh*(.7) (see Eq. 3), M_1 has a closeness of $\mathcal{F}(.5, 1) = .5$. Alternatively, mapping M_2 = [desc contains `vhs`] simply looks for the keyword in desc (a textual description of the media). Suppose that about 80% VHS descriptions mention the word, and on the other hand only VHS items do, M_2 will have $(\mathcal{P}, \mathcal{R}) = (1, .8)$ or $\mathcal{F}(1, .8) = 1$. Thus M_2 is the closest mapping with respect to *RThresh*(.7), *i.e.*, $\mathcal{S}(v) = M_2$ (assuming no other relevant mappings exist). Rule R_1 simply matches any format constraint f (as a *matching*) at the left side and defines $\mathcal{S}(f)$ with respect to $\mathcal{F}$ at the *emit*: clause of the right side.

Furthermore, we notice that a query asking for a pair of formats (such as $v \vee d$, where d =

[format = dvd]) can map perfectly to a particular type (*e.g.*, [type = video]). Since we cannot construct this perfect (and thus the closest) mapping from the components, such a query forms a new semantic unit and thus Rule R_2 defines its translation. At the left side, R_2 will match a disjunctive pattern [format = F1] $\vee$ [format = F2] for those F1 and F2 that satisfy the condition FormatPair($\cdot$) as a pair of formats. For a matching m (*e.g.*, $m = v \vee d$), the right side then finds the corresponding type with function TypeOfPair($\cdot$) and emits $\mathcal{S}(m)$. Note that we assume that conditions and functions are both implemented externally with some programming language. ∎

Our discussion will assume an original query $Q_{med} = t(h \vee c)(v \vee d)$ as a running example. (Referring to Figure 5(a), we are querying the VHS or DVD titles by Tom Hanks or Tom Cruise. Note that we omit the $\wedge$ operator for notational simplicity.) To map a query, we begin by matching it to the rules to find the subqueries for constructing the overall mapping, as we next illustrate.

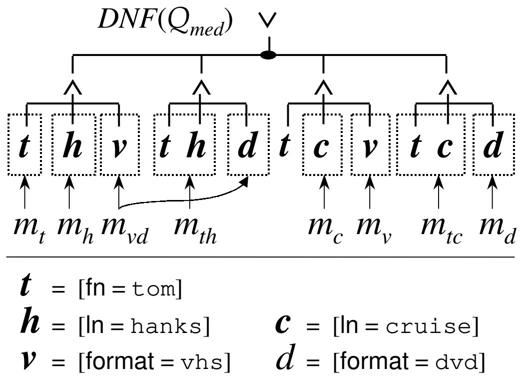
Example 8 (Rule Matching): Consider matching Q_{med} against rules K_{med}; *i.e.*, we want to find the subqueries of Q_{med} that match a pattern described by some rule in K_{med}. Since a matching can be any complex query (with conjunctions, disjunctions, or both), we perform the matching on some normal form, say, DNF (Disjunctive Normal Form). (We could have instead chosen CNF or Conjunctive Normal Form. The choice is not critical, but it does affect how we structure the subsequent algorithm, as Section 5.2 will discuss.)

Specifically, we write Q_{med} in a DNF to compare it with the DNF patterns of the rules. Note that a DNF has the form $\hat{D}_1 \vee \cdots \vee \hat{D}_n$. (Note that we write $\hat{X}$ to stress that it is a conjunction; we will similarly use $\check{X}$ for a disjunction). For Q_{med} we have $\hat{D}_1 = (thv), \ldots, \hat{D}_4 = (tcd)$ (see Figure 5). As our framework also assumes, each rule specifies a DNF pattern of the form $\hat{d}_1 \vee \cdots \vee \hat{d}_m$: *e.g.*, rule R_2 has pattern P_2 with $\hat{d}_1$:[format = F1] and $\hat{d}_2$:[format = F2].

We next determine if the rule pattern matches some subquery of Q_{med}. To see if a pattern P represents a *subquery*, we check if every $\hat{d}_j$ in P is "simpler" than some different $\hat{D}_i$ in the query. Note that, since both $\hat{D}_i$ and $\hat{d}_j$ are a simple conjunction, we say that $\hat{d}_j$ is *simpler* than $\hat{D}_i$ (or $\hat{D}_i$ is *more complex* than $\hat{d}_j$) if $\hat{d}_j$ matches some part of $\hat{D}_i$. For instance, consider pattern $\hat{d}_1 \vee \hat{d}_2$ of R_2: Since $\hat{d}_1$ can match v (with F1 bound to constant vhs), it is simpler than $\hat{D}_1$ (among others). Similarly $\hat{d}_2$ can match d (with F2 = dvd) and is thus simpler than $\hat{D}_2$. Thus R_2 matches subquery $v \vee d$ (or m_{vd} in Figure 5) of Q_{med}, *i.e.*, $v \vee d$ is a matching to R_2.

We repeat this process for every rule to find all the matchings. Figure 5(a) indicates these matchings as subtrees of Q_{med}'s DNF. Figure 5(b) then summarizes each matching m, the rule output for m (denoted by $\overline{m}$), and the estimated $(\mathcal{P}, \mathcal{R})$ (from Figure 4). (As mentioned, our algorithm does not need these parameter values for computing mappings.) ∎

To enable query translation, we assume two essential requirements for semantic rules. *First*, we require that each rule define the closest mappings of the matching queries with respect to $\mathcal{F}(\mathcal{P}, \mathcal{R})$— which we refer to as the *soundness* requirement (*i.e.*, a rule generates sound mappings). To determine such mappings, we can use source statistics (or perform sample queries) to estimate the precision and recall for different mappings (as Example 7 showed) and choose the one with the highest $\mathcal{F}(\mathcal{P}, \mathcal{R})$. In fact, we can also simply make *intuitive* choices; *i.e.*, in practice a closeness function is *not* explicitly required when defining mapping rules, which Section 6 will discuss.



(a) Matchings in query DNF.

rule	matching	rule output	$(\mathcal{P},\mathcal{R})$
R_1	m_v : v	$\overline{m_v}$:[desc contains vhs]	(1., .8)
R_1	m_d : d	$\overline{m_d}$:[desc contains dvd]	(1., .8)
R_2	m_{vd} : $v \vee d$	$\overline{m_{vd}}$:[type = video]	(1., 1.)
R_5	m_t : t	$\overline{m_t}$:[review contains tom]	(.9, .7)
R_6	m_h : h	$\overline{m_h}$:[name = "hanks,*"]	(1., 1.)
R_6	m_c : c	$\overline{m_c}$:[name = "cruise,*"]	(1., 1.)
R_7	m_{th} : th	$\overline{m_{th}}$:[name = "hanks,tom"]	(1., 1.)
R_7	m_{tc} : tc	$\overline{m_{tc}}$:[name = "cruise,tom"]	(1., 1.)

(b) Matchings and their mappings.

Figure 5: Example query $Q_{med} = t(h \vee c)(v \vee d)$ and its matchings with respect to K_{med}.

Second, we require that there be one rule for every semantic unit— which we refer to as the *completeness* requirement, since it enforces necessary rules be supplied. A *semantic unit* (*e.g.*, v and $v \vee d$ in our example) is a query whose closest mapping cannot be constructed from that of its subqueries. Since a semantic unit is "atomic" in query translation, its mapping must be manually defined with a rule (and thus this requirement). Note that any individual constraint (such as v) is clearly a semantic unit; *e.g.*, R_1 and R_3 in K_{med} both describe such single-constraint units.

Moreover, a semantic unit can be a composite query (such as $v \vee d$). Our separability results (Section 4.2) show that query compositions can be inseparable (and thus form a unit) depending on the particular $\mathcal{F}(\mathcal{P},\mathcal{R})$. For instance, since for $\mathcal{F}=$ *RThresh*(.7) disjunctions are not always separable (formally by Theorem 2 in Appendix A), a semantic unit *may* contain disjunctions, *e.g.*, as in R_2 and R_4. (Obviously we only need a rule for interrelated disjuncts; *e.g.*, we do not need one for [ln = hanks] $\vee$ [format = dvd].) Similarly, we may expect a semantic unit with conjunctions (by Theorem 3), *e.g.*, R_7.

In fact, as Section 4.2 discussed, for any closeness metric other than *MinSup* and *MaxSub*, neither disjunctions nor conjunctions are always separable, and thus a semantic unit may be just any complex queries. Although in many cases a unit might be no more complex than simple disjunctions or conjunctions (as in K_{med}), our algorithm can generally handle any complex units.

We stress that our soundness and completeness requirements together enable the analogously *divide-and-conquer* approach. Given an original query Q, if Q can match a rule, then itself is a semantic unit. We simply fire the rule to compute $\mathcal{S}(Q)$. Suppose $\overline{Q}$ denotes the rule output after matching Q, the soundness requirement ensures that $\mathcal{S}(Q) = \overline{Q}$. For instance, since $(v \vee d)$ will match rule R_2, it follows that $\mathcal{S}(v \vee d) =$ [type = video] as given by R_2.

On the other hand, if Q does not match any rule, then by the completeness requirement Q is *not* a semantic unit. In other words, we can construct $\mathcal{S}(Q)$ with the semantic units that are subqueries of Q. For instance, since for Q_{med} we have found the matching subqueries in Figure 5, these semantic units will be the *building blocks* for constructing $\mathcal{S}(Q_{med})$. Such construction of complex mappings thus becomes the main challenge of our framework, which we will discuss next.

5.2 Algorithm *NFB*: Normal-Form Based Algorithm

This section presents the core algorithm of our framework for translating an arbitrary original query. Based on the rule system just discussed, Algorithm *NFB* will construct the mapping of a given query from the semantic units that it contains.

To construct a complex mapping, we are essentially looking for a rewriting using the semantic units. For instance, consider our example query Q_{med}. As we will see, we can construct its mapping from that of the units m_{th}, m_{tc}, and m_{vd} (see Figure 5): *i.e.*, $\mathcal{S}(Q_{med}) = (\overline{m_{th}} \vee \overline{m_{tc}})(\overline{m_{vd}})$. (Recall that $\overline{m}$ denotes the rule output for a matching m.) In other words, we rewrite Q_{med} into a Boolean function of these units: $B_1(m_{th}, m_{tc}, m_{vd}) = (m_{th} \vee m_{tc})(m_{vd})$. (Note that as a rewriting B_1 is logically equivalent to Q_{med}.) We refer to such a rewriting as a *decomposition*, since it breaks the query into the semantic units. Based on decomposition B_1 (but not others), we can simply construct $\mathcal{S}(Q_{med}) = B_1(\overline{m_{th}}, \overline{m_{tc}}, \overline{m_{vd}})$.

There exist *many* decompositions for a given query;*e.g.*, $B_2 = (m_{th} \vee m_{tc})(m_v \vee m_d)$ is another one for Q_{med}. For query mapping we want to find a *safe decomposition*, in which *every* composition (conjunction or disjunction) is guaranteed to be separable. The optimal mapping can then be constructed straightforwardly from such a decomposition: We simply separate every composition, and thus only deal with the semantic units by their rules. To demonstrate, note that B_1 is such a safe decomposition (which can be shown by our results in Appendix B for determining such *safety*). Because $Q_{med} \equiv B_1$, we can obtain $\mathcal{S}(Q_{med})$ or $\mathcal{S}(B_1)$ as $[\mathcal{S}(m_{th}) \vee \mathcal{S}(m_{tc})]\mathcal{S}(m_{vd})$ (by separating every composition since B_1 is safe). Applying the rules for the units (Figure 5), we can construct the mapping form B_1, *i.e.*, $\mathcal{S}(Q_{med}) = (\overline{m_{th}} \vee \overline{m_{tc}})\overline{m_{vd}} = ([\mathsf{name} = \texttt{"hanks,tom"}] \vee [\mathsf{name} = \texttt{"cruise,tom"}]) \wedge [\mathsf{type} = \texttt{video}]$.

Therefore, the main challenge for mapping a query is to find its safe decomposition. Our results (as we will see in Theorem 1) show that, in practical cases, there exists exactly *one* safe decomposition (among many possible rewritings) for a query. Our Algorithm *NFB* (Figure 6) will find such a unique decomposition to construct the closest mapping.

Given a query Q and mapping rules K, *NFB* will output the closest mapping of Q with respect to the closeness metric $\mathcal{F}(\mathcal{P}, \mathcal{R})$ that K is defined upon. Referring to Figure 6, *NFB* first formulates the safe decomposition in Step (1) through Step (4), and finally Step (5) constructs $\mathcal{S}(Q)$ accordingly. We will illustrate by translating Q_{med} using K_{med}, which defines the mappings under $\mathcal{F}$= *RThresh*(.7). Figure 7 summarizes this process, showing the input and output for each step.

Algorithm *NFB* (for *Normal-Form-Based*) is essentially based on Boolean normal forms to systematically rewrite a query into a safe decomposition. In particular, as Figure 7 shows, *NFB* starts by normalizing the input query Q_{med} into a DNF (of the constraints) in Step (1) and finally formulates the safe decomposition B as a CNF (of the semantic units) in Step (4). Note that we could have instead structured a *dual* algorithm that starts with a CNF and concludes at a DNF.

As an overview, we now summarize how *NFB* works, as Figure 7 shows. To begin with, Step (1) normalizes Q_{med} into a DNF, on which Step (2) performs rule matching. Section 5.1 discussed these steps, resulting in the matching units in Figure 5. The algorithm will then systematically rewrite Q_{med} into a safe decomposition B in a CNF (as just mentioned), a simple *two-level* tree with a root conjunction and some leaf disjunctions. To ensure that such CNF rewriting is safe, Step (3)

Algorithm *NFB*: Normal-Form Based Query Mapping

Input: • Q: an arbitrary query in the original context.

• K: the constraint mapping specification for a target system T *w.r.t.* a closeness criterion $\mathcal{F}$.

Output: $\mathcal{S}(Q)$, the closest mapping of Q for T *w.r.t.* $\mathcal{F}$.

Procedure:

(1) DNF Normalization:

• convert Q into DNF: $\mathit{DNF}(Q) \leftarrow \sum_{l=1}^{m} \hat{D}_l$, where $\hat{D}_l$ is a simple conjunction of constraints.

(2) Rule Matching:

• if Q itself matches a rule: $\mathcal{S}(Q) \leftarrow \overline{Q}$; output $\mathcal{S}(Q)$ // *fire the rule for* Q *as a semantic unit.*

• else: find all the matchings $m_1, \ldots, m_u$ of Q *w.r.t.* K; note m_i is in DNF: $m_i = \sum \hat{d}_{ij}$

(3) Safe Conjunction: // *rewrite* $\mathit{DNF}(Q)$ *into a safe conjunction* $Q \equiv \wedge(\mathcal{C})$, *such that*

• $\mathcal{C} = \{\check{C}_1, \ldots, \check{C}_n\} \leftarrow \mathit{SafeConj}(\mathit{DNF}(Q))$ // *every conjunct* $\check{C}_k$ *can then form a safe disjunction.*

(4) Safe Disjunction:

• for all $\check{C}_k \in \mathcal{C}$: rewrite $\check{C}_k = \hat{x}_1 \vee \cdots \vee \hat{x}_m$ into $\check{C}_k \equiv \sum m_i$, for all matchings m_i *s.t.*:

– every $\hat{d}_{ij}$ in m_i appears as some $\hat{x}_l$ in $\check{C}_k$ // *i.e.,* m_i *can be found in* $\check{C}$ *and* $m_i \subseteq \check{C}$.

// *omit* m_i *if* $m_i \subseteq m_{i'}$ *for some* $m_{i'}$, *and thus* m_i *is redundant because of* $m_{i'}$.

– $\nexists\, m_{i'}$ *s.t.* $m_i \subseteq m_{i'}$, *i.e.,* m_i covers only a subset of $\hat{x}_l$ terms that $m_{i'}$ does

• $B(m_1, \ldots, m_u) \leftarrow \prod_{k=1}^{n} \check{C}_k$ // *the safe decompositon in terms of the matchings as units.*

(5) Mapping Construction:

• compute $\overline{m_i}$ for each m_i actually used in B // *fire rules for the relevant matchings.*

• $\mathcal{S}(Q) \leftarrow B(\overline{m_1}, \ldots, \overline{m_u})$; output $\mathcal{S}(Q)$ // *construct the mapping from* B.

Function $\mathit{SafeConj}(\mathit{DNF}(Q) = \sum_{l=1}^{m} \hat{D}_l)$: // *rewrite a query in DNF into a safe conjunction in CNF.*

(a) *Conjunction Rewriting*: // *rewrite each* $\hat{D}_l$ *using the DNF disjuncts of any matching* m_i *found in* Q.

• for all $\hat{D}_l$: rewrite $\hat{D}_l \equiv \prod \hat{d}_{ij}$ for all $\hat{d}_{ij}$ (a disjunct of $\mathit{DNF}(m_i)$; see line 5) *s.t.*:

– $\hat{d}_{ij} \supseteq \hat{D}_l$, *i.e.,* $\hat{d}_{ij}$ is a subconjunction of $\hat{D}_l$ // *as simple conjunctions,* $\hat{d}_{ij}$ *is simpler than* $\hat{D}_l$.

– $\nexists\, \hat{d}_{i'j'}$ *s.t.* $\hat{d}_{ij}$ is a subconjunction of $\hat{d}_{i'j'}$ // *omit* $\hat{d}_{ij}$ *that is simpler than other* $\hat{d}_{i'j'}$.

(b) *CNF Formulation*: // *standard Boolean algebra to convert* Q *into a conjunctive form.*

• rewrite Q in CNF (in terms of the $\hat{d}_{ij}$'s), *i.e.,* $Q \equiv \prod \check{C}$, for all $\check{C}$ *s.t.*

– $\check{C} = \hat{x}_1 \vee \cdots \vee \hat{x}_m$, where every $\hat{x}_l$ denotes some $\hat{d}_{ij}$ from $\hat{D}_l$ as formulated in (a)

// *omit* $\check{C}$ *if* $\check{C}' \subseteq \check{C}$ *for some* $\check{C}'$, *and thus* $\check{C}$ *is redundant in the CNF of* Q.

– $\nexists\, \check{C}' = \hat{x}'_1 \vee \cdots \vee \hat{x}'_m$ *s.t.* $\forall \hat{x}'_j, \exists \hat{x}_i, \hat{x}'_j \subseteq \hat{x}_i$

(c) *Recursive Rewriting*:

• $\mathcal{C} \leftarrow \phi$ // *to store the formulated conjuncts.*

• for all $\check{C} = \hat{x}_1 \vee \cdots \vee \hat{x}_m$ formulated in (b):

– $M \leftarrow \{m_i \mid$ every $\hat{d}_{ij}$ in m_i appears as some $\hat{x}_l$ in $\check{C}\}$ // *all* m_i *found in* $\check{C}$ *s.t.* $m_i \subseteq \check{C}$.

– if every $\hat{x}_l$ in $\check{C}$ appears as $\hat{d}_{ij}$ in some m_i in M: // *i.e.,* $\check{C} \equiv \vee(M)$.

– $\mathcal{C} \leftarrow \mathcal{C} \cup \{\check{C}\}$ // *a safe disjunction of Step (4) can be formed; no need for further rewriting*

– else: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathit{SafeConj}(\check{C})$ // *recursively perform further rewriting of* $\check{C}$.

• return $\mathcal{C}$ // $\mathit{DNF}(Q)$ *is rewritten safely into* $\wedge(\mathcal{C})$.

Figure 6: Algorithm *NFB* for approximate query translation.

focuses on forming a separable conjunction (or a *safe* conjunction) at the root, and similarly Step (4) will form separable disjunctions (or *safe* disjunctions) at the leaves. Specifically, as Example 9 below explains, Step (3) will rewrite Q_{med} into a conjunctive form (see Figure 7) that is separable, *i.e.,* $\mathcal{S}(Q_{med}) = \mathcal{S}(th \vee tc)\mathcal{S}(v \vee d)$. Step (4) then further rewrites each resulted conjunct into a separable disjunction of the semantic unit: *i.e.,* the first conjunct as $(m_{th} \vee m_{tc})$ and the second as (m_{vd}), which Example 10 will illustrate. We have thus formulated the safe decomposition in a CNF: $B = (m_{th} \vee m_{tc})(m_{vd})$. Finally, Algorithm *NFB* simply constructs $\mathcal{S}(Q_{med})$ from B as just discussed.

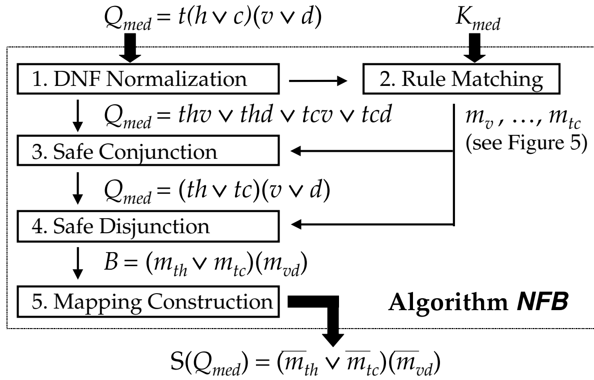


Figure 7: Illustration of Algorithm *NFB* for query Q_{med} with respect to rules K_{med}.

Example 9 (Safe Conjunction): We explain Step (3) of Algorithm *NFB*, which uses function *SafeConj* to rewrite an input query into a conjunction that is guaranteed to be separable. As a basis, to determine whether a conjunction is separable, we have developed the sufficient conditions (called the *safety* conditions) that imply the separability. Due to space limitations, we will leave to Appendix B the safety formalism. We simply stress here that the conjunction that Step (3) formulates will satisfy our safety conditions (Theorem 5 in Appendix B) and thus must be separable.

Intuitively, to eventually form a safe decomposition of Q_{med} using the semantic units (see Figure 5), we first form a safe decomposition for every conjunction in the former using that in the latter. Note that, since both Q_{med} and the units are written in DNF, all their conjunctions are explicit at the leaves (of the query tree); *e.g.*, Q_{med} has (thv), (thd), (tcv), and (tcd) as Figure 7 shows. In particular, we can rewrite (thv) as $(t)(h)(th)(v)$ with the four "subconjunctions" from units m_t, m_h, m_{th}, and m_{vd}. We can then omit (t) and (h), since they are subexpressions of (th) and are thus redundant (*i.e.*, they will not contribute to the next step). Consequently, we have rewritten $Q_{med} = (th)(v) \vee (th)(d) \vee (tc)(v) \vee (tc)(d)$ or $\vee\{(th)(v), (th)(d), (tc)(v), (tc)(d)\}$.

Our goal here is to formulate a conjunction (at the root of the query tree). Since the above rewriting is disjunctive, Step (b) of *SafeConj* simply distributes the outer disjunction over the inner conjunctions (using the standard Boolean algebra). Omitting any redundancies, we will obtain a conjunctive form $Q_{med} = \check{C}_1\check{C}_2$ with two conjuncts $\check{C}_1 = (th \vee tc)$ and $\check{C}_2 = (v \vee d)$. (Example 17 in Appendix B shows that the conjunction is safe and thus separable.)

Finally, Step (c) of *SafeConj* determines if every such conjunct is ready for Step (4) of Algorithm *NFB*, or else it needs further rewriting. In other words, we want to test if $\check{C}_i$ can be written as the sum of the contained units (which is essentially the safe disjunction that Step (4) will formulate). In particular, referring to Figure 5, $\check{C}_2$ contains units m_v, m_d and m_{vd}. Since we can indeed write $\check{C}_2$ as their sum (*i.e.*, $\check{C}_2 \equiv m_v \vee m_d \vee m_{vd}$), it does not need further rewriting, and similarly neither does $\check{C}_1$. ∎

In general, while not shown in the above example, any resulted conjunct that cannot be written as a sum-of-units will be further "decomposed." In other words, such $\check{C}_i$ will be rewritten, by recursively calling *SafeConj*, into simpler conjuncts. To illustrate, consider a query $wx \vee yz$, and suppose that the matching units are $m_1 = wx \vee y$, $m_2 = w$, $m_3 = x$, $m_4 = y$, and $m_5 = z$. Given $wx \vee yz$ as input, *SafeConj* will first rewrite it into $(wx \vee y)(wx \vee z)$. Conjunct $(wx \vee y)$ can be written as sum-of-units simply as m_1, but the latter conjunct cannot. Consequently, a recursive call *SafeConj*$(wx \vee z)$ will further rewrite the latter into $(w \vee z)(x \vee z)$, where the new conjuncts can be written as $(m_2 \vee m_5)$ and $(m_3 \vee m_5)$ respectively.

This recursive process will eventually terminate and produce a separable conjunction: Intuitively, every recursion will derive strictly simpler subqueries as just illustrated. Eventually, *SafeConj* will terminate with the simplest form (if not earlier), *i.e.*, a disjunction of atomic constraints (*e.g.*, $w \vee z$), which is trivially a sum of single-constraint units. Please refer to reference [5] for a formal proof that the conjunctions so formulated are separable.

After forming the safe conjunction in Step (3) as just shown, Algorithm *NFB* will then focus on the disjunctions in Step (4), as the following example illustrates.

Example 10 (Safe Disjunction): Continuing our example, Step (4) will next rewrite each conjunct $\check{C}_i$ into a safe disjunction. (Like our discussion for conjunctions, the resulted disjunctions will satisfy the "safety" conditions of Theorem 4 in Appendix B and thus must be separable.) As just illustrated, Step (3) ensures that every $\check{C}_i$ can be written as the sum of the contained units, *e.g.*, $\check{C}_2 \equiv (m_v \vee m_d \vee m_{vd})$. Removing the redundant terms (*i.e.*, $m_v \subseteq m_{vd}$ and $m_d \subseteq m_{vd}$), we obtain $\check{C}_2 \equiv m_{vd}$. Similarly, we can rewrite $\check{C}_1 \equiv (m_{th} \vee m_{tc})$. (Example 14 in Appendix B shows that the disjunction is indeed safe).

It turns out that, when semantic units are not "interlocking," the disjunctions so formulated will be safe and thus separable. As we will see, in the rare interlocking cases, a safe decomposition may not exist and thus Step (4) may not form safe disjunctions. For the majority of cases, as in this example, the resulted disjunctions (and thus the decomposition overall) are safe. ∎

By constructing a safe decomposition, Algorithm *NFB* will generate the best translation, as we have illustrated. Essentially, based on the optimal mappings for semantic units (as given by sound rules), and by respecting constraint dependencies (as the units indicate) to preserve optimality through query rewriting, *NFB* guarantees the overall optimal mappings. Our results below show that, in the vast majority of cases, namely when no semantic units "interlock" (defined below), a query will have a *unique* safe decomposition. Consequently, Algorithm *NFB* will find this unique decomposition and thus construct the closest mapping. Please refer to reference [5] for a proof.

Theorem 1 (Unique Safe Decomposition): Given a query Q and a mapping specification K *w.r.t.* some closeness criterion $\mathcal{F}$, if Q has no interlocking semantic units by matching K, then

- there exists a *unique* safe decomposition of Q, from which $\mathcal{S}(Q)$ *w.r.t.* $\mathcal{F}$ can be constructed, and
- Algorithm *NFB* will find the safe decomposition and output $\mathcal{S}(Q)$.

Otherwise, when there are interlocking units, a safe decomposition for Q may not exist. ∎

On the other hand, Theorem 1 also states that, when a query involves interlocking units, it may *not* have a safe decomposition. Note that Q still have a best mapping, but $\mathcal{S}(Q)$ must instead be found among the unsafe decompositions. (Our completeness requirement in Section 5.1 asserts that $\mathcal{S}(Q)$ can be constructed from *some* decomposition using its semantic units.) We formally define interlocking below, and then illustrate with Example 11.

Definition 1 (Interlocking Units): A set of semantic units $\mathcal{U}$ is *interlocking*, if for some $m \in \mathcal{U}$, there exist $m_1, \ldots, m_n$ also in $\mathcal{U}$ such that the following hold:

(1) Let $\mathit{DNF}(m) = \sum \hat{d}_k$ and $\mathit{DNF}(m_i) = \sum \hat{d}_{ij}$. (a) Every m_i has some $\hat{d}_{ij}$ that overlaps with but is not strictly simpler than some $\hat{d}_k$; at least one such $\hat{d}_{ij}$ is strictly more complex than the corresponding $\hat{d}_k$. (b) m is simpler than $(m_1 \vee \ldots \vee m_n)$.

(2) Let $\mathit{CNF}(m) = \prod \check{c}_k$ and $\mathit{CNF}(m_i) = \prod \check{c}_{ij}$. For some $\check{c}_k$, $\check{c}_k \subseteq \sum_{i=1}^{n} \check{c}_{ij_i}$ but $\check{c}_k \not\subseteq \check{c}_{ij_i}, \forall i$. ■

Example 11 (Interlocking Units): Consider query $Q = xy \vee z$. Suppose that (by matching rules) Q has semantic units (written in DNF): $m_{x\vee z} = (x) \vee (z)$, $m_{xy} = (xy)$, $m_x = (x)$, $m_y = (y)$, and $m_z = (z)$. Note that these units are interlocking: Intuitively, an interlock exists because constraint x is involved in both conjunction m_{xy} and disjunction $m_{x\vee z}$.

More formally, we show the interlocking by Definition 1. Let $m = m_{x\vee z}$, $m_1 = m_{xy}$, and $m_2 = m_z$. First, Condition (1) will hold: Term (xy) of m_1 satisfies (a) with respect to term (x) of m, and similarly term (z) of m_2 with respect to term (z) of m. In addition, term (xy) of m_1 is also strictly more complex than (x) of m. Because m is indeed a subexpression of $m_1 \vee m_2$, (b) is also satisfied. Second, since $\mathit{CNF}(m) = (x \vee z)$, $\mathit{CNF}(m_1) = (x)(y)$, and $\mathit{CNF}(m_2) = (z)$, Condition (2) also holds because $(x \vee z) \subseteq (x) \vee (z)$ while $(x \vee z) \not\subseteq (x)$ and $(x \vee z) \not\subseteq (z)$.

Consequently, Q may not have a safe decomposition (by Theorem 1) because of the interlocking. Intuitively, to rewrite Q using the semantic units, we must separate *either* the disjunction (between xy and z) *or* the conjunction (between x and y). (Otherwise, Q remains monolithic.) However, *neither* will be safe— The former, such as in $B_1 = m_{xy} \vee m_z$, will break the dependency between x and z (as indicated by $m_{x\vee z}$) and thus is not safe (*i.e.*, it will fail the safety conditions of Theorem 4 in Appendix B). Similarly, the latter will break m_{xy}, such as in $B_2 = m_{x\vee z}(m_y \vee m_z)$, which is also unsafe (by Theorem 5 in Appendix B). ■

We believe that such interlocking cases will be rare in practice. As we intuitively noted above, interlock occurs between such "overlapping" units as m_{xy} and $m_{x\vee z}$. That is, interlocking will happen only when a constraint (*e.g.*, x in our example) participates in *both* a conjunction unit (*e.g.*, m_{xy}) *and* a disjunction unit (*e.g.*, $m_{x\vee z}$). If x appears in only simple-disjunction units, like $(x \vee y)$ and $(x \vee z)$, no interlocking will form. (Similarly, no simple-conjunction units can interlock.) Because a semantic unit represents interdependencies among its constraints, such complex interlocking is very unlikely in practice. In fact, in our case study of real mapping systems (see Section 6), we have indeed observed no instances of such an "anomaly."

When interlocking does occur, because no safe decomposition exists, Algorithm *NFB* will not be able to construct $\mathcal{S}(Q)$. We can address these exceptional cases in two ways: First, we may simply require these interlocking queries (*e.g.*, $xy \vee z$) be defined by rules. Alternatively, we can find all the unsafe decompositions, estimate the closeness of each corresponding mapping, and select the best as $\mathcal{S}(Q)$. Note that it is possible to estimate the $\mathcal{P}$ and $\mathcal{R}$ parameters and thus the closeness of a constructed mapping; we show such estimation technique in Appendix C due to space limitations.

Finally, we conclude by analyzing the running time of Algorithm *NFB*. First, in Step (1) and Step (3) (*i.e.*, subroutine *SafeConj*), *NFB* will perform DNF and CNF conversion respectively. Such a conversion is in general exponential in the number of query constraints (because the Boolean satisfiability problem is NP-complete [37]). However, this conversion has been well studied and practical algorithms have been proposed in the literature [38]. Therefore, for queries of practical sizes, we believe this normalization can be reasonably efficient.

Furthermore, the other steps of Algorithm *NFB* are quite efficient and actually run in linear time of the input size: Consider a query Q and rules K as input (note that they are all in DNF after Step (1)). Let D_Q and N_Q be the number of disjuncts and constraints-per-disjunct in the DNF of Q; similarly, let D_R and N_R be those of the DNF query pattern in a rule. Let R be the number of rules in K. With these input size parameters, Step (2) will take time $O(N_Q N_R D_Q D_R R)$: That is, the algorithm will match each pair of constraints (thus the factor $N_Q N_R$), for each pair of query and rule disjuncts (thus the factor $D_Q D_R$), and for each rule (thus the factor R). Step (4) will then run in $O(C_Q M)$ time, where C_Q is the number of CNF conjuncts of Q (thus an upper bound of what Step (3) can generate) and M is the number of matchings found in Step (2). Finally, Step (5) will simply take time of $O(M)$.

6 Practical Implications: A Case Study

To verify that our framework makes sense in practice, we explore several information sources on the Web. We wish to study how to "program" our general framework for a specific mapping system. In other words, we will demonstrate the mapping rules for a representative scenario. Through this concrete example, we also want to understand practical issues such as the ease of composing rules, the number of rules typically required, and whether approximation is essential in practice.

6.1 A Book-Search Mediator

Let us consider building a book-search mediator that integrates online sources *Amazon* at `www.amazon.com` and *BN* at `www.barnsandnoble.com` (both are online bookstores), *EB* at `www.evenbetter.com` (a comparison shopping service), and *Socrates* at `socrates.stanford.edu` (Stanford library online catalog, currently not publicly accessible). Our scenario assumes that the mediator integrates these sources by adopting *Amazon*'s query context and thus needs translation for the other sources.

We will thus demonstrate constraint mappings from *Amazon* to respectively *BN*, *EB*, and *Socrates*. For each target source, we compare its constraint vocabulary[1] (*i.e.*, supported constraints) with that of *Amazon* and define the mapping rules. As Figure 8 shows, we need eight rules for *BN*, six rules for *Socrates*, and seven rules for *EB*. As Section 5.1 discusses, each rule gives the best mapping for the matching semantic unit with respect to the closeness metric, which we assume to be $\mathcal{F} = \mathit{PThresh}(.7)$. In fact, in practice it is *not* required to explicitly consider the closeness function, as we will discuss in Section 6.2.

For instance, rules $B_1, \ldots, B_8$ map the constraints on attributes title, ln, fn, subject, and format in the *Amazon*[2] context to ones on title, keyword, author, subject, and format in the *BN* context. Note that when defining mapping rules, we only need to focus on the corresponding *clusters* of attributes

[1] For these sources, the query interfaces (that we use in our case study) and their documentations are available through the URL links at `http://www-db.stanford.edu/~kevin/aqt`.

[2] Since *Amazon* distinguishes the first and last names in author attribute, we separate it into ln and fn in translation.

Target Source: *BN* (BarnesAndNoble.com) at `www.barnesandnoble.com`

B_1) [title O T] $\mapsto$ W = `WordsIn`(T); *emit*: [title contains W]
B_2) [fn = F] $\mapsto$ *emit*: [keyword contains F]
B_3) [ln = L] $\mapsto$ A = `LnFnToName`(L, `null`); *emit*: [author = A]
B_4) [ln = L] $\wedge$ [fn = F] $\mapsto$ A = `LnFnToName`(L, F); *emit*: [author = A]
B_5) [subject O S1]; `EqualsOrStarts`(O) $\mapsto$ S2 = `MapSubjHeading`(S1); *emit*: [subject = S2]
B_6) [subject contains W] $\mapsto$ S = `SubjKwdToSubjHeading`(W); *emit*: [subject = S]
B_7) [format = F1] $\mapsto$ F2 = `MapFormat`(F1); *emit*: [format = F2]

Target Source: *Socrates* at `socrates.stanford.edu`

S_1) [title O T] $\mapsto$ [title O T]
S_2) [A = N]; `LnOrFn`(A) $\mapsto$ *emit*: [au contains N]
S_3) [subject O S1]; `EqualsOrStarts`(O) $\mapsto$ S2 = `MapSubjHeading`(S1); *emit*: [subject = S2]
S_4) [subject contains W1] $\mapsto$ W2 = `MapSubjKwd`(W1); *emit*: [subject contains W2]
S_5) [format = F] $\mapsto$ [keyword contains F]
S_6) [format = `hardcover`] $\vee$ [format = `paperback`] $\mapsto$ *True*

Target Source: *EB* (EvenBetter.com) at `www.evenbetter.com`

E_1) [title O T] $\mapsto$ W = `WordsIn`(T); *emit*: [title contains W]
E_2) [fn = F] $\mapsto$ *emit*: [keyword contains F]
E_3) [ln = L] $\mapsto$ A = `LnFnToName`(L, `null`); *emit*: [author = A]
E_4) [ln = L] $\wedge$ [fn = F] $\mapsto$ A = `LnFnToName`(L, F); *emit*: [author = A]
E_5) [subject O S] $\mapsto$ W = `WordsIn`(S); *emit*: [keyword contains W]
E_6) [format = F] $\mapsto$ *emit*: *False*
E_7) [format = `hardcover`] $\vee$ [format = `paperback`] $\mapsto$ *True*

Figure 8: Rules for mapping *Amazon* to sources *BN*, *Socrates*, and *EB*.

in either contexts. For instance, cluster {ln, fn} at *Amazon* corresponds to {author, keyword} at *BN*, whose mappings are given by B_2, B_3, and B_4. In addition, note that *True* (a trivial superset) and *False* (a trivial subset) are both possible mappings (when no better ones exist, as in rules S_6, E_6, and E_7), which will effectively remove the matching units from the translated query.

6.2 Observations

Our case study shows that the query-mapping framework that we have presented can be easily applied in practice. The number of mapping rules are small: Observe that constraint dependencies exist only within a cluster of attributes (*e.g.*, {ln, fn} as in B_4 and {format} as in S_6) and are typically simple. Thus we will only need a few more rules (that describe compositional units) than the number of atomic constraint patterns in the original context. In addition, note that we only need a rule for a query *pattern* (*e.g.*, [subject O S1] of B_5) when its *instantiations* (*e.g.*, [subject = `"web design"`] and [subject starts `"web"`]) share the same way of mapping, which we found to be true in our study. Furthermore, it is often possible to reuse mapping rules for different sources as they share some common constraints; *e.g.*, E_2, E_3, and E_4 are reused from B_2, B_3, and B_4.

Furthermore, we indeed observed no instances of interlocking units (as Section 5.2 discussed). Note that semantic units essentially indicate the correspondence of attributes, such as the *con-*

junction of ln and fn versus author (or similarly month and year versus date) and the *disjunction* of format versus type. We have observed no attributes involved in *both* types of correspondence, without which interlocking simply cannot occur (see Section 5.2). Our algorithm will thus generate the unique best mappings in the practical cases.

While our framework is formally supported by the notion of closeness, a closeness function is *not* explicitly required when defining a mapping rule. In other words, given a semantic unit, we can *intuitively* choose its best mapping when there are competing choices (without explicitly computing their closeness values with a metric as in Example 7). Such mappings are thus defined with some *implicit* metric that corresponds to the "intuition" we may have in mind. Nevertheless, we stress that, as Theorem 1 states, our algorithms will preserve the optimality with respect to any closeness metrics that the mapping rules conform to– be it explicitly or implicitly.

Furthermore, our case study shows that the general notion of approximation (as this paper specifically introduces) is truly essential for a "usable" query-mapping framework. Observe that, among the twenty rules in Figure 8, only six (B_3, B_4, B_7, S_1, E_3, E_4) are perfect mappings, a ratio of 30% – the other 70% is not possible without approximation. Moreover, we need a general framework that can deal with *all* types of approximation, *i.e.*, supersets (*e.g.*, B_1, S_2), subsets (*e.g.*, E_6), and hybrid mappings that contain both false-positives and false-negatives (*e.g.*, B_2, B_5). The approximate translation technique presented in this paper can thus be very helpful in practice. We have studied additional scenarios to the one presented here, and in all cases we have found our observations to hold.

7 Conclusion

In this paper we have presented a framework and the associated algorithm for approximate query translation. Our framework is robust under virtually any reasonable closeness metric that combines both precision and recall. We also intuitively presented our results on the separability and safety of query compositions (and we cover the full details in Appendix). These results are critical for the development of any algorithm that attempts approximate query translation. Our Algorithm *NFB* will generate a unique best-mapping in the practical cases when semantic units do not "interlock."

While our algorithm generates the closest mappings, it does not compute the actual "closeness," in terms of the precision and recall parameters. (In fact, our algorithm does not explicitly use the parameter values to compute the best mappings.) While not essential for the operation of our algorithm, in some cases it may be desirable to estimate the $\mathcal{P}$ and $\mathcal{R}$ (or the corresponding closeness) of a mapping. As a complement to our translation machinery, we have developed simple formulas for such estimation, and it is covered in Appendix C.

We have implemented a running prototype for query translation in the Stanford Digital Libraries Project. Our prototype was based on an earlier work [3] that focused on minimal-superset mappings. From the experience we realized that approximate translation is critical for real-world applications, which motivated the work described in this paper. We are in the process of extending the prototype with the algorithm discussed in this paper.

References

[1] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):51–60, Mar. 1992.

[2] J. D. Ullman. Information integration using logical views. In *Proceedings of the 6th International Conference on Database Theory*, Delphi, Greece, Jan. 1997. Springer, Berlin.

[3] C.-C. K. Chang and H. García-Molina. Mind your vocabulary: Query mapping across heterogeneous information sources. In *Proceedings of the 1999 ACM SIGMOD Conference*, pages 335–346, Philadelphia, Pa., June 1999. ACM Press, New York.

[4] C.-C. K. Chang and H. García-Molina. Mind your vocabulary: Query mapping across heterogeneous information sources (extended version). Technical Report SIDL-WP-1998-0095, Stanford Univ., 1999. Accessible at `http://www-diglib.stanford.edu`.

[5] C.-C. K. Chang and H. García-Molina. Approximate query translation across heterogeneous information sources (extended version). Technical Report SIDL-WP-1999-0115, Stanford Univ., 1999. Accessible at `http://www-db.stanford.edu/~kevin/aqt`.

[6] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 51–61, Tucson, Ariz., 1997. ACM Press, New York.

[7] M. A. Hearst. Trends & controversies: Information integration. *IEEE Intelligent System*, 13(5):12–24, Sept. 1998.

[8] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd VLDB Conference*, pages 251–262, Bombay, India, 1996. VLDB Endowment, Saratoga, Calif.

[9] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query-answering algorithms for information agents. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI-96*, Portland, Oreg., Aug. 1996. AAAI Press, Menlo Park, Calif.

[10] Y. Papakonstantinou, H. García-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, La., 1996.

[11] Y. Papakonstantinou, H. García-Molina, A. Gupta, and J. Ullman. A query translation scheme for rapid implementation of wrappers. In *Proceedings of the 4th International Conference on Deductive and Object-Oriented Databases*, pages 161–186, Singapore, Dec. 1995. Springer, Berlin.

[12] O. M. Duschka. *Query Planning and Optimization in Information Integration*. PhD thesis, Stanford Univ., Dec. 1997.

[13] M. R. Genesereth, A. M. Keller, and O. M. Duschka. Infomaster: An information integration system. In *Proceedings of the 1997 ACM SIGMOD Conference*, Tucson, Ariz., 1997. ACM Press, New York.

[14] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proceedings of the 23rd VLDB Conference*, pages 276–285, Athens, Greece, Aug. 1997. VLDB Endowment, Saratoga, Calif.

[15] M. T. Roth and P. M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proceedings of the 23rd VLDB Conference*, pages 266–275, Athens, Greece, Aug. 1997. VLDB Endowment, Saratoga, Calif.

[16] O. Kapitskaia, A. Tomasic, and P. Valduriez. Dealing with discrepancies in wrapper functionality. Technical Report RR-3138, INRIA, 1997.

[17] H. García-Molina, W. Labio, and R. Yerneni. Capability sensitive query processing on internet sources. In *Proceedings of the 15th International Conference on Data Engineering*, Sydney, Australia, Mar. 1999. Accessible at `http://www-db.stanford.edu/`.

[18] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 105–112, San Jose, Calif., May 1995.

[19] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external query processors. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 27–37, Montreal, Canada, June 1996.

[20] L. G. DeMichiel. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. *IEEE Transactions on Knowledge and Data Engineering*, 1(4):485–493, 1989.

[21] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous information systems. *Transactions on Database Systems*, 19(2):254–290, June 1994.

[22] P. Buneman, S. Davidson, K. Hart, C. Overton, and L. Wong. A data transformation system for biological data sources. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, 1995. VLDB Endowment, Saratoga, Calif.

[23] S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. In *Proceedings of the 6th International Conference on Database Theory*, Delphi, Greece, 1997. Springer, Berlin.

[24] S. Cluet, C. Delobel, J. Simon, and K. Smaga. Your mediators need data conversion! In *Proceedings of the 1998 ACM SIGMOD Conference*, Seattle, Wash., 1998. ACM Press, New York.

[25] C.-C. K. Chang and H. García-Molina. Conjunctive constraint mapping for data translation. In *Proceedings of the Third ACM International Conference on Digital Libraries*, pages 49–58, Pittsburgh, Pa., June 1998. ACM Press, New York.

[26] S. Chaudhuri. Finding nonrecursive envelopes for datalog predicates. In *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 135–146, Washington, D.C., 1993. ACM Press, New York.

[27] S. Chaudhuri and P. G. Kolaitis. Can datalog be approximated? In *Proceedings of the 13rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 86–96, Minneapolis, Minn., 1994. ACM Press, New York.

[28] N. Shivakumar, H. Garcia-Molina, and C. Chekuri. Filtering with approximate predicates. In *Proceedings of the 24th VLDB Conference*, pages 263–274, New York City, USA, 1998. VLDB Endowment, Saratoga, Calif.

[29] K.-L. Tan, C. H. Goh, and B. C. Ooi. On getting some answers quickly, and perhaps more later. In *Proceedings of the 15th International Conference on Data Engineering*, pages 32–39, Sydney, Austrialia, 1999. ACM Press, New York.

[30] V. Poosala and V. Ganti. Fast approximate query answering using precomputed statistics. In *Proceedings of the 15th International Conference on Data Engineering*, page 252, Sydney, Austrialia, 1999. ACM Press, New York.

[31] M. J. Carey and D. Kossmann. On saying "enough already!" in sql. In *Proceedings of the 1997 ACM SIGMOD Conference*, pages 219–230, Tucson, Arizona, 1997. ACM Press, New York.

[32] M. J. Carey and D. Kossmann. Reducing the braking distance of an sql query engine. In *Proceedings of the 24th VLDB Conference*, pages 158–169, New York City, USA, 1998. VLDB Endowment, Saratoga, Calif.

[33] P. Buneman, S. B. Davidson, and A. Watters. A semantics for complex objects and approximate answers. *Journal of Computer and System Sciences*, 43(1):170–218, Aug. 1991.

[34] W. W. Chu, M. A. Merzbacher, and L. Berkovich. The design and implementation of cobase. In *Proceedings of the 1993 ACM SIGMOD Conference*, pages 517–522, Washington, D.C., 1993. ACM Press, New York.

[35] C. J. V. Rijsbergen. *Information Retrieval, 2nd edition.* Butterworths, London, 1979.

[36] G. Salton. *Automatic Text Processing.* Addison-Wesley, Reading, Mass., 1989.

[37] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, Reading, Mass., 1974.

[38] E. J. McCluskey. *Logic Design Principles.* Prentice Hall, Englewood Cliffs, N.J., 1986.

Appendix

A Compositional Separability

A major complication for query mapping is that compositions cannot be arbitrarily separated to obtain their best mappings. Formally, for $Q = Q_1 \odot \cdots \odot Q_n$ (as a conjunction or disjunction), Q is *separable* if $\mathcal{S}(Q) = \mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n)$. This section presents our results of how this separability depends on the closeness criteria. For instance, as shown in Example 6, *MinSup* and *MaxSub* have opposite results of separability.

To see why compositions may be inseparable, we first focus on disjunctions. Note that disjunctions can potentially enhance the recall of mappings: A disjunction such as $(t_1 \vee t_2)$ represents a broader condition than disjunct t_1 and t_2 alone. When we consider the disjuncts together, the broader condition can sometimes (depending on the actual semantics) lead to a broader mapping (than the individual disjuncts combined). Such a broader mapping can potentially reduce the false-negatives and thus enhance the recall (while maintaining the precision) to result in an overall closer mapping. (Note that in Section 3 we have explained that metric $\mathcal{F}$ is monotonic in $\mathcal{P}$ and $\mathcal{R}$.) In Example 6, $\mathcal{S}(t_1 \vee t_2)$ under *MaxSub* results in [bimonth = 2:4], where the additional bimonth 3 indeed reduces false-negatives without introducing false positives.

Consequently, our results below state that disjunctions are always separable, *when and only when* all the valid mappings under $\mathcal{F}(\mathcal{P}, \mathcal{R})$ already have perfect recall (*i.e.*, $\mathcal{R} = 1$). In particular, note that a translation system under *MinSup* (Example 6) explicitly requires $\mathcal{R} = 1$ for all mappings. Disjunctions are thus always separable, since any mappings (separated or not) already have perfect recall (that cannot be enhanced). To contrast, because *MaxSub* allows $\mathcal{R} < 1$, disjunctions can further enhance $\mathcal{R}$ and thus may not be separable (Example 6). Theorem 2 formally states this result. Please refer to reference [5] for a proof. (Note that below we write $\check{Q}$ to stress that it is a disjunction; we will similarly use $\hat{Q}$ for a conjunction).

Theorem 2 (Disjunction Separability): With respect to a closeness criterion $\mathcal{F}(\mathcal{P}, \mathcal{R})$, disjunctions are always separable, *i.e.*, $\forall \check{Q} = D_1 \vee \cdots \vee D_n$, $\mathcal{S}(\check{Q}) = \mathcal{S}(D_1) \vee \cdots \vee \mathcal{S}(D_n)$, if and only if any closest mapping with respect to $\mathcal{F}$ has perfect recall, *i.e.*, $\forall Q : \ \mathcal{R}[\mathcal{S}(Q), Q] = 1$. ∎

Similarly, the dual of the above results can apply to conjunctions, as Theorem 3 states. That is, conjunctions are always separable when and only when $\mathcal{F}(\mathcal{P}, \mathcal{R})$ enforces perfect precision. Example 6 also verifies this result, since conjunctions under *MaxSub* or *PThresh*(1) are separable but not under *MinSup*.

Theorem 3 (Conjunction Separability): With respect to a closeness criterion $\mathcal{F}(\mathcal{P}, \mathcal{R})$, conjunctions are always separable, *i.e.*, $\forall \hat{Q} = C_1 \wedge \cdots \wedge C_n$, $\mathcal{S}(\hat{Q}) = \mathcal{S}(C_1) \wedge \cdots \wedge \mathcal{S}(C_n)$, if and only if any closest mapping with respect to $\mathcal{F}$ has perfect precision *i.e.*, $\forall Q : \ \mathcal{P}[\mathcal{S}(Q), Q] = 1$. ∎

Specifically, we can verify that $\mathcal{F}=$ *MinSup* and $\mathcal{F}=$ *MaxSub*, by their definitions as *RThresh*(1) and *PThresh*(1), satisfy Theorem 2 and 3 respectively. For these important special cases, the mapping problem is "simpler" because we can eliminate the complication of disjunctions (for *MinSup*) or conjunctions (for *MaxSub*).

$\mathcal{F}(\mathcal{P},\mathcal{R})$	$\mathcal{S}(t_1)$	$\mathcal{S}(t_2)$	$\mathcal{S}(t_1 \vee t_2)$	$\mathcal{S}(t_1 \wedge t_2)$
RThresh(.7)	[bimonth = 1:2] $(\mathcal{P},\mathcal{R}) = (.75, .75)$	[bimonth = 4:5] $(\mathcal{P},\mathcal{R}) = (.75, .75)$	[bimonth = 1:5] $(\mathcal{P},\mathcal{R}) = (.8, 1)$	*False* $(\mathcal{P},\mathcal{R}) = (1, 1)$

Figure 9: Example closest mappings for t_1:[term = 1] and t_2:[term = 2].

On the other hand, Theorem 2 and 3 show that a general closeness criteria can imply inseparability for both types of compositions. In fact, *MinSup* and *MaxSub* are probably the only reasonable metrics that will satisfy the theorems. In particular, although *MinSup* and *MaxSub* are simply special cases of *RThresh* and *PThresh*, we stress that in the latter neither disjunctions nor conjunctions are generally separable. Example 12 contrasts *RThresh*(0.7) with *RThresh*(1) or *MinSup*.

Example 12: Continue Example 6 with $\mathcal{F}=$ *RThresh*(.7). By Theorem 2, disjunctions may not be separable because under *RThresh*(.7) a mapping can have $\mathcal{R} < 1$. To illustrate, referring to Figure 9, $(t_1 \vee t_2)$ is indeed inseparable: $\mathcal{S}(t_1 \vee t_2) =$ [bimonth = 1:5] cannot be obtained by $\mathcal{S}(t_1) \vee \mathcal{S}(t_2)$. As compared to $\mathcal{S}(t_1 \vee t_2)$ with $(\mathcal{P},\mathcal{R}) = (.8, 1)$, the latter will miss bimonth 3 and can only achieve $(\mathcal{P},\mathcal{R}) = (.75,.75)$. ■

Therefore, a general framework must cope with the potential inseparability for both disjunctions and conjunctions. To begin with, a semantic unit can be any complex query (with conjunctions and disjunctions). Furthermore, the mapping algorithm must consider both types of compositions carefully. This paper mainly studies such a general mapping framework.

In fact, even with *MinSup* or *MaxSub*, both compositions may be inseparable due to practical limitations at the target system. (This complication further reinforces the need for the general algorithm that we will provide.) In particular, since we focus on semantic (but not syntactic) translation, we have implicitly assumed that the target supports queries of arbitrary sizes. If this is not the case, disjunctions can be inseparable even for *MinSup* and similarly conjunctions for *MaxSub*.

Example 13: Consider mapping conjunction a:[ln = "smith"] $\wedge$ b:[fn = "john"]. Suppose we can query the target with a full name by specifying at least the last name (*e.g.*, "smith,∗"). The query thus has a perfect mapping $\mathcal{S}(a \wedge b) =$ [name = "smith,john"].

Suppose that we adopt the *MaxSub* closeness criterion. By Theorem 3, conjunctions are separable, in theory. First, $\mathcal{S}(a) =$ [name = "smith,∗"]. Second, to translate b, by enumerating all alphabetic strings (*e.g.*, "smith", "xyz", *etc.*), we will not miss any last names. That is, conceptually we have an equivalent (but unbounded) translation as $\mathcal{S}(b) =$ [name = "smith,john"] $\vee$ [name = "xyz,john"] $\vee \cdots$. Consequently, $a \wedge b$ is separable because $\mathcal{S}(a) \wedge \mathcal{S}(b)$

$$= \text{[name = "smith,*"][name = "smith,john"]} \vee \text{[name = "smith,*"][name = "xyz,john"]} \vee \cdots$$
$$= \text{[name = "smith,john"]} \vee \mathit{False} \vee \cdots = \text{[name = "smith,john"]} = \mathcal{S}(a \wedge b).$$

In practice the unbounded mappings may not be supported. In this case, we can formulate the maximal-subset mapping $\mathcal{S}(b)$ by enumerating some most probable names. However, any finite but incomplete enumeration may miss some potential names. When $\mathcal{S}(b)$ indeed misses "smith,john", $(a \wedge b)$ is not separable because $\mathcal{S}(a) \wedge \mathcal{S}(b) = \mathit{False}$. ■

B Safety for Separating Compositions

This section develops the safety conditions that guarantee a *particular* query disjunction or conjunction can be separated. That is, our safety conditions are the *sufficient* conditions that imply separability of a composition. As Theorem 2 and 3 show, query compositions may not generally be separable. This section discusses how to *mechanically* determine whether a composition can be separated or not, based on the associated semantic units. Example 14 starts by showing that we can separate a disjunction if all the possible decompositions already do so.

Example 14 (Disjunction Separation): Consider $\check{Q}_a = D_1{:}th \vee D_2{:}tc$ (a subquery of Q_{med}). As we will see with Theorem 4, it is safe to separate D_1 and D_2, *i.e.*, $\mathcal{S}(\check{Q}_a) = \mathcal{S}(D_1) \vee \mathcal{S}(D_2)$. Note that since $\check{Q}_a$ does not match a rule in K_{med}, we can construct $\mathcal{S}(\check{Q}_a)$ from some decomposition using its subquery matchings (Section 5.1). In particular, $\check{Q}_a$ has matchings m_t, m_h, m_c, m_{th}, and m_{tc} (see Figure 5).

Intuitively, $\check{Q}_a$ is separable because any *potential* decomposition in terms of these semantic units must implicitly separate D_1 and D_2. Let us verify with some alternative decompositions, *e.g.*, $B_1 = m_{th} \vee m_{tc}$. Note that B_1 is simply a rewriting (*i.e.*, $B_1 \equiv \check{Q}_a$) using the semantic units. Observe that B_1 implicitly separates D_1 and D_2— that is, we can write B_1 as $B_{11} \vee B_{12}$, such that $B_{11} \equiv D_1$ and $B_{12} \equiv D_2$. In this case, simply let $B_{11} = m_{th}$ and $B_{12} = m_{tc}$ and the separation is clear. As another example, consider $B_2 = m_t(m_h \vee m_c)$. We can write B_2 as $B_{21}{:}m_t m_h \vee B_{22}{:}m_t m_c$. Since $B_{21} \equiv D_1$ and $B_{22} \equiv D_2$, B_2 also implicitly separates D_1 and D_2. ■

Example 15 (Disjunction Separation): To contrast Example 14, consider $\check{Q}_b = D_1{:}tv \vee D_2{:}td$ (another subquery of Q_{med}). As we will see, Theorem 4 will determine that it is not safe to separate $\check{Q}_b$ (and in fact $\mathcal{S}(\check{Q}_b) \neq \mathcal{S}(D_1) \vee \mathcal{S}(D_2)$). Intuitively, some decomposition of $\check{Q}_b$ does *not* separate D_1 and D_2. To illustrate, note that $\check{Q}_b$ has matchings m_v, m_d, m_{vd}, and m_t (see Figure 5) with respect to K_{med}.

Not all decompositions built on these units will separate D_1 and D_2, *e.g.*, $B_1 = m_t m_{vd}$. In fact, as a pure conjunction, B_1 cannot be written as some $B_{11} \vee B_{12}$ that corresponds to D_1 and D_2. In other words, if we first separate D_1 and D_2, we cannot subsequently derive the decomposition B_1.

The separation is therefore *unsafe*, because the missing decomposition *may* lead to $\mathcal{S}(\check{Q}_b)$. In fact, B_1 is indeed such a decomposition; the corresponding mapping $M_1 = \overline{m_t}\,\overline{m_{vd}}$ (see Figure 5) is the closest to $\check{Q}_b$. To verify, we show that M_1 is actually a perfect mapping, and thus by definition $\mathcal{S}(\check{Q}_b) = M_1$: Note that we can write $\check{Q}_b = t(v \vee d)$. As given by rule R_5, $\overline{m_t}$ perfectly translates t with $(\mathcal{P}, \mathcal{R}) = (1, 1)$, *i.e.*, $t \equiv \overline{m_t}$. Similarly, $(v \vee d) \equiv \overline{m_{vd}}$ according to rule R_2. Thus $t(v \vee d) \equiv \overline{m_t}\,\overline{m_{vd}}$ or $\check{Q}_b \equiv M_1$. In other words, M_1 based on B_1 is the best mapping (with perfect precision and recall). ■

Observe that $\check{Q}_a$ contrasts with $\check{Q}_b$ in that the disjuncts in the former form *no* new semantic units. The matchings of $\check{Q}_a$ can either be found in D_1 (*i.e.*, m_t, m_h, m_{th}) or D_2 (*i.e.*, m_t, m_c, m_{tc}). Since no semantic units span across the disjuncts, every decomposition (with these well-separated units) must implicitly separate D_1 and D_2. In contrast, matching m_{vd} of $\check{Q}_b$ can be found in neither disjuncts. Consequently, a decomposition with this "cross-unit" (*e.g.*, $B_1 = m_t m_{vd}$) will merge the disjuncts. In other words, by separating D_1 and D_2, we cannot use m_{vd} to construct the safe decomposition B_1. In fact, missing m_{vd} is critical because it perfectly maps the factor $(v \vee d)$ (Example 15).

Therefore, to determine if a disjunction can be safely separated, we must check the effect of the disjunction against each semantic unit. Theorem 4 formally states the safety conditions for disjunctions, which Example 16 then illustrates. We give the proof in reference [5].

Theorem 4 (Disjunction Safety): Let disjunction $\check{Q} = D_1 \vee \cdots \vee D_n$. Suppose $\mathit{CNF}(D_i) = \check{C}_{i1} \wedge \cdots \wedge \check{C}_{im_i}$. $\check{Q}$ is separable *w.r.t.* rules K if, for every $\check{C} = \check{C}_{1k_1} \vee \cdots \vee \check{C}_{nk_n}$, either of the following holds:

1. for every matching m of $\check{Q}$ *w.r.t.* K, if $m \subseteq \check{C}$ then $m \subseteq \check{C}_{ik_i}$ for some i; or
2. $\exists \check{C}' = \check{C}_{1h_1} \vee \cdots \vee \check{C}_{nh_n}$, such that $\forall \check{C}_{jh_j}, \exists \check{C}_{ik_i}, \check{C}_{jh_j} \subseteq \check{C}_{ik_i}$ and $\check{C}'$ satisfies (1). ■

Example 16: We illustrate Theorem 4 by showing that $\check{Q}_b$ (in Example 15) is not safe (*i.e.*, it will fail the conditions). We start by writing $\mathit{CNF}(D_1) = (t)(v)$ and $\mathit{CNF}(D_2) = (t)(d)$. Therefore, we need to test the four combinations $\check{C}_1 = (t) \vee (t)$, $\check{C}_2 = (t) \vee (d)$, $\check{C}_3 = (v) \vee (t)$, and $\check{C}_4 = (v) \vee (d)$. In particular, $\check{C}_4$ will fail Condition (1): we find $m_{vd} \subseteq \check{C}_4$ but neither $m_{vd} \subseteq (v)$ nor $m_{vd} \subseteq (d)$. Since $\check{C}_4$ does not satisfy Condition (2) either, we conclude that $\check{Q}_b$ is unsafe and thus may not be separable. ■

Theorem 5 formally states the dual safety condition for conjunctions. We then intuitively illustrate the theorem in Example 17.

Theorem 5 (Conjunction Safety): Let conjunction $\hat{Q} = C_1 \wedge \cdots \wedge C_n$. Suppose $\mathit{DNF}(C_i) = \hat{D}_{i1} \vee \cdots \vee \hat{D}_{im_i}$. $\hat{Q}$ is separable *w.r.t.* rules K if, for every $\hat{D} = \hat{D}_{1k_1} \wedge \cdots \wedge \hat{D}_{nk_n}$, either of the following holds:

1. for every matching m of $\hat{Q}$ *w.r.t.* K, if $m \supseteq \hat{D}$ then $m \supseteq \hat{D}_{ik_i}$ for some i; or
2. $\exists \hat{D}' = \hat{D}_{1h_1} \wedge \cdots \wedge \hat{D}_{nh_n}$, such that $\forall \hat{D}_{jh_j}, \exists \hat{D}_{ik_i}, \hat{D}_{jh_j} \supseteq \hat{D}_{ik_i}$ and $\hat{D}'$ satisfies (1). ■

Example 17 (Conjunction Safety): To illustrate Theorem 5, we show that $\hat{Q} = C_1{:}(th \vee tc) \wedge C_2{:}(v \vee d)$ is safe. Since $\hat{Q}$ is simply a rewriting of Q_{med} (Figure 5), they share the same matchings. As Theorem 5 requires, C_1 and C_2 are already in DNF. We then find the *conjunction factors* of $\hat{Q}_a$ by combining that from C_1 and C_2, *i.e.*, $(th)(v)$, $(th)(d)$, $(tc)(v)$, $(tc)(d)$. Intuitively, $\hat{Q}$ is safe because for every matching, the contribution of each conjunction factor is from either C_1 or C_2 part. For instance, consider matching m_{th}, we find that $m_{th} \supseteq (th)(v)$ and also $m_{th} \supseteq (th)$. Consequently, Theorem 5 will determine that $\hat{Q}$ is safe. ■

Apparently, Theorem 4 and 5 seem to disregard the closeness criteria. However, what semantic units can match the composition actually depends on the mapping rules, which in turn are defined specifically with regard to some particular $\mathcal{F}(\mathcal{P}, \mathcal{R})$ (as Section 5.1 discussed). For instance, we have shown that $\check{Q}_b$ is unsafe because of the "cross-unit" m_{vd}, which matches R_2 in K_{med}. This unsafety actually depends on our $\mathcal{F}= \mathit{RThresh}(.7)$, which K_{med} is based upon. In other words, for a different metric such as $\mathcal{F}' = \mathit{MinSup}$, we may not have a rule that defines m_{vd} as a semantic unit, in which case $\check{Q}_b$ would instead be safe under $\mathcal{F}'$. This is not surprising: the safety for a composition essentially means that the closest mappings under the particular $\mathcal{F}(\mathcal{P},\mathcal{R})$ can be constructed regardless of the separation.

C Estimating Precision and Recall

While our algorithm generates best translations, it does not compute how "good" they actually are, in terms of precision and recall. In fact, given a query Q, Algorithm *NFB* finds the unique best-mapping (Theorem 1) by observing safety, and in particular does not explicitly use the precision and recall values of the semantic units. However, in some cases it may be desirable to estimate the $\mathcal{P}$ and $\mathcal{R}$ (or the corresponding closeness) of a mapping. For instance, a mediator can use such estimation to further compute the query cost before submitting a translated query. In addition, the estimation can also help to determine $\mathcal{S}(Q)$ among unsafe decompositions in the interlocking case (Section 5.2).

This section presents simple formulas for estimating $\mathcal{P}$ and $\mathcal{R}$. As we will see, the formulas are not accurate, but we believe that they are reasonable for approximately measuring the closeness of mappings. Our goal in presenting the approximation formulas is simply to show that such "reasonable" formulas exist. It is beyond the scope of this paper to experimentally show how well these formulas work in practice. Again, these formulas are a complement to our translation machinery, but are not essential. We also note that our estimations are analogous to what query optimizers have done to estimate query result sizes.

Since a mapping M of query Q is constructed from its semantic units, we estimate the $\mathcal{P}$, $\mathcal{R}$ parameters of the former based on those of the latter. Essentially, we must estimate how $\mathcal{P}$ and $\mathcal{R}$ evolve through Boolean compositions (that combine basic units into complex queries). In particular, let M_a and M_b be respectively mappings for queries A and B. Suppose that M_a approximates A with precision $\mathcal{P}_a$ and recall $\mathcal{R}_a$, and similarly M_b with $\mathcal{P}_b$ and $\mathcal{R}_b$. Consequently, what is the precision and the recall when we consider $(M_a \wedge M_b)$ as a mapping for the composite query $(A \wedge B)$? In other words, we want to find $(\mathcal{P}_{a\wedge b}, \mathcal{R}_{a\wedge b})$ in terms of $(\mathcal{P}_a, \mathcal{R}_a)$ and $(\mathcal{P}_b, \mathcal{R}_b)$. (We will consider disjunctions later.)

To begin with, we can compute the recall with Eq. 2 as

$$\mathcal{R}_{a\wedge b} = \frac{|(A \wedge B) \wedge (M_a \wedge M_b)|}{|(A \wedge B)|} = \frac{|(A \wedge M_a) \wedge (B \wedge M_b)|}{|(A \wedge B)|} \tag{5}$$

We then must estimate the sizes of the queries in Eq. 5. Note here we only need their selectivities (or size relative to the database), which we denote by $\sigma(Q)$ for a query Q. As a conjunction, we can write $\sigma(A \wedge B)$ as $\sigma(A) \cdot \sigma(B) \cdot C_{ab}$, with some factor C_{ab} that models the "correlation" between A and B. Intuitively, if A and B are logically independent, this correlation factor is simply $C_{ab} = 1$. Otherwise, we will see $C_{ab} > 1$ for "positive" correlation and $C_{ab} < 1$ for "negative" correlation.

Our estimation is based on assuming *unbiased mappings*, in which the correlation between A and B is preserved through translation. In particular, we assume that mapping M_a and M_b have the same correlation as A and B do. In other words, we can estimate $\sigma(M_a \wedge M_b) \approx \sigma(M_a) \cdot \sigma(M_b) \cdot C_{ab}$ (which Eq. 7 below will need). Consequently, the correlation between $(A \wedge M_a)$ and $(B \wedge M_b)$ is also (approximately) the same, and thus we obtain $\sigma((A \wedge M_a) \wedge (B \wedge M_b)) \approx \sigma(A \wedge M_a) \cdot \sigma(B \wedge M_b) \cdot C_{ab}$. Substituting these relative sizes, we can estimate the recall. Similarly, Eq. 7 shows the $\mathcal{P}_{a\wedge b}$-estimate.

$$\mathcal{R}_{a\wedge b} \approx \frac{\sigma(A \wedge M_a) \cdot \sigma(B \wedge M_b) \cdot C_{ab}}{\sigma(A) \cdot \sigma(B) \cdot C_{ab}} = \frac{\sigma(A \wedge M_a)}{\sigma(A)} \cdot \frac{\sigma(B \wedge M_b)}{\sigma(B)} = \mathcal{R}_a \cdot \mathcal{R}_b \tag{6}$$

$$\mathcal{P}_{a\wedge b} \approx \frac{\sigma(A \wedge M_a) \cdot \sigma(B \wedge M_b) \cdot C_{ab}}{\sigma(M_a) \cdot \sigma(M_b) \cdot C_{ab}} = \frac{\sigma(A \wedge M_a)}{\sigma(M_a)} \cdot \frac{\sigma(B \wedge M_b)}{\sigma(M_b)} = \mathcal{P}_a \cdot \mathcal{P}_b \tag{7}$$

We next consider the estimation for disjunctions. That is, given $(M_a \vee M_b)$ as a mapping for $(A \vee B)$, we want to find $(\mathcal{P}_{a\vee b}, \mathcal{R}_{a\vee b})$. By Eq. 2, we compute recall as

$$\mathcal{R}_{a\vee b} = \frac{|(A \vee B) \wedge (M_a \vee M_b)|}{|(A \vee B)|} \tag{8}$$

It seems less clear how we can estimate the relative sizes or selectivities of disjunctions. (In fact, we have not seen similar analysis for disjunction selectivity in the literature). Our estimation is essentially a *first-order approximation*, which will only count the most dominating terms. To begin with, we can write $\sigma(A \vee B) = \sigma(A) + \sigma(B) - \sigma(A \wedge B)$. We will approximate with only the first-order terms $\sigma(A)$ and $\sigma(B)$, since in comparison $\sigma(A \wedge B)$ is a "magnitude" (of selectivity) smaller; thus $\sigma(A \vee B) \approx \sigma(A) + \sigma(B)$. Furthermore, we can compute the numerator of Eq. 8 as

$$\begin{aligned} \sigma((A \vee B) \wedge (M_a \vee M_b)) &= \sigma((A \wedge M_a) \vee (A \wedge M_b) \vee (B \wedge M_a) \vee (B \wedge M_b)) \\ &= \sigma(A \wedge M_a) + \sigma(A \wedge M_b) + \sigma(B \wedge M_a) + \sigma(B \wedge M_b) + \cdots \end{aligned} \tag{9}$$

Observe that the first-order terms in Eq. 9 are only $\sigma(A \wedge M_a)$ and $\sigma(B \wedge M_b)$. Intuitively, since a query and its mapping must be highly correlated (relatively to arbitrary independent queries), $\sigma(A \wedge M_a)$ is of the same "magnitude" as $\sigma(A \wedge A)$ or $\sigma(A)$, and similarly $\sigma(B \wedge M_b)$ as $\sigma(B)$. In comparison, the other terms are (at least) a magnitude smaller; *e.g.*, $\sigma(A \wedge M_b)$ is of the same magnitude as $\sigma(A \wedge B)$. Furthermore, note that by Eq. 2 we can write $\sigma(A \wedge M_a) = \sigma(A) \cdot \mathcal{R}_a$ and $\sigma(B \wedge M_b) = \sigma(B) \cdot \mathcal{R}_b$. Therefore, the first-order approximation of Eq. 9 becomes $\sigma((A \vee B) \wedge (M_a \vee M_b)) \approx \sigma(A) \cdot \mathcal{R}_a + \sigma(B) \cdot \mathcal{R}_b$. Finally, substituting these size estimates into Eq. 8, we obtain that $\mathcal{R}_{a\vee b}$ is simply the weighted average of $\mathcal{R}_a$ and $\mathcal{R}_b$. Similarly, we can derive the $\mathcal{P}_{a\vee b}$-estimate, as Eq. 11 shows.

$$\mathcal{R}_{a\vee b} \approx \frac{\sigma(A) \cdot \mathcal{R}_a + \sigma(B) \cdot \mathcal{R}_b}{\sigma(A) + \sigma(B)} = wavg[\sigma(A),\ \sigma(B)](\mathcal{R}_a, \mathcal{R}_b) \tag{10}$$

$$\mathcal{P}_{a\vee b} \approx wavg[\sigma(A) \cdot \frac{\mathcal{R}_a}{\mathcal{P}_a},\ \sigma(B) \cdot \frac{\mathcal{R}_b}{\mathcal{P}_b}](\mathcal{P}_a, \mathcal{P}_b) \tag{11}$$