The VLDB Journal manuscript No.
(will be inserted by the editor)

Approximate Query Mapping: Accounting for Translation Closeness

Kevin Chen-Chuan Chang[1], **Héctor García-Molina**[2] ⋆

[1] Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA; e-mail: `kcchang@cs.uiuc.edu`
[2] Computer Science Department, Stanford University, Stanford, CA 94305, USA; e-mail: `hector@db.stanford.edu`

Received: date / Revised version: date

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.

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 may 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. Specifically, the mediator presents `Media`(name, format, $\cdots$) as a unified view for user querying (*i.e.*, it supports a query interface with the above constraining attributes). Suppose a user wants to find the "VHS" items by some actor "Harrison." Let us consider translating the corresponding constraints $v =$ [format = `vhs`] and $h =$ [name contains `Harrison`].

The mediator will find perfect mappings whenever possible (*e.g.*, it will translate h to [au contains `Harrison`] for source `fatbrain.com`, and it will leave v as is for `amazon.com`). However, in many cases such perfect mappings simply do not exist, because of the limited query "capabilities" of target sources. For instance, for source *EB* at `www.evenbetter.com`, neither v nor h can be translated precisely. In particular, *EB* does not support v (*i.e.*, selecting only VHS but not DVD) although it can differentiate movies from other media types (such as books).

When perfect mappings do not exist, some schemes focus on finding "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 h 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 h 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`]

⋆ *Present address:* Insert the address here if needed 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, 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 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 "="). Section 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 [3] in our earlier work, 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 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 (as Section 7 will discuss), 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 any reasonable metric that is "monotonic" (as Section 4 will explain). 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 allow 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 5). 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 $Q = v$:[format = vhs] $\vee$ d:[format = dvd]. Let us denote the closest mapping (for some closeness metric) of query Q as $\mathcal{S}(Q)$.

First, suppose 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, *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 next 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 v and 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 then 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 (h), (v), (d), and $(v \vee d)$, and we wish to translate query $hv \vee hd$. (Note that we omit the $\wedge$ operator for notational simplicity.) We can decompose the query as $(h)(v) \vee (h)(d)$, or with some rewriting, as $(h)(v \vee d)$. 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 a safe decomposition exists.

– 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 Sect. 2, and then start in Sect. 3 by introducing the constraint mapping problem. Sect. 4 then defines closeness criteria that combine precision and recall. Section 5 studies a basic assumption on compositional monotonicity and our results on compositional separability. In Section 6 we present our framework and Algorithm *NFB*. Section 7 discusses a case study to show how our approach may be applied in practice. Section 8 concludes with simple formulas for estimating precision and recall of translated query compositions. Interested readers may also refer to Appendix for the safeness formalism and proof of our results.

2 Related Work

Information integration has been an active research area [1, 2, 5, 6]; however, we believe that our focus on the *constraint mapping* problem is unique. Many integration systems have dealt with source (syntactic) capabilities, *e.g.,* Information Manifold [7,8], TSIMMIS [9,10], Infomaster [11,12], Garlic [13,14], DISCO [15], and others [16–18]. These efforts have mainly focused on generating query plans that observe the native *grammar* restrictions (*e.g.,* 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 only the constraint vocabulary understood by the target source) can be the input to the capability mapping that others have analyzed. Section 3 discusses our particular focus and how our approach can be generally applied in a common mediation architecture.

There has also been much work on data translation and schema integration. The main focus of these related efforts (such as [19–24]) 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 discussed). Furthermore, we consider translation errors and closeness, which as far as we know are not considered in traditional schema and data translation work.

Surprisingly, although 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 [25,26] study the approximate fixpoints of Datalog predicates, and [27] uses approximate predicates as filters for expensive ones. Second, several researchers have explored accelerated but approximated query answering [28–31] to reduce response time. Third, reference [32] develops a framework for representing approximate complex-objects and supporting queries over them. Finally, CoBase [33] explored query relaxation for approximate answering.

We define our translation metrics based on the precision and recall parameters. Both classic notions have been commonly used for quantifying respectively false-positives and false-negatives, most notably for information retrieval [34, 35]. In addition, single-valued measures for IR effectiveness have also been proposed, such as the well-known E-measure [34] (see Section 4).

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 it cannot accommodate general closeness metrics. In contrast, this paper specifically explores the notion of *approximation*, and deals with mappings under any reasonable closeness metrics (that are monotonic).

3 The Constraint Mapping Problem

We describe the *constraint mapping* problem in a common mediation architecture [1,2] for integrating autonomous and heterogeneous *sources*. In such systems, *wrappers* unify the source data models, and *mediators* interact with the wrappers to process queries transparently. Our discussion assumes a simple relational view of data. Specifically, wrappers present each source as a set of *source relations*. We believe our framework is not sensitive to the data model; *e.g.,* in reference [24] we discuss the translation of hierarchical data.

A mediator exports integrated *mediator views* (as query interface) for users to formulate queries. Thus, a *user query* $\mathcal{U}$ over some views $\mathtt{V_i}$ has the form (in an SQL-like expression) **select** ... **from** $\mathtt{V_1}, \ldots, \mathtt{V_h}$ **where** C, or algebraically $\mathcal{U} = \sigma_C(\mathtt{V_1} \times \cdots \times \mathtt{V_h})$, where C is a Boolean expression of *constraints*. (The projection operation is omitted as it is irrelevant to our discussion.) Note again that we do not consider negation in this paper. A constraint is either a *selection* condition [$\mathtt{V_i}$.attr1 op value] or a *join* condition [$\mathtt{V_i}$.attr1 op $\mathtt{V_j}$.attr2], where attr1 and attr2 are attributes of view $\mathtt{V_i}$ and $\mathtt{V_j}$ respectively. For simplicity, we may write a selection constraint as [attr1 op value] when the containing view of attr1 is clear from the context (such as in Example 1 where we considered only one integrated view).

A mediator view is typically an SPJ query over some source relations plus possibly some *data conversion* functions. For instance, view (title, ln, fn, review) might be a join of relation (title, review) from source T_1, (title, author) from

T_2, and a function NameLnFn(author, ln, fn) for converting author to last and first names. We can model such a function as a *conceptual relation* with the tuples [author, ln, fn] that "satisfy" the function. Note that in general a view can be a union of SPJ components; *e.g.*, a book view can be a union of two relations from two bookstore sources. In this case, we can process each component separately and union the results as is typically done.

For source execution, the mediator must rewrite a user query in terms of the source relations. Thus, with view expansion, $\mathcal{U}$ will be rewritten to the following form in Eq. 1, where $\mathbf{R}_i$ is the cross-product of all the source relation instances that a particular source T_i contributes to any queried views, and $\mathbf{X}$ is the cross product of the relevant conceptual relations. We specifically refer to the selection condition Q as a *constraint query*: In most cases Q is simply the user-query condition C, but in addition Q can also include the constraints used in the view definitions.

$$\mathcal{U} = \sigma_Q(\mathbf{R}_1 \times \cdots \times \mathbf{R}_n \times \mathbf{X}) \tag{1}$$

Intuitively, the *constraint mapping* problem is to *push* as much as possible the constraint query to the sources. That is, the mapping translates Q from the mediator's *original context* to the *target context* at each source. Note that the constraints in Q are generally not readily executable across different contexts. First, there exists *schema* difference between the views and the sources: The conversion functions in $\mathbf{X}$ can present new attributes (*e.g.*, ln and fn that replace author) or change data representations. Second, there exists *capability* differences: Unless the mediator only allows the least common denominator of what the sources support, the constraints can be beyond the capabilities of some sources.

Thus, constraint mapping will find the mapping of Q for each source T_i, denoted $\mathcal{S}_i(Q)$, to retrieve the relevant subset of $\mathbf{R}_i$. Because of different source capabilities, a perfect mapping such that $Q \equiv \mathcal{S}_i(Q)$ often does not exist. Suppose for now that $\mathcal{S}_i(Q)$ may return extra answers (*i.e.*, it has false-positives) but may not miss any answers (*i.e.*, it has no false-negatives). The mediator then combines these source results, passes them through the conversion functions, and post-processes with a *filter* query $\mathcal{H}$ (to remove false-positives) consisting of the residue conditions not fully pushed to the sources, *i.e.*,

$$\mathcal{U} = \sigma_{\mathcal{H}}[\sigma_{\mathcal{S}_1(Q)}(\mathbf{R}_1) \times \cdots \times \sigma_{\mathcal{S}_n(Q)}(\mathbf{R}_n) \times \mathbf{X}]. \tag{2}$$

Comparing Eq. 1 and Eq. 2, we obtain the essential property for a *correct* translation:

$$Q = \mathcal{H} \wedge \mathcal{S}_1(Q) \wedge \cdots \wedge \mathcal{S}_n(Q) \tag{3}$$

We next illustrate this translation problem with Example 3, which considers a mediator that integrates two sources.

Example 3 Consider a mediator for two sources. Suppose that source T_1 provides relation paper(ti, au) for paper titles and authors and aubib(name, bib) for author names and their bibliography. The second source T_2 has prof(ln, fn, dept) for professor last, first names, and departments.

To illustrate, suppose that the mediator exports a faculty view fac(ln, fn, bib, dept) integrated from aubib and prof, and a publication view pub(ti, ln, fn) from paper(ti, au). In particular, the fac view joins aubib and prof through a conceptual relation (a conversion function) NameLnFn(name, ln, fn) with some view-definition conditions joining the name related attributes.

Suppose that a user is looking for the papers written by some CS faculty interested in data mining. The constraint query Q includes both selection and join constraints as the following. (Note that we omit the view-definition conditions in Q, as they will be processed in the mediator rather than the sources.)

$$\begin{aligned} Q = {} & a\text{:[fac.ln = pub.ln]} \wedge b\text{:[fac.fn = pub.fn]} \wedge \\ & c\text{:[fac.bib contains data(near)mining]} \wedge \\ & d\text{:[fac.dept = cs]}. \end{aligned}$$

Let's first consider the mapping for source T_1, *i.e.*, for relations paper and aubib. The join conditions $a \wedge b$ together map to x_1 : [paper.au = aubib.name]. If T_1 does not support the proximity operator near, rather than dropping constraint c, we can relax it to (x_2:[aubib.bib contains data] $\wedge$ x_3:[aubib.bib contains mining]), which requires only the occurrences of keywords. Lastly, constraint d maps to $True$ (it can only be processed in T_2). Thus, $\mathcal{S}_1(Q) = x_1 \wedge x_2 \wedge x_3$.

We next perform the mapping for source T_2, which contributes relation prof. All the constraints except d map to $True$. Suppose that T_2 uses department code 230 for CS, thus $\mathcal{S}_2(Q) =$ [prof.dept = 230].

Finally, the filter query $\mathcal{H}$ is simply the constraint c (*i.e.*, $\mathcal{H} = c$), the only constraint that is not fully realized at the underlying sources. Thus, $Q = \mathcal{H} \wedge \mathcal{S}_1(Q) \wedge \mathcal{S}_2(Q)$. □

For an *exact* query processing that cannot miss any potential answers, $\mathcal{S}_u(Q)$ as the *closet mapping* of Q must logically subsume Q, *i.e.*, $\mathcal{S}_u(Q)$ will retrieve a superset of what Q does as just illustrated. However, in many other applications, a mediator may be willing to tolerate false-negatives as well as false-positives, *e.g.*, to explore more cost-effective native queries. Therefore, while post-filtering can remove false-positives, the result will be a subset of what Q would return were it supported. In other words, Eq. 3 becomes

$$Q \supseteq \mathcal{H} \wedge \mathcal{S}_1(Q) \wedge \cdots \wedge \mathcal{S}_n(Q) \tag{4}$$

This paper specifically addresses the constraint mapping problem, *i.e.*, translating Q into $\mathcal{S}_u(Q)$ which best approximates Q. The derivation of filter queries $\mathcal{H}$ is thus beyond the scope of this paper. A filter query can simply be the original query, *i.e.*, $\mathcal{H} = Q$, or it can consist of essentially only those constraints whose translated versions may retrieve false positives. Reference [36] discusses how to derive filters with the least processing cost.

As we can perform the mappings for different sources separately (as illustrated), we will focus on a particular source

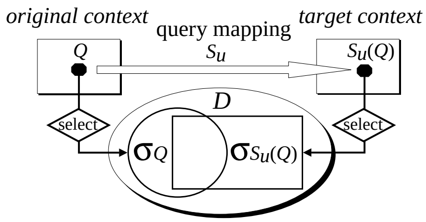


Fig. 1 Conceptual illustration of query mapping.

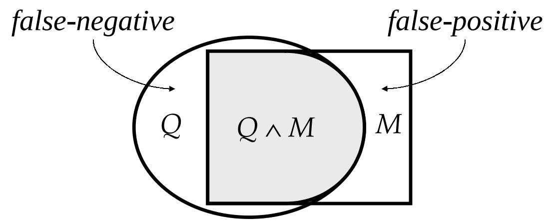


Fig. 2 Venn diagram of a query Q and its mapping M.

T_u as the translation *target* and discuss the requirements for $\mathcal{S}_u(Q)$: To begin with, $\mathcal{S}_u(Q)$ must be *supported* (or expressible) in target T_u; *i.e.*, $\mathcal{S}_u(Q)$ contains only those constraints that T_u supports with its schema and capability. Thus, $\mathcal{S}_u(Q)$ adopts the native vocabulary of T_u.

Furthermore, $\mathcal{S}_u(Q)$ is the *closest approximation* of Q with respect to some *closeness* criterion that measures how close a mapping is to a query. The goal of query mapping is to find the "best translation" of an original query. That is, for a relation D (in this case $D = \mathbf{R}_1 \times \cdots \times \mathbf{R}_n \times \mathbf{X}$ as in Eq. 1), we want to find the mapping $\mathcal{S}_u(Q)$ such that the result set $\sigma_{\mathcal{S}_u(Q)}(D)$ is the closest to $\sigma_Q(D)$ among all possible mappings with respect to some metric that compares the positive and negative errors between the results. Figure 1 shows this query mapping.

A main contribution of this paper is a general framework and algorithms for query constraint mapping. Because different mediator applications may need different senses of closeness, we must develop a general framework that supports a flexible notion of closeness. This paper will formally develop our closeness notion and the mapping algorithms that guarantee closest translations of a constraint query Q. Note that from now on we will simply refer to such Q as a *query* (not to be confused with a full *user query* $\mathcal{U}$). Also, we write the mapping as $\mathcal{S}(Q)$ (without a subscript) when the target source is clear. To simplify our discussion, we will assume selection constraints; join constraints can be handled similarly.

We stress that the constraint mapping problem focuses on translating the vocabulary but not the syntactic structure of queries. Since our goal is to generate mappings that use only native constraints, we make the following assumption to stress the focus of constraint mapping. While a target may not satisfy the Boolean closure assumption, many related efforts have addressed this complementary problem of translating Boolean structures, such as mapping a long conjunction into multiple native queries with simpler conjunctions.

Assumption 1 (Boolean Closure) *The queries supported by a target source are closed under Boolean conjunctions and disjunctions: If queries Q_1 and Q_2 are both supported, then so are $(Q_1 \wedge Q_2)$ and $(Q_1 \vee Q_2)$.* □

4 Closeness: 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 more closely 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 2 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. *Precision* measures the proportion of the mapping results that are correct:

$$\mathcal{P}[M, Q] = \frac{|Q \wedge M|}{|M|}. \tag{5}$$

(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 2 suggests, precision will increase as we reduce the number of false-positives.

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

$$\mathcal{R}[M, Q] = \frac{|Q \wedge M|}{|Q|}. \tag{6}$$

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

First, consider M_1:[bimonth = 1:3] (for 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. 6, as $Q \wedge M_1 = Q$, the recall is *perfect*, *i.e.*, $\mathcal{R}[M_1, Q] = 1$. Furthermore, according to Eq. 5, 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:[bimonth = 2] is a subset of Q ($M_2 \subseteq Q$). As Figure 3(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.*, $\mathcal{R}[M_2, Q] = 2/4 = .5$.

A mapping may have neither perfect precision nor perfect recall. For M_3:[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). □

For quantifying 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:

$$\textit{RThresh}(\theta) : \mathcal{F}(\mathcal{P}, \mathcal{R})[M, Q] = \mathcal{F}(\mathcal{P}[M, Q], \mathcal{R}[M, Q]) = \begin{cases} \mathcal{P}[M, Q] & \text{if } \mathcal{R}[M, Q] \geq \theta \\ \textit{undefined} & \text{otherwise} \end{cases} \quad (7)$$

Example 5 Consider the mappings in Example 4 and assume that $\mathcal{F}$= *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 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 with respect to *RThresh*$(.7)$. □

We similarly define *PThresh*(θ) as follows:

$$\textit{PThresh}(\theta) : \mathcal{F}(\mathcal{P}, \mathcal{R})[M, Q] = \mathcal{F}(\mathcal{P}[M, Q], \mathcal{R}[M, Q]) = \begin{cases} \mathcal{R}[M, Q] & \text{if } \mathcal{P}[M, Q] \geq \theta \\ \textit{undefined} & \text{otherwise} \end{cases} \quad (8)$$

The *PThresh* and *RThresh* classes represent many intuitive and important closeness metrics. We stress two special instances typically adopted for query mapping, namely *RThresh*(1) and *PThresh*(1). First, some applications may require perfect recall and hence *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* [25,26]. We designate *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 *PThresh*(1) with perfect precision. Unlike *MinSup*, the goal now is to find the *maximal subsumed mapping* or the *tight lower envelope* [25, 26]. We thus refer to *PThresh*(1) as *MaxSub*.

Example 6 Different closeness criteria will determine different mappings as the closest translations. Example 5 showed that $\mathcal{S}(Q) = M_3$ *w.r.t.* $\mathcal{F}$= *RThresh*$(.7)$ in our calendar example. Let's contrast with $\mathcal{F}$= *MinSup*: We obtain $\mathcal{F}[M_1, Q]$ = $\mathcal{F}(.67, 1)$ = .67, $\mathcal{F}[M_2, Q]$ = $\mathcal{F}(1, .5)$ = *undefined*, and $\mathcal{F}[M_3, Q]$ = $\mathcal{F}(.75, .75)$ = *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 [26]) 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* [34], a conventional single-valued measure for information retrieval.

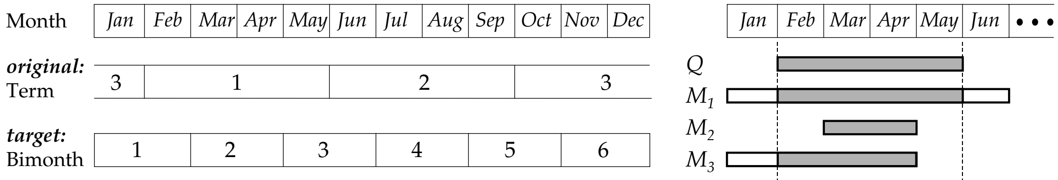
Finally, we stress that our general framework (Section 6) does not assume particular metrics. However, we do require that the closeness criterion be (strictly) *monotonic*. That is, if $P_1 \leq P_2$ and $R_1 < R_2$, or if $P_1 < P_2$ and $R_1 \leq R_2$, then $\mathcal{F}(P_1, R_1) < \mathcal{F}(P_2, R_2)$. 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 one parameter while comparable (at least as good) in the other, 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 separability theorems in Section 5.2.)

5 Query Compositions

Our translation approach is by divide-and-conquer (as Section 1 mentioned), which presents two main challenges. First, we must define a *rule system* to capture human expertise for translating basic *semantic units* (whose mappings cannot be synthesized). Second, to translate *composite queries*, we need a *machinery* to decompose them into semantic units and thus synthesize the mappings. (Section 6 will study the framework and algorithm.) To enable and understand such decompositions, this section addresses two fundamental questions.

The first question (Section 5.1) is about compositional monotonicity. For instance, if we can translate query $Q = A \wedge B$ as $M_a \wedge M_b$ by combining some separate mappings M_a and M_b of components A and B respectively, can we focus on the best component mappings and thus use $\mathcal{S}(A)$ for M_a and $\mathcal{S}(B)$ for M_b? Or will somehow the translation M_a depend on the fact that it will be eventually intersected with M_b?

Further, Section 5.2 addresses the second question about compositional separability: whether in general it is possible



(a) Correspondence of term and bimonth.

(b) Mappings for Q = [term = 1].

Fig. 3 Mapping term constraints to bimonth constraints.

to find the best mapping for a query like $Q = A \wedge B$ by separately translating its components A and B.

5.1 Compositional Monotonicity

Consider query composition $Q = Q_1 \odot \cdots \odot Q_n$, where operator $\odot$ is either $\wedge$ or $\vee$. Suppose that the mapping rules give translations for basic semantic units $m_1, \cdots, m_u$. To construct $\mathcal{S}(Q)$ with the units, our decomposing machinery will rewrite Q using the units, *i.e.*, $Q = B(m_1, \ldots, m_u)$ and synthesize $\mathcal{S}(Q) = B(\overline{m_1}, \ldots, \overline{m_u})$, using the rule-given best mappings $\overline{m_i}$ of the units.

The main challenge is thus to search for the right decomposition B. In this search, can we separately consider Q_i? Section 5.2 will show that such separation is not always the right strategy; our safety result (Appendix C) then determines when it is safe to separate.

If the separation of Q_i is determined to be safe, how shall we continue to decompose the components to complete the decomposition of Q? Can we decompose Q_i as if we were constructing its best mapping $\mathcal{S}(Q_i)$ independent of its context Q? Assumption 2 reduces our search space when searching for component mappings.

Assumption 2 (Compositional Monotonicity) *Let query Q be a composition $Q = Q_1 \odot \cdots \odot Q_n$ and $m_1, \ldots, m_u$ be the semantic units defined by rules.*

If $\mathcal{S}(Q)$ can be constructed by separately mapping the components, i.e., $\forall i, \exists B_i(m_1, \ldots, m_u) \equiv Q_i$ as a rewriting of Q_i such that

$$\mathcal{S}(Q) = B_1(\overline{m_1}, \ldots, \overline{m_u}) \odot \cdots \odot B_n(\overline{m_1}, \ldots, \overline{m_u}),$$

then each decomposition B_i will construct the best mapping of Q_i individually, i.e., $\forall i$

$$\mathcal{S}(Q_i) = B_i(\overline{m_1}, \ldots, \overline{m_u}).$$

□

This assumption tells us that, if we wish to search for the best way to decompose a query by decomposing its components, we can focus on using the "local optimals" as the building blocks, *i.e.*, $\mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n)$, 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. (We study this "separability" issue in Section 5.2.) It only says that $\mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n)$ is the best of the mappings for Q that separate components, *i.e.*, when the separation is *given*. (Specifically, Appendix C studies when such separation is safe, the premise of Assumption 2.)

Under certain closeness metrics, such as *MinSup* as well as *MaxSub*, we can formally verify Assumption 2 (see Appendix A). 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 2 should be valid. Otherwise, when Q_i's are indeed interrelated, $\mathcal{S}(Q)$ probably cannot be constructed by separating the components. For such "inseparable" compositions (Section 5.2), our algorithm will not separately handle Q_i and thus will not use Assumption 2. 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.

5.2 Compositional Separability

When translating a composition $Q = Q_1 \odot \cdots \odot Q_n$, can we handle the subqueries separately? We say that Q is *separable* if

$$\mathcal{S}(Q) = \mathcal{S}(Q_1) \odot \cdots \odot \mathcal{S}(Q_n). \tag{9}$$

That is, we can obtain the overall mapping by translating the components individually. (Observe that Assumption 2 is implicit in Eq. 9: When Q is separable, we use $\mathcal{S}(Q_i)$ to construct $\mathcal{S}(Q)$.) This section presents our results showing how this separability depends on the closeness criteria. Example 7 shows that *MinSup* and *MaxSub* have opposite results of separability.

Example 7 (Separability) Consider queries t_1:[term = 1] and t_2:[term = 2] (for the calendar systems in Example 4). 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 4 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 6 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 $\mathcal{S}(t_1 \wedge t_2) = False$, 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 $\mathcal{S}(t_1 \wedge t_2) = \mathcal{S}(t_1) \wedge \mathcal{S}(t_2) = False$(see Figure 4). 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]. □

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 t_1 and t_2 alone. When we consider the disjuncts together, the broader condition may (depending on the actual semantics) lead to a broader mapping (than the individual disjunct mappings 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 4 we have explained that metric $\mathcal{F}$ is monotonic in $\mathcal{P}$ and $\mathcal{R}$.) In Example 7, $\mathcal{S}(t_1 \vee t_2)$ under *MaxSub* results in [bimonth = 2:4], where the additional bimonth 3 indeed reduces false-negatives without adding false positives.

Consequently, our results below state that disjunctions are always separable, *when and only when* the closest mappings under $\mathcal{F}(\mathcal{P}, \mathcal{R})$ already have maximal possible recall. In particular, note that *MinSup* (Example 7) explicitly requires $\mathcal{R} = 1$ for all mappings. Disjunctions are thus always separable, since any mappings (separated or not) already have perfect and thus maximal recall (which cannot be enhanced). As another example, the following $\mathcal{F}$= *RMono* (which is monotonic in $\mathcal{R}$) also requires maximal recall (but possibly $\mathcal{R} < 1$, unlike *MinSup*) and thus disjunctions are always separable:

$$\textit{RMono}: \quad \mathcal{F}(\mathcal{P}, \mathcal{R}) = \mathcal{R} \tag{10}$$

To contrast, because *MaxSub* may select a mapping with perfect precision but lower recall (than others), disjunctions can further enhance $\mathcal{R}$ and thus may not be separable (Example 7). Theorem 1 formally states this result. Please refer to Appendix B 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 1 (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 under $\mathcal{F}$ has maximal possible recall, i.e., for any query X and any arbitrary mapping M:

$$\mathcal{R}[\mathcal{S}(X), X] \geq \mathcal{R}[M, X].$$

□

Similarly, the dual of the above results can apply to conjunctions, as Theorem 2 states. That is, conjunctions are always separable when and only when $\mathcal{F}(\mathcal{P}, \mathcal{R})$ enforces maximal possible precision. Example 7 also supports this result, since conjunctions under *MaxSub* or *PThresh*(1) are separable but not under *MinSup*. Dual to *RMono*, the following $\mathcal{F}$= *PMono* (which is monotonic in $\mathcal{P}$) also requires maximal precision which makes conjunctions always separable:

$$\textit{PMono}: \quad \mathcal{F}(\mathcal{P}, \mathcal{R}) = \mathcal{P} \tag{11}$$

Theorem 2 (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 under $\mathcal{F}$ has maximal possible precision, i.e., for any query X and any arbitrary mapping M:

$$\mathcal{P}[\mathcal{S}(X), X] \geq \mathcal{P}[M, X].$$

□

Note that we can verify that *MinSup* and *RMono* as well as *MaxSub* and *PMono* satisfy Theorems 1 and 2 respectively. For these important special cases, the mapping problem is "simpler" because we can eliminate the complications of either disjunctions or conjunctions.

On the other hand, the results show that a general closeness criteria can imply inseparability for both types of compositions. It is clear that only a very restricted class of closeness functions can satisfy Theorem 1. These functions $\mathcal{F}$ (as some variations of *MinSup* and *RMono*) must illustrate $\mathcal{R}$-*monotonicity* for disjunctions to be always separable, *i.e.*,

$$\textit{if } \mathcal{F}(\mathcal{P}_1, \mathcal{R}_1) \geq \mathcal{F}(\mathcal{P}_2, \mathcal{R}_2) \textit{ then } \mathcal{R}_1 \geq \mathcal{R}_2.$$

Similarly, for conjunctions to be always separable, as Theorem 2 asserts, $\mathcal{F}$ (as variations of *MaxSub* and *PMono*) must illustrate $\mathcal{P}$-*monotonicity*, *i.e.*,

$$\textit{if } \mathcal{F}(\mathcal{P}_1, \mathcal{R}_1) \geq \mathcal{F}(\mathcal{P}_2, \mathcal{R}_2) \textit{ then } \mathcal{P}_1 \geq \mathcal{P}_2.$$

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 8 contrasts *RThresh*(0.7) with *RThresh*(1) or *MinSup*.

Example 8 Continue Example 7 with $\mathcal{F}$= *RThresh*$(.7)$. By Theorem 1, disjunctions may not be separable because under *RThresh*$(.7)$ a closest mapping can have non-maximal recall. To illustrate, referring to the *RThresh*$(.7)$ row in Figure 4, $(t_1 \vee t_2)$ 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). Further, the mapping algorithm must consider both types of compositions carefully. This paper studies such a general 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

$\mathcal{F}(\mathcal{P},\mathcal{R})$	$\mathcal{S}(t_1)$	$\mathcal{S}(t_2)$	$\mathcal{S}(t_1 \vee t_2)$	$\mathcal{S}(t_1 \wedge t_2)$
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* $(\mathcal{P},\mathcal{R}) = (1, 1)$
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* $(\mathcal{P},\mathcal{R}) = (1, 1)$

Fig. 4 Example closest mappings for t_1:[term = 1] and t_2:[term = 2] with respect to different closeness criteria.

for the general algorithm that we will provide.) In particular, since we focus on semantic (but not syntactic) translation, Assumption 1 implies that the target supports queries of arbitrary sizes (by compositions). If this is not the case, disjunctions can be inseparable even for *MinSup* and similarly conjunctions for *MaxSub*.

Example 9 Consider translating conjunction

$$a\text{:[ln = "smith"]} \wedge\ b\text{:[fn = "john"]}.$$

Assume we can query the target with a full name, where *at least* the last name must be specified (*e.g.*, "smith,john" or "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 2, 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 $\mathcal{S}(a) \wedge \mathcal{S}(b)$

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

In practice such unbounded queries 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 $\mathcal{S}(a) \wedge \mathcal{S}(b) = \mathit{False}$. □

6 Framework and Algorithm

This section presents our framework and the associated algorithm for approximate translation. Section 6.1 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, such rewriting must respect compositional separability to ensure mapping optimality– Section 6.2 will present such an algorithm.

6.1 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 such 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 soundness and completeness requirements (see later). For instance, if necessary, our framework 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 7.

Figure 5 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 7 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 we next illustrate. (Note that, as Section 7 will discuss, we typically only need a rule for a query "pattern" rather than every "instance.") Figure 5 also shows the estimated $(\mathcal{P},\mathcal{R})$ for the particular 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 Section 8).

Example 10 (Mapping Rules) We illustrate rule R_1 and R_2 for mapping 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, *e.g.*, v= [format = vhs]. As an atomic constraint, it needs a rule to define its mapping. To illustrate, we have at least two choices: First,

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]	// $(\mathcal{P},\mathcal{R}) = (0.9, 1.0)$

Fig. 5 Example mapping specification K_{med} with respect to $\mathcal{F}$= *RThresh*$(.7)$.

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. 7), M_1 has a closeness value of $\mathcal{F}(.5, 1) = .5$. Alternatively, $M_2 =$ [desc contains vhs] is a mapping that 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 (*e.g.*, our implementation uses Java). □

Our discussion will assume an original query $Q_{med} = t(h \vee c)(v \vee d)$ as a running example. (Referring to Figure 6(a), we are querying the VHS or DVD titles by Tom Hanks or Tom Cruise.) 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.

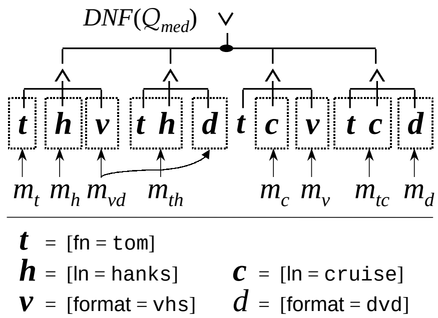
Example 11 (Rule Matching) Consider matching query Q_{med} to rules K_{med}; *i.e.*, we want to find the subqueries of Q_{med} that match a pattern described by some rule in K_{med}. Since a matching can be any complex query (with conjunctions, disjunctions, or both), we perform the matching on some 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 6.2 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 $\hat{D}_1 \vee \cdots \vee \hat{D}_n$. (Recall that we write $\hat{X}$ to stress that it is a conjunction and similarly $\check{X}$ for a disjunction). For Q_{med} we have $\hat{D}_1 = (thv), \ldots, \hat{D}_4 = (tcd)$ (see Figure 6). 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 6) 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 6(a) indicates these matchings as some subtrees of Q_{med}'s DNF. Figure 6(b) summarizes each matching m, the rule output for m (denoted by $\overline{m}$), and the estimated $(\mathcal{P},\mathcal{R})$ (from Figure 5). (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* re-



(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.

Fig. 6 Example query $Q_{med} = t(h \vee c)(v \vee d)$ and its matchings with respect to K_{med}.

quirement (*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 10 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, as Section 7 will discuss.

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 5.2) 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 (according to Theorem 1), 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] ∨ [format = dvd].) Similarly, we may expect a semantic unit with conjunctions (by Theorem 2), *e.g.*, R_7.

In fact, as we discussed in Section 5.2, for any closeness metric other than those with either $\mathcal{R}$-monotonicity or $\mathcal{P}$-monotonicity, neither disjunctions nor conjunctions are always separable, and thus a semantic unit may be any complex query. 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 *divide-and-conquer* approach. To translate an original query Q, if Q can match a rule (*i.e.*, it is a semantic unit), then 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 6, 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 next discuss.

6.2 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 6): *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 call such a rewriting a *decomposition*, since it breaks the query into the semantic units.

There exist *many* decompositions for a 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 a safe decomposition– We can show that its conjunction and disjunction are both separable (respectively as Example 17 and 14 of Appendix C discuss), and thus B_1 is safe. Because $Q_{med} \equiv B_1$, we can obtain $\mathcal{S}(Q_{med})$ or $\mathcal{S}(B_1)$ as

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) Rule Matching:
• 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.
• find all the matchings $m_1, \ldots, m_u$ of Q *w.r.t.* K by comparing $\mathit{DNF}(Q)$ and $\mathit{DNF}(m_i)$
(2) Safe Decomposition:
• $B(m_1, \ldots, m_u) \leftarrow \mathit{SafeDecom}(Q, m_1, \ldots, m_u)$
(3) 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 $\mathit{SafeDecom}(Q, m_1, \ldots, m_u) \rightarrow B(m_1, \ldots, m_u)$, the unique safe decomposition of Q
(1) Atomic Unit: // *terminal case: if* Q *is already an atomic unit.*
• if $Q = m_i$, *i.e.*, Q is a matching: return m_i
// *otherwise, decompose* Q *into a non-trivial safe conjunction or disjunction.*
(2) Safe Conjunction:
• $Q \equiv C_1 \wedge \ldots \wedge C_n \leftarrow \mathit{SafeConj}(Q, m_1, \ldots, m_u)$
• if $n > 1$: // *non-trivial conjunction, i.e.,* Q *is decomposed into more than one conjunct.*
// *recursively decompose the simpler components.*
– return $\mathit{SafeDecom}(C_1, m_1, \ldots, m_u) \wedge \ldots \wedge \mathit{SafeDecom}(C_n, m_1, \ldots, m_u)$
(3) Safe Disjunction:
• $Q \equiv D_1 \vee \ldots \vee D_n \leftarrow \mathit{SafeDisj}(Q, m_1, \ldots, m_u)$
• if $n > 1$: // *non-trivial disjunction, i.e.,* Q *is decomposed into more than one disjunct.*
// *recursively decompose the simpler components.*
– return $\mathit{SafeDecom}(D_1, m_1, \ldots, m_u) \vee \ldots \vee \mathit{SafeDecom}(D_n, m_1, \ldots, m_u)$

Fig. 7 Algorithm *NFB* for approximate query translation.

$[\mathcal{S}(m_{th}) \vee \mathcal{S}(m_{tc})]\mathcal{S}(m_{vd})$ (by separating every composition since B_1 is safe). Applying the rules for the units (Figure 6), we can construct the mapping from B_1, *i.e.*, $\mathcal{S}(Q_{med})$ = $(\overline{m_{th}} \vee \overline{m_{tc}})\, \overline{m_{vd}}$ = ([name = "hanks,tom"] ∨ [name = "cruise,tom"])∧[type = video].

Therefore, the main challenge for mapping a query is to find its safe decomposition. Our results (as we will see in Theorem 5) show that, in practical cases when such a decomposition exists (which is typical when semantic units do not "interlock"), the safe decomposition is unique (among many possible rewritings). Our Algorithm *NFB* (Figure 7) will find such a unique decomposition to construct the closest mapping.

Given a query Q and mapping rules K, Algorithm *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 7, *NFB* consists of three steps. We will illustrate by translating Q_{med} using K_{med} (which defines the mappings under $\mathcal{F}$= *RThresh*(.7)), as Figure 8 summarizes. First, Step (1) matches the rules to find the semantic units by comparing query patterns in DNF. Section 6.1 discussed this step, resulting in the matching units in Figure 6. Second, as we will explain next, Algorithm *NFB* formulates the safe decomposition $B = (m_{th} \vee m_{tc})(m_{vd})$ using the function *SafeDecom*. Finally, Algorithm *NFB* simply constructs $\mathcal{S}(Q_{med})$ from B as just discussed.

To address the main challenge of constructing the safe decomposition, *NFB* systematically rewrites a query based on Boolean normal forms (thus the name *NFB*, for Normal-Form Based). As Figure 7 shows, function *SafeDecom* performs this rewriting. The function takes as inputs a query Q and semantic units $m_1, \ldots, m_u$ and output the safe decomposition $B(m_1, \ldots, m_u)$. As a recursive procedure, the function decomposes Q and the result subqueries into the semantic units. Obviously, if Q is itself a semantic unit, no rewriting is necessary; its mapping is defined by a rule. Step (1) handles this case, which terminates the recursive procedure.

Otherwise, a query that is not a semantic unit can be decomposed (according to the completeness requirement discussed in Section 6.1). In other words, we want to rewrite such a query into a (conjunctive or disjunctive) composition of *simpler* queries that are separable. Step (2) of *SafeDecom* will first try to rewrite Q into a separable conjunction (or a *safe* conjunction) using subroutine *SafeConj*. When such a safe conjunction does not exist, in which case *SafeConj* will return a trivial rewriting with a single conjunct (*i.e.*, Q itself), Step (3) will then decompose Q into a safe disjunction. (This order of conjunction then disjunction is not critical.) We will discuss later that whenever a non-trivial safe conjunction exists, *SafeConj* will return non-trivial conjunctions, which similarly holds for *SafeDisj*. Thus when Q can be decomposed (conjunctively or disjunctively), either *SafeConj* or *SafeDisj* will make progress. In either case, *SafeDecom* continues to recursively rewrite the new components into simpler ones, until it eventually terminates when all components become semantic units.

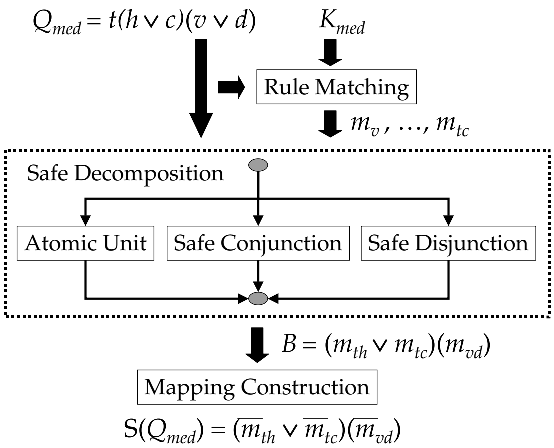


Fig. 8 Illustration of Algorithm *NFB* for query Q_{med} with respect to rules K_{med}.

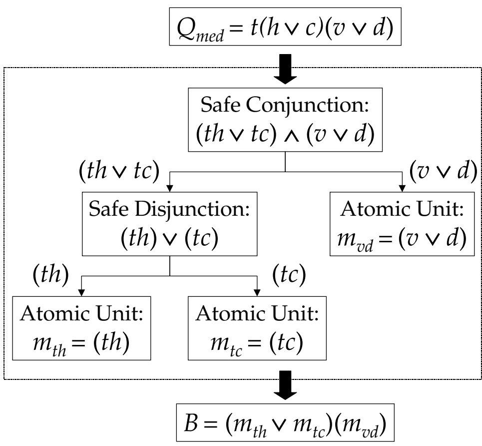


Fig. 9 The safe decomposition process for query Q_{med}.

We illustrate how *SafeDecom*$(Q_{med}, m_v, \ldots, m_{tc})$ executes (as called by Algorithm *NFB*) in Figure 9. In Step (2), *SafeDecom* will rewrite Q_{med} (by calling *SafeConj*) into a safe conjunction $C_1{:}(th \vee tc) \wedge C_2{:}(v \vee d)$ which is separable, *i.e.*, $\mathcal{S}(Q_{med}) = \mathcal{S}(C_1)\mathcal{S}(C_2)$. Section 6.2.1 will explain this safe conjunction decomposition. The process then continues to decompose each new conjunct recursively. Since C_1 is not yet an atomic unit (thus Step (1) will fail) and it cannot be decomposed conjunctively (thus Step (2) will also fail; see Section 6.2.1), Step (3) will then rewrite C_1 as a safe disjunction $D_1{:}(th) \vee D_2{:}(tc)$ which guarantees $\mathcal{S}(C_1) = \mathcal{S}(D_1) \vee \mathcal{S}(D_2)$. Section 6.2.2 will discuss this safe disjunction decomposition. Finally, the recursive decomposition of D_1, D_2 and C_2 will all be handled by Step (1), which returns atomic unit m_{th}, m_{tc}, and m_{vd} respectively. These "ground" cases will terminate *SafeDecom*$(Q_{med}, m_v, \ldots, m_{tc})$ to output the safe decomposition $B = (m_{th} \vee m_{tc})(m_{vd})$.

6.2.1 Safe Conjunction Decomposition We now focus on the algorithm *SafeConj* (in Figure 10) for rewriting 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 sufficient conditions (called *safety* conditions) that imply separability. While not explicitly used by the algorithm, this safety is an essential underlying principle– We thus present the formalism in Appendix C (Theorem 6 and 7). We stress here that (as Theorem 3 will state) any conjunction that *SafeConj* formulates will satisfy our formal safety conditions (*i.e.*, Theorem 7 in Appendix C) and thus must be separable.

As just discussed, we will illustrate by rewriting $Q_{med} \equiv C_1{:}(th \vee tc) \wedge C_2{:}(v \vee d)$. Intuitively, to eventually form a safe decomposition of Q_{med} using the semantic units (see Figure 6), we first form a safe decomposition for every conjunction in the former using the conjunctions in the latter. We perform this process systematically by comparing them in DNF, where conjunctions are explicit at the leaves (of the query tree), *e.g.*, Q_{med} has (thv), (thd), (tcv), and (tcd) as Figure 6 shows. In particular, we can rewrite (thv) as $(t)(h)(th)(v)$ with four "subconjunctions" from units m_t, m_h, m_{th}, and m_{vd}. We can 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. Since the above rewriting is disjunctive, Step (2) of *SafeConj* simply distributes the outer disjunction over the inner conjunctions (using the standard Boolean algebra). Omitting redundant conjuncts, we obtain a conjunctive form $Q_{med} = \check{E}_1\check{E}_2$ with two *essential candidate conjuncts* (which will form the final conjuncts in the next step): $\check{E}_1 = (th \vee tc)$ and $\check{E}_2 = (v \vee d)$.

Finally, Step (3) of *SafeConj* will eliminate conjunction redundancies by grouping those $\check{E}_j$'s with redundant conjunction factors. We will explain this grouping separately. Since there is no redundancy between $\check{E}_1$ and $\check{E}_2$, Step (3) will simply group the conjuncts into $\mathcal{E}_1 = \{\check{E}_1\}$ and $\mathcal{E}_2 = \{\check{E}_2\}$, and then formulate two final conjuncts accordingly: *i.e.*, $Q_{med} = C_1 \wedge C_2$, where $C_1 = \check{E}_1$ and $C_2 = \check{E}_2$. (Example 17 in Appendix C verifies that this result is correct, *i.e.*, the conjunction is safe and thus separable.)

In general, redundancies between conjuncts (see Definition 1) should be eliminated (by grouping conjuncts) for optimal mapping. For approximate mappings with *arbitrary* recall, our conjunction safety results (Theorem 7, Appendix C) require such non-redundancy for conjuncts to be separable. Intuitively, since recall may not be perfect (*i.e.*, $\mathcal{R} < 1$), query subsumption may *not* be preserved in mapping, *i.e.*, $X \supseteq Q$ does not imply $\mathcal{S}(X) \supseteq \mathcal{S}(Q)$. A redundant factor (such as X in $Q = XY$) may thus lead to suboptimal mappings, rather than remaining redundant in mapping. (However, in a special case such as $\mathcal{F}=$ *MinSup* when perfect recall is guaranteed, conjunction redundancies can be retained since query subsumption will be preserved [3].)

Definition 1 defines conjunction redundancies. To illustrate, consider query $Q = abc \vee bd$; suppose that Step (2) of *SafeConj* forms four candidate conjuncts: $\check{E}_1 = ab \vee b = b$, $\check{E}_2 = ab \vee d$, $\check{E}_3 = c \vee b$, and $\check{E}_4 = c \vee d$ (with match-

Function *SafeConj*$(Q, m_1, \ldots, m_u) \to Q \equiv C_1 \wedge \cdots \wedge C_n$ as a safe conjunction

(1) *Conjunction Rewriting*:
- let $\mathit{DNF}(Q) = \sum_{l=1}^{m} \hat{D}_l$; let $\mathit{DNF}(m_i) = \sum \hat{d}_{ij}$

// rewrite each $\hat{D}_l$ using the DNF disjuncts of matching m_i found in Q.
- for all $\hat{D}_l$: rewrite $\hat{D}_l \equiv \prod \hat{x}_{lk}$, where each $\hat{x}_{lk}$ denotes some $\hat{d}_{ij}$ in $\mathit{DNF}(m_i)$ *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 $\hat{d}_{i'j'}$.*

(2) *Conjunction Formulation*: *// standard Boolean algebra to convert Q into a conjunctive form.*
- rewrite Q into a conjunctive form (as a CNF of the $\hat{x}_{lk}$'s), *i.e.*, $Q \equiv \prod_{j=1}^{e} \check{E}_j$, for all $\check{E}_j$ *s.t.*:

  // generate a candidate conjunct *by combining $\hat{x}_{lk_l}$ in disjunctions.*
  - $\check{E}_j = \hat{x}_{1k_1} \vee \cdots \vee \hat{x}_{mk_m}$, *i.e.*, one $\hat{x}_{lk_l}$ from each $\hat{D}_l$ just rewritten
  - $\check{E}_j$ is not redundant in the conjunctive form of Q:

    // $\check{E}_j$ is not redundant if it has at least a conjunction factor that is not redundant in Q.
    - $\exists X$ (as a conjunction factor) in $\mathit{CNF}(\check{E}_j)$ *s.t.* $\forall i \neq j$ and $\forall Y$ in $\mathit{CNF}(\check{E}_i)$, $X \not\supseteq Y$
- $\mathcal{E} \leftarrow \{\check{E}_1, \ldots, \check{E}_e\}$ *// let $\mathcal{E}$ be the* essential candidate conjuncts *so formulated.*

(3) *Conjunct Grouping*:

// eliminate redundancies: merge essential candidate conjuncts with redundant factors into a group.
- group $\check{E}_1, \ldots, \check{E}_e$ (*i.e.*, partition $\mathcal{E}$) into disjoint sets $\mathcal{E}_1, \ldots, \mathcal{E}_n$, *i.e.*, $\mathcal{E} = \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_n$ *s.t.*
  - $C_i \leftarrow \prod_{\check{E}_j \in \mathcal{E}_i} \check{E}_j$ *// let C_i be the merged conjunction of the conjuncts in $\mathcal{E}_i$.*
  - there is no conjunction redundancy across C_i's: $\forall X$ in $\mathit{CNF}(C_i), \forall Y$ in $\mathit{CNF}(C_j), X \not\supseteq Y$

(4) *Output*: return $Q \equiv C_1 \wedge \cdots \wedge C_n$

Function *SafeDisj*$(Q, m_1, \ldots, m_u) \to Q \equiv D_1 \vee \cdots \vee D_n$ as a safe disjunction
This procedure straightforwardly parallels *SafeConj* as a dual algorithm; we thus omit the details.

Fig. 10 Safe conjunction and disjunction decomposition.

ings ab, a, b, c, and d). Step (3) will then group them into $\mathcal{E}_1 = \{\check{E}_1, \check{E}_2, \check{E}_3\}$ and $\mathcal{E}_2 = \{\check{E}_4\}$. In particular, comparing $\mathit{CNF}(\check{E}_1) = (b)$, $\mathit{CNF}(\check{E}_2) = (a \vee d)(b \vee d)$, $\mathit{CNF}(\check{E}_3) = (c \vee b)$, we observe that both $(b \vee d)$ in $\check{E}_2$ and $(c \vee b)$ in $\check{E}_3$ are redundant with respect to (b) in $\check{E}_1$, *i.e.*, $(b \vee d) \supseteq (b)$ and $(c \vee b) \supseteq (b)$. Consequently, they are grouped together to eliminate the redundancies. Thus *SafeConj* will output $Q = C_1 \wedge C_2$, where $C_1 = \check{E}_1 \check{E}_2 \check{E}_3$ and $C_2 = \check{E}_4$. Note that there are no redundancies between the final conjuncts.

Definition 1 (Conjunction Redundancy) *For a conjunction $C_1 \wedge \cdots \wedge C_n$, there exists a* conjunction redundancy *between C_i and C_j (where $i \neq j$) if $X \supseteq Y$, for some conjunct X in CNF(C_i) and Y in CNF(C_j).* □

Finally, there are cases when a safe conjunction does not exist, in which case *SafeConj* will return Q in its entirety. For instance, consider further rewriting $C_1 = th \vee tc$ of Q_{med} (as we just obtained above). Step (1) of *SafeConj* cannot decompose (th) and (tc) because they respectively match m_{th} and m_{tc} entirely. Consequently, when a safe conjunction does not exist, *SafeConj* will simply return Q itself as a single conjunct (which is certainly safe but makes no progress). In this case, *SafeDecom* will then continue to call *SafeDisj* to find a safe disjunction instead.

6.2.2 Safe Disjunction Decomposition We can use the dual process of *SafeConj* (Figure 10) for constructing safe disjunctions. Given query Q (and its matchings), *SafeDisj* will rewrite Q into a disjunction that is guaranteed to be separable. (That is, the resulting disjunctions will satisfy Theorem 6 in Appendix C.)

Let us illustrate this dual process by rewriting $C_1 = th \vee tc$ of Q_{med}. We start to make disjunctions explicit using CNF: $\mathit{CNF}(C_1) = (t)(h \vee c)$. Comparing each disjunction to the disjunctions of the units (Figure 6), Step (1) of *SafeDisj* (dual to *SafeConj*) will leave (t) as is and decompose $(h \vee c)$ as $(h) \vee (c)$. Step (2) then formulates a disjunction $(th) \vee (tc)$ with two essential candidate disjuncts. Step (3) will merge the disjuncts with redundancies, which Definition 2 formally defines (parallel to Definition 1). Since there is no redundancy, *SafeDisj* will conclude $C_1 \equiv (th) \vee (tc)$ as a safe disjunction. (Dual to what we noted for conjunctions, disjunction redundancies will imply unsafety, unless perfect precision is guaranteed as in $\mathcal{F}$= *MaxSub*.)

Definition 2 (Disjunction Redundancy) *For a disjunction $D_1 \vee \cdots \vee D_n$, there exists a disjunction redundancy between D_i and D_j (where $i \neq j$) if $X \subseteq Y$, for some disjunct X in DNF(D_i) and Y in DNF(D_j).* □

6.2.3 Algorithm NFB By formulating a safe decomposition, Algorithm *NFB* will construct the best translation, as illustrated. Essentially, based on the optimal mappings of 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. Note that *NFB* relies on *SafeConj* and *SafeDisj* respectively for conjunction and disjunction rewriting. Therefore, the correctness of *NFB* is guaranteed if the rewritings of both compositions are safe. We state this correctness in Theorems 3 and 4, and give a proof in Appendix D.

Theorem 3 (*SafeConj* Correctness)
Given a query Q and the associated semantic units,

– (soundness) *SafeConj will rewrite Q into a safe conjunction, and*
– (liveness) *if there exists a non-trivial safe conjunction for Q, then the conjunction that SafeConj returns is also non-trivial.*

□

Theorem 4 (*SafeDisj* Correctness)
Given a query Q and the associated semantic units,

– (soundness) *SafeDisj will rewrite Q into a safe disjunction, and*
– (liveness) *if there exists a non-trivial safe disjunction for Q, then the disjunction that SafeDisj returns is also non-trivial.*

□

In addition to the soundness, Theorems 3 and 4 also assert the liveness of *SafeConj* and *SafeDisj*. Note that the algorithms may return a trivial composition (of the input query itself), which is obviously safe but makes no progress (since the goal is to decompose the query). However, the liveness guarantees that *SafeConj* and *SafeDisj* will do so only when a non-trivial composition does not exist. In other words, the dual algorithms will make progress whenever possible, and thus together eventually decompose an input query into semantic units.

Our results (Theorem 5) show that, for any query with a safe decomposition, Algorithm *NFB* will find the *unique* decomposition and thus construct the best mapping. (There are cases when a safe decomposition may not exist, which we discuss later.) Such a decomposition is uniquely safe, and thus the resulting best mapping is unique. Please refer to Appendix D for a proof.

Theorem 5 (Unique Safe Decomposition)
Given a query Q and a mapping specification K w.r.t. a closeness criterion $\mathcal{F}$, if a safe decomposition exists:

– (uniqueness) *the safe decomposition is* unique *for Q, i.e., any other distinct decomposition of Q must be unsafe, and*
– (correctness) *Algorithm NFB will find the unique safe decomposition and output $\mathcal{S}(Q)$ w.r.t. $\mathcal{F}$.*

□

However, depending on what semantic units are given, not all queries have a safe decomposition. When a safe decomposition does not exist, a query or some part of it (during the execution of *SafeDecom*) cannot be decomposed disjunctively or conjunctively. Consequently, neither *SafeConj* nor *SafeDisj* can return a non-trivial composition (as one does not exist). The following example illustrates such an "interlocking" case.

Example 12 Consider query $Q = xy \vee z$ with semantic units $m_{x\vee z} = x \vee z$, $m_{xy} = xy$, $m_x = x$, $m_y = y$, and $m_z = z$. Note that $m_{x\vee z}$ and m_{xy} are overlapping with a common constraint x, which appears in a disjunction in $m_{x\vee z}$ and a conjunction in m_{xy}.

We show that Q does not have a safe decomposition. 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 6 in Appendix C). 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 (formally by Theorem 7).

Since Q has no safe decomposition, Algorithm *NFB* cannot construct one. That is, as Q cannot be decomposed conjunctively or disjunctively (as just shown), neither *SafeConj* nor *SafeDisj* can make progress. □

When Q does not have a safe decomposition, it still has a best mapping, but $\mathcal{S}(Q)$ must instead be found among the unsafe decompositions. (Our completeness requirement in Section 6.1 asserts that $\mathcal{S}(Q)$ can be constructed from *some* decomposition using semantic units.) While it is beyond the scope of Algorithm *NFB*, we discuss possible ways for handling interlocking cases later.

Note that the existence of a safe decomposition depends on the semantic units defined. As Example 12 observed, the overlapping (of common constraint x) leads to the "interlocking" of conjunction and disjunction (thus neither *SafeConj* nor *SafeDisj* can make progress). We found (informally) that such interlocking units can typically result in the nonexistence of a safe decomposition. Definition 3 below characterizes semantic-unit interlocking. When a query involves interlocking units, it may not be decomposable conjunctively or disjunctively. Example 13 will show how different types of interlocking units can lead to unsafe decompositions.

Definition 3 (Interlocking Units) *Let m_1 and m_2 be two semantic units; suppose that DNF(m_i) = $\sum \hat{d}_{ij}$ and CNF(m_i) = $\prod \check{c}_{ij}$. We say that m_1 and m_2 are* interlocking, *if any of the following holds:*

1. (conjunction overlapping) *$\hat{d}_{1j}$ and $\hat{d}_{2k}$ overlap, and none is simpler than the other.*
2. (disjunction overlapping) *$\check{c}_{1j}$ and $\check{c}_{2k}$ overlap, and none is simpler than the other.*
3. (conjunction-disjunction overlapping) *$\check{d}_{1j}$ and $\hat{c}_{2k}$ overlap.*

□

Example 13 (Interlocking Units) For each type of interlocking in Definition 3, we show that a query containing interlocking units may not have a safe decomposition.

Case 1: Consider query $Q = xyz$ with matchings (written in DNF) $m_1 = (xy)$ and $m_2 = (xz)$. Note that m_1 and m_2 are interlocking, because $\hat{d}_{11} = xy$ and $\hat{d}_{21} = xz$ overlap (with a common constraint x), while one is not simpler than the other.

We show that Q does not have a safe decomposition. To see why, note that, as a simple conjunction, Q contains no disjunctions and thus can only be decomposed as a safe conjunction, if any. We can enumerate all the possible conjunctive rewritings (without redundancies): $B_1 = (x)(y)(z)$, $B_2 = (x)(yz)$, $B_3 = (xz)(y)$, and $B_4 = (xy)(z)$. None of them are separable, because any B_i will break the dependency of either (xy) of m_1 or (xz) of m_2. (We can show that all these rewritings will fail the safety test of Theorem 7 in Appendix C.)

It is instructive to observe how *SafeConj* and *SafeDisj* will both return a trivial rewriting. As Q contains no disjunctions, *SafeDisj* will return Q as a single disjunct. It is more interesting to see what *SafeConj* does: Step (1) will rewrite the (only) conjunction xyz as $(xy)(xz)$. After Step (2), it will formulate two candidate conjuncts (xy) and (xz). Step (3) will then merge the conjuncts (which contain a redundant factor x) and thus return a trivial conjunction.

Case 2: Consider query $Q = x \vee y \vee z$ with matchings $m_1 = (x \vee y)$ and $m_2 = (x \vee z)$. We can similarly show that, as the dual of Case 1, m_1 and m_2 are interlocking and there does not exist a safe decomposition for Q.

Case 3: Example 12 illustrated this case, where $m_{x \vee z}$ and m_{xy} are interlocking. □

We believe that such interlocking units (and thus the potential nonexistence of safe decompositions) will be rare in practice. As we noted above, interlocking 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) can participate in two different units (*e.g.*, m_{xy} and $m_{x\vee z}$) that are not simpler than each other. For increasingly more complex units, *e.g.*, (x), $(x \vee y)$, and $(x \vee y \vee z)$, no interlocking will form. 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 7), we observed no such "anomaly" instances.

It is interesting to observe that interlocking will not occur when either perfect recall or perfect precision is guaranteed, as in $\mathcal{F}=$ *MinSup* and $\mathcal{F}=$ *MaxSub*. For instance, consider the case $\mathcal{F}=$ *MinSup* with perfect recall, as Section 6.2.1 noted, conjunction redundancies– such as overlapping constraints in conjuncts– will not cause unsafety in mapping. Consequently, interlocking conjunctions in Case 1 of Definition 3 can indeed be safely separated. That is, a decomposition $B = (xy)(xz)$ for query Q in Example 13 (Case 1) is actually safe for *MinSup*. Further, with perfect (and thus maximal possible) recall, semantic units cannot have disjunctions (by Theorem 1). Thus, neither Case 2 nor Case 3 of Definition 3 can occur. Consequently, no semantic units will interlock. We can similarly show for $\mathcal{F}=$ *MaxSub*.

We stress that, in any case, one can avoid the nonexistence of safe decompositions by simply defining additional rules for "ambiguous" query patterns: When a safe decomposition does not exist, no rewritings are "clearly" the best for constructing a mapping. For instance, in Example 12, the choice between rewritings B_1 and B_2 illustrates such "ambiguity." This ambiguity in mapping can be resolved explicitly by a more "complete" mapping specification that defines additional rules for the interlocking patterns. For instance, we can define a rule for pattern $xy \vee z$ to remove the ambiguity.

When there does not exist a safe decomposition, 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 patterns (*e.g.*, $xy \vee z$) be defined by rules as just described. Alternatively, we can find all unsafe decompositions, estimate the closeness of each corresponding mapping, and select the best as $\mathcal{S}(Q)$. Note that it is possible to estimate $\mathcal{P}$ and $\mathcal{R}$ and thus the closeness of a constructed mapping; we discuss such estimation in Section 8.

Finally, we conclude by analyzing the running time of Algorithm *NFB*. The dominant cost is query normalization (which is not surprising since Algorithm *NFB* is normal-form based). First, in Step (1) *NFB* must convert the query into DNF for rule matching. Further, Step (2) will use subroutines *SafeConj* and *SafeDisj* (through *SafeDecom*), which will need DNF and CNF conversion. (Note that the normalizations can be done only once and reused throughout the process.) 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.

Further, the other steps of Algorithm *NFB* are quite efficient: Consider a query Q and rules K as input. Let D_Q and N_Q be the number of disjuncts and constraints-per-disjunct in *DNF*(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, rule matching of Step (1) will take time $O(N_Q N_R D_Q D_R R)$: It 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). Finally, Step (3) will take time of $O(M)$, where M is the number of matchings actually used.

7 Practical Implications: A Case Study

To verify that our framework makes sense in practice, we explore several 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. This case study is based on a similar scenario that is available online for demonstrating our translation server (see Section 9).

7.1 A Book-Search Mediator

Let's consider building a book-search mediator that integrates online sources[1] *Amazon* at `www.amazon.com` and *BN* at `www.barnsandnoble.com` (both are online bookstores), *EB* at `www.evenbetter.com` (as a comparison shopping service), and *Socrates* at `socrates.stanford.edu` (the 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 (*i.e.*, supported constraints as described in the specific query interface and the documentation) with that of *Amazon* and define the mapping rules. As Figure 11 shows, we need seven rules for *BN*, six rules for *Socrates*, and seven rules for *EB*. As Section 6.1 discusses, each rule gives the best mapping (using as much as possible the source query capabilities) 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 7.2.

For instance, rules $B_1, \ldots, B_7$ 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 rules, we only need to focus on the corresponding *clusters* of attributes 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.

7.2 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 6.2 discussed). Note that semantic units essentially indicate the correspondence of attributes, such as the *conjunction* 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 unit interlocking of Case 3 (Definition 3) simply cannot occur. For Case 1 (and similarly Case 2) to cause unsafe decompositions, as Example 13 illustrated, the same constraint (*e.g.*, $x =$ [fn = `tom`]) must appear in different semantic units (*e.g.*, xy and xz, where $y =$ [ln = `hanks`] and $z =$ [ln = `cruise`]), and *in addition* these units must appear in the same query composition (*e.g.*, $Q = xyz$). We again observed no such complex compositions (*e.g.*, in this example, xyz is in fact not a reasonable query). Our algorithm will thus generate unique best mappings in 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. That is, given a semantic unit, we can *intuitively* select its best mapping among competing choices (without explicitly computing their closeness with a metric as in Example 10). As long as the selection is reasonable (in particular it does prefer a mapping with both higher $\mathcal{P}$ and $\mathcal{R}$), there probably will be a monotonic function that supports the selection. Thus the selected mappings are effectively defined with some *implicit* metric that corresponds to the "intuition" of selection. However, we stress that, as Theorem 5 states, our algorithms will preserve the optimality with respect to any monotonic 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 11, only six (B_3, B_4, B_7, S_1, E_3, E_4) are perfect mappings, a ratio of 30% – the other 70% are 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 (*e.g.*, online real estate search and job opening search involving about ten sources) to the one presented here, and in all cases we have found our observations to hold.

8 Estimating Precision and Recall

While our algorithm generates best mappings, it does not compute how "close" they actually are. Given a query Q, Algorithm *NFB* finds the unique $\mathcal{S}(Q)$ (Theorem 5) by observing safety in decomposition; in particular, it does not use the $\mathcal{P}$ and $\mathcal{R}$ parameters to compute and compare closeness among choices.

[1] We performed this case study in February 2000.

[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*

Fig. 11 Rules for mapping *Amazon* to different sources.

However, in addition to finding best translations, in many cases it is desirable for a complete system to indicate the resulting closeness. For instance, such estimation can be useful for the following applications.

(query optimization) If there are alternative query plans, the estimation can enable cost-based optimization for distributed processing. For instance, among comparable sources (*e.g.*, similar online bookstores) for querying, a mediator may prefer one that can execute a query more closely (*i.e.*, with a high-closeness translation).

(unsafe decompositions) Our algorithm relies on a safe decomposition to find the best mappings, as Section 6.2 discussed. When a safe decomposition does not exist, the estimation can help to determine $\mathcal{S}(Q)$ among unsafe decompositions.

(system design) When designing a mapping system (choosing sources, determining a closeness metric, and developing rules), we believe that estimating closeness can be valuable for comparing various design decisions. For instance, if $\mathcal{F}$= *MinSup* results in low estimated precision, one may consider *RThresh*$(.5)$ instead.

This section therefore 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 show experimentally 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 Q's 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. 6:

$$\mathcal{R}_{a\wedge b} = \frac{|(A \wedge B) \wedge (M_a \wedge M_b)|}{|(A \wedge B)|} = \frac{|(A \wedge M_a) \wedge (B \wedge M_b)|}{|(A \wedge B)|} \quad (12)$$

We then must estimate the sizes of the queries in Eq. 12. 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 in translation. In particular, we assume that mappings 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. 14 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. 14 shows the $\mathcal{P}_{a\wedge b}$-estimate.

$$\mathcal{R}_{a\wedge b} \approx \frac{\sigma(A \wedge M_a) \cdot \sigma(B \wedge M_b) \cdot C_{ab}}{\sigma(A) \cdot \sigma(B) \cdot C_{ab}} = \frac{\sigma(A \wedge M_a)}{\sigma(A)} \cdot \frac{\sigma(B \wedge M_b)}{\sigma(B)} = \mathcal{R}_a \cdot \mathcal{R}_b \quad (13)$$

$$\mathcal{P}_{a\wedge b} \approx \frac{\sigma(A \wedge M_a) \cdot \sigma(B \wedge M_b) \cdot C_{ab}}{\sigma(M_a) \cdot \sigma(M_b) \cdot C_{ab}} = \frac{\sigma(A \wedge M_a)}{\sigma(M_a)} \cdot \frac{\sigma(B \wedge M_b)}{\sigma(M_b)} = \mathcal{P}_a \cdot \mathcal{P}_b \quad (14)$$

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. 6, we compute recall as

$$\mathcal{R}_{a\vee b} = \frac{|(A \vee B) \wedge (M_a \vee M_b)|}{|(A \vee B)|} \quad (15)$$

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. 15 as

$$\begin{aligned} &\sigma((A \vee B) \wedge (M_a \vee M_b)) \\ &= \sigma((A \wedge M_a) \vee (A \wedge M_b) \vee (B \wedge M_a) \vee (B \wedge M_b)) \\ &= \sigma(A \wedge M_a) + \sigma(A \wedge M_b) + \sigma(B \wedge M_a) + \sigma(B \wedge M_b) \\ &\quad + \cdots\cdots \end{aligned} \quad (16)$$

Observe that the first-order terms in Eq. 16 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. 6 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. 16 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. 15, 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. 18 shows.

$$\mathcal{R}_{a\vee b} \approx \frac{\sigma(A) \cdot \mathcal{R}_a + \sigma(B) \cdot \mathcal{R}_b}{\sigma(A) + \sigma(B)} = wavg[\sigma(A),\ \sigma(B)](\mathcal{R}_a, \mathcal{R}_b) \quad (17)$$

$$\mathcal{P}_{a\vee b} \approx wavg[\sigma(A) \cdot \frac{\mathcal{R}_a}{\mathcal{P}_a},\ \sigma(B) \cdot \frac{\mathcal{R}_b}{\mathcal{P}_b}](\mathcal{P}_a, \mathcal{P}_b) \quad (18)$$

9 Conclusion

In this paper we have presented a framework and the associated algorithm for approximate query translation. Our framework is robust under any closeness metric that monotonically combines both precision and recall. We also presented our results on the separability and safety of query compositions. 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 queries have safe decompositions. Finally, we presented simple formulas for estimating translation closeness.

Although we present our approach specifically for translating queries across contexts, we believe its generality can support much broader heterogeneous problems. For instance, the framework can be applied to map *data* and queries across *ontologies*. In fact, we have studied in [24] how to model data as conjunctive queries and thus apply the minimal-superset algorithms [3] for data translation.

We have implemented the approximate query translation mechanism described in this paper in the Stanford Digital Libraries Project. Our implementation of an online translation server is available for demonstration. While the backend translation server is generic, we program it (by defining specific mapping rules) to demonstrate a particular translation scenario of online media search (Section 7 presented a simplified version). The demonstration can be accessed at the URL `http://www-db.stanford.edu/~kevin/aqt`.

Acknowledgements We thank the anonymous referees for their review comments, which improved this paper and in particular helped to identify and correct an error in Theorem 1 (and the dual Theorem 2) of our initial manuscript.

This material is based upon work supported by the National Science Foundation under Cooperative Agreement IRI-9411306. In addition, funding for this cooperative agreement is also provided by DARPA, NASA, and the industrial partners of the Stanford Digital Libraries Project.

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. K. C.-C. 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. K. C.-C. 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. 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.
6. M. A. Hearst. Trends & controversies: Information integration. *IEEE Intelligent System*, 13(5):12–24, Sept. 1998.
7. 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.
8. 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.
9. 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.
10. 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.
11. O. M. Duschka. *Query Planning and Optimization in Information Integration*. PhD thesis, Stanford Univ., Dec. 1997.
12. 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.
13. 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.
14. 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.
15. O. Kapitskaia, A. Tomasic, and P. Valduriez. Dealing with discrepancies in wrapper functionality. Technical Report RR-3138, INRIA, 1997.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. K. C.-C. 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.
25. 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, Washingtion, D.C., 1993. ACM Press, New York.
26. 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.
27. 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.
28. 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.
29. 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.
30. 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.

31. 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.
32. 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.
33. 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.
34. C. J. V. Rijsbergen. *Information Retrieval, 2nd edition*. Butterworths, London, 1979.
35. G. Salton. *Automatic Text Processing*. Addison-Wesley, Reading, Mass., 1989.
36. K. C.-C. Chang. *Query and Data Mapping Across Heterogeneous Information Sources*. PhD thesis, Stanford Univ., Jan. 2001. Accessible at `http://www-faculty.cs.uiuc.edu/~kcchang/`.
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 Monotonicity

We show that Assumption 2 indeed holds for *MinSup* and *MaxSub*. We will specifically consider $\mathcal{F}$ = *MinSup* or *RThresh*(1); the dual case *MaxSub* can be shown similarly. Let $M_i = B_i(\overline{m_1}, \ldots, \overline{m_u})$ and $S_i = \mathcal{S}(Q_i)$. Assume that for some M_i: $M_i \neq S_i$ and consequently

$$\mathcal{F}[S_i, Q_i] > \mathcal{F}[M_i, Q_i],$$

where we assume they do not tie, or otherwise one can reassign to make $S_i = M_i$. We will show that

$$\mathcal{F}[M_1 \odot \cdots \odot M_n, Q] < \mathcal{F}[S_1 \odot \cdots \odot S_n, Q].$$

Therefore, $\mathcal{S}(Q) = M_1 \odot \cdots \odot M_n$ cannot hold and thus a contradiction.

Any constructed mapping M_i (including S_i) must have perfect recall, *i.e.*, $\mathcal{R}[M_i, Q_i] = 1$ or $Q_i \subseteq M_i$. Since $\overline{m_j}$ = $\mathcal{S}(m_j)$ as given by the rules with respect to $\mathcal{F}$= *RThresh*(1) (Eq. 7), it follows that $\forall j : m_j \subseteq \overline{m_j}$, and thus

$$Q_i = B_i(m_1, \ldots, m_u) \subseteq B_i(\overline{m_1}, \ldots, \overline{m_u}) = M_i.$$

We show that for those $M_i \neq S_i$: $M_i \supset S_i$. As showed above, $\mathcal{F}[S_i, Q_i] > \mathcal{F}[M_i, Q_i]$. Since $\mathcal{F}$= *RThresh*(1) as given in Eq. 7, it follows that $\mathcal{P}[S_i, Q_i] > \mathcal{P}[M_i, Q_i]$. Assuming that $M_i \not\supset S_i$, we shall derive a contradiction: First, $(S_i \wedge M_i)$ must be supported by the target source as both S_i and M_i are, according to Assumption 1. Second, the conjunction is a valid *MinSup* mapping of Q_i, *i.e.*, $S_i \wedge M_i \supseteq Q_i$, since $M_i \supseteq Q_i$ and $S_i \supseteq Q_i$. Furthermore, the conjunction is in fact a "smaller" mapping (note that *MinSup* will look for "minimal" superset-mappings): Since $M_i \not\supset S_i$ by assumption, we have $S_i \wedge M_i \subset S_i$. It will then follow from Eq. 5 that $\mathcal{P}[S_i, Q_i] < \mathcal{P}[S_i \wedge M_i, Q_i]$, *i.e.*, $(S_i \wedge M_i)$ will be a closer mapping than S_i, a contradiction.

Consequently, $\mathcal{M}_1 \odot \cdots \odot \mathcal{M}_n \supset S_1 \odot \cdots \odot S_n$ and thus the right side is a closer mapping for the composition $Q = Q_1 \odot \cdots \odot Q_n$ under *MinSup*: First, both have perfect recall because $\mathcal{M}_1 \odot \cdots \odot \mathcal{M}_n \supseteq Q$ as well as $S_1 \odot \cdots \odot S_n \supseteq Q$. Second, because $\mathcal{M}_1 \odot \cdots \odot \mathcal{M}_n \supset S_1 \odot \cdots \odot S_n$, it will follow from Eq. 5 that $\mathcal{P}[\mathcal{M}_1 \odot \cdots \odot \mathcal{M}_n, Q] < \mathcal{P}[S_1 \odot \cdots \odot S_n, Q]$. According to Eq. 7, we obtain that $\mathcal{F}[\mathcal{M}_1 \odot \cdots \odot \mathcal{M}_n, Q] < \mathcal{F}[S_1 \odot \cdots \odot S_n, Q]$. □

B Compositional Separability

Theorem 1 (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 under $\mathcal{F}$ has maximal possible recall, i.e., for any query X and any arbitrary mapping M:

$$\mathcal{R}[\mathcal{S}(X), X] \geq \mathcal{R}[M, X].$$

□

Proof **(if)** We first show that if $\mathcal{F}$ requires maximal recall, then disjunctions are always separable. We will prove by contradiction. Assuming that for some $\check{Q}$, $\mathcal{S}(\check{Q})$ cannot be computed by $\mathcal{S}(D_1) \vee \cdots \vee \mathcal{S}(D_n)$. Let us denote the former by S (*i.e.*, $\mathcal{S}(\check{Q})$ = S) and the latter by M (*i.e.*, $\mathcal{S}(D_1) \vee \cdots \vee \mathcal{S}(D_n)$ = M); note that $S \neq M$. Our assumption is thus $\mathcal{F}[S, \check{Q}] > \mathcal{F}[M, \check{Q}]$ (where we assume there is no tie between S and M, or otherwise one can simply choose $\mathcal{S}(\check{Q})$ = M). Because $\mathcal{F}$ is monotonic in $\mathcal{P}$ and $\mathcal{R}$, S must have (at least) higher recall or higher precision. (Otherwise $\mathcal{F}[S, \check{Q}]$ cannot be higher than $\mathcal{F}[M, \check{Q}]$.)

We argue that S *cannot have higher recall*: Assume that $\mathcal{R}[S, \check{Q}] > \mathcal{R}[M, \check{Q}]$, *i.e.*, $|S \wedge \check{Q}| > |M \wedge \check{Q}|$. There thus exists an answer a for $\check{Q}$, which is retrieved by S but not by M. Since a matches $\check{Q} = D_1 \vee \cdots \vee D_n$, it must match some D_i. However, since a is not in M, it was not retrieved by $\mathcal{S}(D_i)$. Thus, $\mathcal{S}(D_i)$ cannot have maximal recall, a contradiction to what $\mathcal{F}$ requires: The mapping $\mathcal{S}(D_i) \vee S$ (which is a valid mapping based on our Boolean closure assumption) will have higher recall, since it retrieves entire $\mathcal{S}(D_i)$ plus at least a.

Note that while S cannot have higher recall, $\mathcal{F}$ guarantees that S has maximal recall, at least equal to that of M. That is, $\mathcal{R}[S, \check{Q}] = \mathcal{R}[M, \check{Q}]$, or $|S \wedge \check{Q}| = |M \wedge \check{Q}|$.

Thus S has higher precision; *i.e.*, $\mathcal{P}[S, \check{Q}] > \mathcal{P}[M, \check{Q}]$ or $|S \wedge \check{Q}|/|S| > |M \wedge \check{Q}|/|M|$. It follows from $|S \wedge \check{Q}| = |M \wedge \check{Q}|$, as just obtained, that $|S| < |M|$.

Because $|S| < |M|$, it must follow that $M \not\subseteq S$. Since $M = \mathcal{S}(D_1) \vee \cdots \vee \mathcal{S}(D_n)$, there must exist some D_i such that $\mathcal{S}(D_i) \not\subseteq S$. Let us now focus on this D_i.

We will show that $\mathcal{S}(D_i)$ cannot be the closest mapping for D_i, a contradiction. To see why, note that by supporting both $\mathcal{S}(D_i)$ and S, the target must also support $\mathcal{S}(D_i) \wedge S$ (by Assumption 1). We show that, for D_i, $(\mathcal{S}(D_i) \wedge S)$ is a closer mapping than $\mathcal{S}(D_i)$ by the monotonicity of $\mathcal{F}(\mathcal{P}, \mathcal{R})$.

- $(\mathcal{S}(D_i) \wedge S)$ and $\mathcal{S}(D_i)$ have equal recall: Assume their recall values are different; since $\mathcal{F}$ requires $\mathcal{S}(D_i)$ to have maximal recall, $\mathcal{S}(D_i)$ must have higher recall. There thus exists an answer a for D_i, which is retrieved by $\mathcal{S}(D_i)$ but not by S, although such a must also match $\check{Q}$. Thus S cannot have maximal recall for $\check{Q}$, a contradiction, since $\mathcal{S}(D_i) \vee S$ will have higher recall by retrieving a.
- $(\mathcal{S}(D_i) \wedge S)$ has higher precision than $\mathcal{S}(D_i)$ does: Since $\mathcal{S}(D_i) \not\subseteq S$, it follows that $\mathcal{S}(D_i) \wedge S \subset \mathcal{S}(D_i)$ and thus $|\mathcal{S}(D_i) \wedge S| < |\mathcal{S}(D_i)|$. Therefore $(\mathcal{S}(D_i) \wedge S)$ indeed removes some false-positives that were in $\mathcal{S}(D_i)$ but not in S. Formally, together with $(\mathcal{S}(D_i) \wedge S) \wedge D_i = \mathcal{S}(D_i) \wedge D_i$ (because they have the same recall, as shown above),

$$\begin{aligned} &\mathcal{P}[(\mathcal{S}(D_i) \wedge S), D_i] \\ &= \frac{|\mathcal{S}(D_i) \wedge S \wedge D_i|}{|\mathcal{S}(D_i) \wedge S|} > \frac{|\mathcal{S}(D_i) \wedge D_i|}{|\mathcal{S}(D_i)|} = \mathcal{P}[\mathcal{S}(D_i), D_i]. \end{aligned}$$

(only if) We show that, if $\mathcal{F}$ does not maximize recall, it is *possible* to construct a scenario with $\mathcal{F}$, in which a disjunction is not separable. Thus disjunctions are not *always* separable in every scenario that uses $\mathcal{F}$. That is, at least the identified scenario is a counter example (although obviously there can be other cases when disjunctions are separable with $\mathcal{F}$).

We construct the scenario as follows. In this example we represent a query Q with the set of answer objects that Q matches (as in Section 4 where a "calendar" query is represented with the months it covers). Assume a target system supports only the native queries $S_1, \ldots, S_n$, and T, as defined below, as well as their Boolean combinations (according to Assumption 1). In these queries, we use integer parameters e, n, and k; they will be determined later.

$$\begin{aligned} S_i &= \{a_{i1}, \ldots, a_{ie}\} \\ T &= \{a_{11}, \ldots, a_{1e}, a_{1(e+1)}, \\ &\quad\; a_{21}, \ldots, a_{2e}, a_{2(e+1)}, \\ &\quad\; \ldots\ldots, \\ &\quad\; a_{n1}, \ldots, a_{ne}, a_{n(e+1)}\} \\ &= S_1 \vee \cdots \vee S_n \vee \{a_{1(e+1)}, \ldots, a_{n(e+1)}\} \end{aligned}$$

Suppose that the mediator supports queries $D_1, \ldots, D_n$, defined as follows, where $k > e$ (and thus $D_i \supset S_i$).

$$D_i = \{a_{i1}, \ldots, a_{ik}\}$$

Let's translate each D_i to find $\mathcal{S}(D_i)$. Consider the potential choices S_i and T. The respective precisions and recalls are as follows:

- $S_i : P_S = \mathcal{P}[S_i, D_i] = 1, R_S = \mathcal{R}[S_i, D_i] = \frac{e}{k}$
- $T : P_T = \mathcal{P}[T, D_i] = \frac{1}{n}, R_T = \mathcal{R}[T, D_i] = \frac{e+1}{k}$

To find $\mathcal{S}(D_i)$ for D_i, observe that S_i and T are the only comparable choices. In particular, S_j when $j \neq i$ is "irrelevant" to D_i; its $\mathcal{P}$ and $\mathcal{R}$ are both zero. While the source can also support any Boolean combinations of the above queries, it is obvious that none of them will make better mappings, since only S_i and T_i contain the relevant objects ($a_{i1}, \ldots, a_{ik}$) for D_i. For instance, the disjunction $(S_i \vee S_j)$ needs not be considered, as it will have the same recall but lower precision ($\mathcal{P} = .5$) as compared to S_i. Likewise, conjunctions like $S_i \wedge S_j = \phi$ and $S_i \wedge T = S_i$ will not results in better mappings.

For *any* $\mathcal{F}$ that does not maximize recall, we show that there exists a case (as characterized by parameters k, e, and n) in which $\mathcal{F}$ will determine that $\mathcal{S}(D_i) = S_i$, *i.e.*, $\mathcal{F}[S_i, D_i] > \mathcal{F}[T, D_i]$. Since $\mathcal{F}$ does not always maximize recall, there must exist (P_1, R_1) and (P_2, R_2), where $R_1 < R_2$, such that

$$\mathcal{F}(P_1, R_1) > \mathcal{F}(P_2, R_2).$$

Our proof will use the following statement, which orders the $\mathcal{P}$ and $\mathcal{R}$ parameters. We give a proof separately.

a. Given (P_1, R_1) and (P_2, R_2) as above, there exist integer assignments of k, e, n, such that $R_1 < R_S < R_T < R_2$, $P_T < P_2$, and $P_1 \leq P_S$.

With the ordering of closeness parameters that Statement (a) asserts to exist, S_i will be a better mapping for D_i. The monotonicity of $\mathcal{F}$ implies the following ordering of closeness: First, $\mathcal{F}(P_S, R_S) > \mathcal{F}(P_1, R_1)$ because $R_S > R_1$ and $P_S \geq P_1$. Second, $\mathcal{F}(P_T, R_T) < \mathcal{F}(P_2, R_2)$ because $R_T < R_2$ and $P_T < P_2$. Consequently, we conclude that $\mathcal{F}(P_S, R_S) > \mathcal{F}(P_T, R_T)$, *i.e.*,

$$\forall i : \mathcal{S}(D_i) = S_i.$$

We next show that $\check{Q} = D_1 \vee \cdots \vee D_n$ is not separable, *i.e.*, $\mathcal{S}(\check{Q}) \neq \mathcal{S}(D_1) \vee \cdots \vee \mathcal{S}(D_n)$ or $\mathcal{S}(\check{Q}) \neq S_1 \vee \cdots \vee S_n$. Denoting $S = S_1 \vee \cdots \vee S_n$, we show that T will be a better mapping than S for $\check{Q}$, and thus $\mathcal{S}(\check{Q}) \neq S$, based on the following analysis of $\mathcal{P}$ and $\mathcal{R}$:

- $S = S_1 \vee \cdots \vee S_n$: $\mathcal{P}[S, \check{Q}] = 1$, $\mathcal{R}[S, \check{Q}] = \frac{e}{k}$
- T: $\mathcal{P}[T, \check{Q}] = 1$, $\mathcal{R}[T, \check{Q}] = \frac{e+1}{k}$

Therefore, $\mathcal{F}$ must determine that T is a closer mapping than S for $\check{Q}$, because T has higher recall while both have the same precision. Since $\mathcal{F}$ is monotonic,

$$\mathcal{F}(\mathcal{P}[T, \check{Q}], \mathcal{R}[T, \check{Q}]) > \mathcal{F}(\mathcal{P}[S, \check{Q}], \mathcal{R}[S, \check{Q}]).$$

We thus conclude $\mathcal{S}(\check{Q}) \neq S$, *i.e.*, $\mathcal{S}(\check{Q}) \neq \mathcal{S}(D_1) \vee \cdots \vee \mathcal{S}(D_n)$, *i.e.*, $\check{Q}$ is not separable.

(a) First, we show that $R_1 < R_S < R_T < R_2$. Since $R_1 < R_2$ as given, there exists a real number u between them, *i.e.*, $R_1 < u < R_2$. Assign k such that $\frac{1}{k} < R_2 - u$ (which is possible since k can be arbitrarily large), *i.e.*,

$$u + \frac{1}{k} < R_2 \tag{19}$$

We then assign e such that $\frac{e}{k} = u$ or $e = uk$. Since we derived that $R_S = \frac{e}{k}$ and that $R_T = \frac{e+1}{k}$, both can be written in terms of u as

$$R_S = u \tag{20}$$

$$R_T = u + \frac{1}{k} \tag{21}$$

Consequently, we derive $R_1 < R_S < R_T < R_2$: Since $R_1 < u < R_2$, by Eq. 20, $R_1 < R_S < R_2$. It follows from Eq. 20 and 21 that $R_S < R_T$. Lastly, $R_T < R_2$ by Eq. 19.

Second, we show that $P_T < P_2$. Given P_2, since n can be arbitrarily large, we can assign n such that $\frac{1}{n} < P_2$, and thus $P_T < P_2$.

Finally, it is immediate that $P_1 \leq P_S$, because $P_S = 1$, the maximal possible recall. □

Theorem 2 (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 under $\mathcal{F}$ has maximal possible precision, i.e., for any query X and any arbitrary mapping M:

$$\mathcal{P}[\mathcal{S}(X), X] \geq \mathcal{P}[M, X].$$

□

Proof The proof parallels that of Theorem 1. We hence omit the details here. □

C 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 Theorems 1 and 2 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 query $\check{Q}_a = D_1{:}th \vee D_2{:}tc$ (a subquery of Q_{med}). As we will see with Theorem 6, 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 6.1). In particular, $\check{Q}_a$ has matchings m_t, m_h, m_c, m_{th}, and m_{tc} (see Figure 6).

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 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 the preceding example, consider $\check{Q}_b = D_1{:}tv \vee D_2{:}td$ (another subquery of Q_{med}). As we will see, Theorem 6 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 6) 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 6) is the closest to $\check{Q}_b$. To verify, we can write $\check{Q}_b = t(v \vee d)$. As given by rule R_5, $\overline{m_t}$ best translates t, and similarly $\overline{m_{vd}}$ for $v \vee d$ (by rule R_2). Note that the two translations can be separately done, since fn constraints are independent of format ones (in particular, they are not jointly involved in any semantic units that the rules define). Thus M_1 based on B_1 is the best mapping. □

Observe that $\check{Q}_a$ (of Example 14) contrasts with $\check{Q}_b$ (of Example 15) 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 6 formally states the safety conditions for disjunctions, which Example 16 then illustrates. We give a proof in the next section.

Theorem 6 (Disjunction Safety) *Let disjunction $\check{Q} = D_1 \vee \cdots \vee D_n$, where there are no disjunction redundancies. Let $CNF(D_i) = \check{C}_{i1} \wedge \cdots \wedge \check{C}_{im_i}$. $\check{Q}$ 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 6 by showing that $\check{Q}_b$ (in Example 15) is not safe (*i.e.*, it will fail the conditions). We start by writing $CNF(D_1) = (t)(v)$ and $CNF(D_2) = (t)(d)$. 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 $\check{C}_4 = (v) \vee (d)$. (Observe that these $\check{C}_i$'s represent all the *disjunction factors* in $\check{Q}_b$.) 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 7 formally states the dual safety condition for conjunctions, which Example 17 will then illustrate.

Theorem 7 (Conjunction Safety) *Let conjunction $\hat{Q}$ = $C_1 \wedge \cdots \wedge C_n$, where there are no conjunction redundancies. Let* 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 7, 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 6), they share the same matchings. As Theorem 7 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 *entirely* from either C_1 or C_2. For instance, consider matching $m_{th} = th$, we find that $m_{th} \supseteq (th)(v)$ and also $m_{th} \supseteq (th)$. Consequently, Theorem 7 will determine that $\hat{Q}$ is safe. □

Apparently, Theorems 6 and 7 seem to disregard the specific closeness criteria. However, what particular semantic units can match a 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 6.1 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 $\mathcal{F}=$ RThresh$(.7)$, which K_{med} is based upon. In other words, for a different metric such as $\mathcal{F}' =$ 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.1 Proof of the Safety Theorems

Theorem 6 (Disjunction Safety)

Proof Suppose that $\check{Q}$ has matchings $m_1, \ldots, m_u$ with respect to K (a sound and complete mapping specification). There must exist a decomposition $B(m_1, \ldots, m_u)$ of $\check{Q}$ such that $\mathcal{S}(\check{Q})$ = $B(\overline{m_1}, \ldots, \overline{m_u})$: When $\check{Q}$ is not itself a matching, such a decomposition exists since K is complete. Otherwise, if $\check{Q}$ itself is a matching m, then $B = m$, and thus $\mathcal{S}(\check{Q})$ = $\overline{m}$ since K is sound.

As Lemma 1 below shows, B represents a decomposition along the disjuncts, *i.e.*, B can be written in terms of expressions that correspond to D_i: B = $B_1(m_1, \ldots, m_u) \vee \cdots \vee B_n(m_1, \ldots, m_u)$, where $B_i \equiv D_i$. Consequently, because $\mathcal{S}(\check{Q}) = B(\overline{m_1}, \ldots, \overline{m_u})$, it follows that

$$\mathcal{S}(\check{Q}) = B_1(\overline{m_1}, \ldots, \overline{m_u}) \vee \cdots \vee B_n(\overline{m_1}, \ldots, \overline{m_u}).$$

Therefore $\mathcal{S}(\check{Q})$ *can* be obtained by separately mapping the disjuncts, *i.e.*, $\mathcal{S}(\check{Q}) = M_1 \vee \cdots \vee M_n$, where M_i is a mapping of D_i constructed as $B_i(\overline{m_1}, \ldots, \overline{m_u})$. Further by Assumption 2, $\mathcal{S}(\check{Q})$ can be constructed when each M_i best approximates D_i, *i.e.*,

$$\mathcal{S}(D_i) = B_i(\overline{m_1}, \ldots, \overline{m_u}).$$

That is, $\mathcal{S}(\check{Q}) = \mathcal{S}(D_1) \vee \cdots \vee \mathcal{S}(D_n)$, and thus $\check{Q}$ is separable. □

Lemma 1 *Let $\check{Q}$ be a safe disjunction with respect to semantic units $m_1, \ldots, m_u$, as Theorem 6 specifies. Let $B(m_1, \ldots, m_u)$ be a rewriting of $\check{Q}$ in terms of the units. B must implicitly separate the disjuncts of $\check{Q}$, i.e., $B = B_1 \vee \cdots \vee B_n$ such that $B_i(m_1, \ldots, m_u) \equiv D_i$.* □

Proof As a remark, we note that this lemma formally states the "implicit separation" illustrated in Example 14. Our proof uses the following statements (which we will show later).

a. Let $\hat{X} = \prod m$ be a simple conjunction of some matchings m in $\{m_1, \ldots, m_u\}$. If $\hat{X} \subseteq \check{Q}$, then $\hat{X} \subseteq D_i$ for some D_i.
b. Let B_i be a function of $m_1, \ldots, m_u$. Assume that there is no negation in queries (as this paper assumes). If $B_1 \vee \cdots \vee B_n = D_1 \vee \cdots \vee D_n$ and $\forall i : B_i \subseteq D_i$, then $\forall i : B_i = D_i$.

Let us write $B(m_1, \ldots, m_u)$ in DNF as

$$\textit{DNF}(B) = \sum \hat{X}_k(m_1, \ldots, m_u), \tag{22}$$

where each $\hat{X}_k$ is a simple conjunction of some m_i. Note that because $B \equiv \check{Q}$ and $\hat{X}_k \subseteq B$, it follows that $\hat{X}_k \subseteq \check{Q}$. Consequently, by Statement (a), $\hat{X}_k \subseteq D_i$ for some D_i.

Based on Eq. 22 we now rewrite $B = B_1 \vee \cdots \vee B_n$; *i.e.*, we construct B_i to satisfy the lemma. Let B_i be the disjunction of all $\hat{X}_k$ such that $\hat{X}_k \subseteq D_i$, *i.e.*,

$$B_i(m_1, \ldots, m_u) = \sum_{\hat{X}_k \subseteq D_i} \hat{X}_k. \tag{23}$$

Note that every $\hat{X}_k$ in Eq. 22 must appear in some B_i, because $\hat{X}_k \subseteq D_i$ for some D_i. Consequently, we can rewrite the disjunction of all $\hat{X}_k$ (Eq. 22) as that of all B_i (Eq. 23). That is, Eq. 22 becomes $B \equiv B_1 \vee \cdots \vee B_n$. Meanwhile, note that by construction $B_i \subseteq D_i$.

Since B is a rewriting of $\check{Q}$, $B \equiv \check{Q}$ or $B_1 \vee \cdots \vee B_n \equiv D_1 \vee \cdots \vee D_n$. Furthermore, for every B_i, $B_i \subseteq D_i$ as we constructed above. Consequently, by Statement (b), $B_i = D_i$, *i.e.*, B_i represents D_i in the decomposition. Thus this lemma is proven; we next give a proof of the statements.

(a) Suppose that $\hat{X} \subseteq \check{Q}$ and, $\forall i \in [1:n-1]$, $\hat{X} \not\subseteq D_i$ (otherwise, clearly $\hat{X} \subseteq D_i$ for some D_i). We will show that $\hat{X} \subseteq D_n$ and thus the statement holds. Let $\mathit{CNF}(D_i) = \check{C}_{i1} \wedge \cdots \wedge \check{C}_{im_i}$, where each $\check{C}_{ij}$ is a simple disjunction. We first prove two auxiliary statements.

– (a1) $\forall i \in [1:n-1]$, $\exists\, \check{C}_{ij}$, *such that for every* m *in* $\hat{X}$, $m \not\subseteq \check{C}_{ij}$.
  Given that $\hat{X} \not\subseteq D_i$, it follows that $\hat{X} \not\subseteq \check{C}_{ij}$ for some $\check{C}_{ij}$ in $\mathit{CNF}(D_i)$. For such $\check{C}_{ij}$, since $\hat{X} = \prod m$, it must hold that, for every m in $\hat{X}$, $m \not\subseteq \check{C}_{ij}$.
– (a2) *Let* $\check{C} = \check{C}_{1k_1} \vee \cdots \vee \check{C}_{nk_n}$, *i.e., a combination of one* $\check{C}_{ik_i}$ *from each* $\mathit{CNF}(D_i)$. *For every such* $\check{C}$, *there exists some* m *in* $\hat{X}$, *such that* $m \subseteq \check{C}$.
  Because $\check{C}_{ik_i}$ is a conjunct in $\mathit{CNF}(D_i)$, it follows that $\forall i : D_i \subseteq \check{C}_{ik_i}$. Consequently, $(D_1 \vee \cdots \vee D_n) \subseteq (\check{C}_{1k_1} \vee \cdots \vee \check{C}_{nk_n})$ or $\check{Q} \subseteq \check{C}$. Given that $\hat{X} \subseteq \check{Q}$, we obtain $\hat{X} \subseteq \check{C}$ or $\prod m \subseteq \check{C}$.
  Note that $\check{C}$ is a simple disjunction of constraints (which we do not interpret since they are atomic), because every $\check{C}_{ik_i}$ is. Since $\check{C}$ contains no conjunctions, for $\prod m \subseteq \check{C}$ to hold, there must exist some m in $\hat{X}$ such that $m \subseteq \check{C}$.

Based on the auxiliary statements, we now show that $\hat{X} \subseteq D_n$. Let $\mathit{CNF}(D_n) = \check{C}_{n1} \vee \cdots \vee \check{C}_{nm_n}$. For every $\check{C}_{nk_n}$ (where $k_n \in [1:m_n]$), we will show below that $m \subseteq \check{C}_{nk_n}$ for some m in $\hat{X} = \prod m$. It then follows from $\hat{X} \subseteq m$ that $\hat{X} \subseteq \check{C}_{nk_n}$. Consequently, $\hat{X} \subseteq (\check{C}_{n1} \vee \cdots \vee \check{C}_{nm_n})$ or $\hat{X} \subseteq D_n$.

We next show that $m \subseteq \check{C}_{nk_n}$ for some m in $\hat{X}$. For any $\check{C}_{nk_n}$, let us consider the combination $\check{C} = \check{C}_{1k_1} \vee \cdots \vee \check{C}_{(n-1)k_{n-1}} \vee \check{C}_{nk_n}$, such that $\forall i \in [1:n-1]$, $m \not\subseteq \check{C}_{ik_i}$ for every m in $\hat{X}$. Statement (a1) asserts that such $\check{C}_{ik_i}$ exists.

Since the lemma requires that $\check{Q}$ be as Theorem 6 specifies, $\check{C}$ must satisfy either Condition (1) or (2) of the theorem. For either case, we show that $m \subseteq \check{C}_{nk_n}$ for some m in $\hat{X}$:

– if $\check{C}$ satisfies Condition (1): By Statement (a2), for some m in $\hat{X}$, $m \subseteq \check{C}$. Furthermore, by Condition (1), there exists some $\check{C}_{ik_i}$ such that $m \subseteq \check{C}_{ik_i}$. Because $m \not\subseteq \check{C}_{ik_i}, \forall i \in [1:n-1]$, it must hold that $m \subseteq \check{C}_{nk_n}$.
– if $\check{C}$ satisfies Condition (2): That is, there exists another combination $\check{C}' = \check{C}_{1h_1} \vee \cdots \vee \check{C}_{nh_n}$ that satisfies Condition (1). By Statement (a2), for some m in $\hat{X}$, $m \subseteq \check{C}'$– Consequently, Condition (1) (which $\check{C}'$ satisfies) asserts that $m \subseteq \check{C}_{jh_j}$ for some $\check{C}_{jh_j}$.
  As Condition (2) (which $\check{C}$ satisfies) requires, there exists some $\check{C}_{ik_i}$ such that $\check{C}_{jh_j} \subseteq \check{C}_{ik_i}$ and thus $m \subseteq \check{C}_{ik_i}$. Since $m \not\subseteq \check{C}_{ik_i}$ for $i \in [1:n-1]$, it follows that $m \subseteq \check{C}_{nk_n}$.

(b) We will prove by contradiction. Assume that, without loss of generality, $B_1 \neq D_1$ (we can similarly prove for any other B_i). Given that $B_i \subseteq D_i$, this assumption means that $B_1 \subset D_1$ and thus $D_1(\neg B_1) \neq \phi$. We will derive a contradiction that $\check{Q}$ must have disjunction redundancy, unlike what Theorem 6 specifies.

We derive that $D_1(\neg B_1) \subseteq (D_2 \vee \cdots \vee D_n)$ as follows: $D_1(\neg B_1) \subseteq D_1 \subseteq (D_1 \vee \cdots \vee D_n) = (B_1 \vee \cdots \vee B_n)$. Because $D_1(\neg B_1) \not\subseteq B_1$, it follows that $D_1(\neg B_1) \subseteq (B_2 \vee \cdots \vee B_n)$. Since $B_i \subseteq D_i$, we thus obtain

$$D_1(\neg B_1) \subseteq (D_2 \vee \cdots \vee D_n). \tag{24}$$

Suppose that $\mathit{DNF}(D_i) = \sum_j d_{ij}$. Suppose also that $\mathit{CNF}(B_1) = \prod_k b_k$, where $b_k = \sum_l c_{kl}$ is a simple disjunction of constraints c_{kl}. Using DeMorgan's laws, we obtain that $\neg B_1 = \sum_k \neg b_k = \sum_k(\prod_l \neg c_{kl})$. In terms of these normal forms, Eq. 24 becomes

$$\sum_j d_{1j} \sum_k(\prod_l \neg c_{kl}) \subseteq \sum_j d_{2j} \vee \cdots \vee \sum_j d_{nj}, i.e.,$$
$$\sum_{j,k} d_{1j}(\prod_l \neg c_{kl}) \subseteq \sum_j d_{2j} \vee \cdots \vee \sum_j d_{nj}.$$

Since the left side is not trivially false (*i.e.*, $D_1(\neg B_1) \neq \phi$), there exist some j and k such that $d_{1j}(\prod_l \neg c_{kl}) \neq \phi$. Let us focus on this particular term. Obviously $d_{1j}(\prod_l \neg c_{kl}) \subseteq \sum_j d_{2j} \vee \cdots \vee \sum_j d_{nj}$. Note that the left side is simply a conjunction of (positive or negative) constraints; in particular, there is no disjunction. (Also note that we do not interpret constraints; they are "atomic.") Consequently there exists some d_{uv} in the right side such that $d_{1j}(\prod_l \neg c_{kl}) \subseteq d_{uv}$. Since d_{uv} is a simple conjunction of only *positive* constraints (we assume there are no negations), it follows that $\forall l : \neg c_{kl} \not\subseteq d_{uv}$ and thus $d_{1j} \subseteq d_{uv}$. In other words, d_{1j} in $\mathit{DNF}(D_1)$ is redundant with respect to d_{uv} in $\mathit{DNF}(D_u)$, *i.e.*, $\check{Q}$ has disjunction redundancy, which is a contradiction. □

Theorem 7 (Conjunction Safety)

Proof The proof parallels that of Theorem 6. We hence omit the details here. □

Lemma 2 *Let* $\hat{Q}$ *be a safe conjunction with respect to semantic units* $m_1, \ldots, m_u$, *as Theorem 7 specifies. Let* $B(m_1, \ldots, m_u)$ *be a rewriting of* $\hat{Q}$ *in terms of the units.* B *must implicitly separate the conjuncts of* $\hat{Q}$, *i.e.,* $B = B_1 \wedge \cdots \wedge B_n$ *such that* $B_i(m_1, \ldots, m_u) \equiv C_i$. □

Proof The proof parallels that of Lemma 1. □

D Unique Safe Decomposition

Theorem 3 (*SafeConj* Correctness)
Given a query Q *and the associated semantic units,*

– (soundness) *SafeConj will rewrite* Q *into a safe conjunction, and*
– (liveness) *if there exists a non-trivial safe conjunction for* Q, *then the conjunction that SafeConj returns is also non-trivial.*

□

Proof **(soundness)** Suppose that, given query Q, *SafeConj* returns $Q \equiv (C_1 \wedge \cdots \wedge C_n)$. We show that this conjunction is safe by Theorem 7.

First, Theorem 7 requires that there be no conjunct redundancies. By grouping conjuncts with redundant factors, *SafeConj* in Step (3) ensures that $C_1, \ldots, C_n$ have no redundancies.

Second, Theorem 7 requires that the conjunctions of the DNF disjuncts of C_i satisfy Condition (1) or (2). For this test, instead of C_i, we will focus on the *essential candidate conjuncts* $\check{E}_1, \ldots, \check{E}_e$ formulated in Step (2). It is straightforward to show that if these candidate conjuncts can satisfy the test then the merged *final conjuncts* C_i in Step (3) will too. For instance, if $(\check{E}_1) \wedge (\check{E}_2) \wedge (\check{E}_3)$ can pass the test, so does $(\check{E}_1\check{E}_2) \wedge (\check{E}_3)$.

To apply the test, we need every $\check{E}_j$ in DNF. Note that *SafeConj* already formulates $\check{E}_j$ in DNF as $\hat{x}_{j1} \vee \cdots \vee \hat{x}_{jm}$. Since each $\hat{x}_{jl}$ represents a simple conjunction in *DNF*(m_i) for some matching m_i, let's refer to each $\hat{x}_{jl}$ as a *cunit* (*i.e.*, *conjunction unit*).

To apply Theorem 7, we test every combination $\hat{D} = (\hat{x}_{1u_1}) \cdots (\hat{x}_{eu_e})$, where each $\check{E}_j$ contributes one cunit $\hat{x}_{ju_j}$. We show that $\hat{D} \subseteq \hat{D}_l$, *in terms of the cunits*, for some $\hat{D}_l$ in *DNF*(Q). (That is, we interpret $\hat{D}$ and $\hat{D}_l$ as functions of the cunits, without considering the constraints within.) Since $\hat{x}_{ju_j} \subseteq \check{E}_j, \forall j \in [1:e]$, it follows that $(\hat{x}_{1u_1}) \cdots (\hat{x}_{eu_e}) \subseteq \check{E}_1 \wedge \cdots \wedge \check{E}_e$. In the right side, since each $\check{E}_j$ combines one cunit from every $\hat{D}_l$ (which is written in terms of the cunits in Step (1) of *SafeConj*) in *DNF*(Q), their conjunction will recover to *DNF*(Q). The above subsumption thus becomes

$$\hat{D} \subseteq \hat{D}_1 \vee \cdots \vee \hat{D}_m$$

Note the subsumption is in terms of the cunits. The left side $\hat{D} = (\hat{x}_{1u_1}) \cdots (\hat{x}_{eu_e})$ is a simple conjunction of some cunits. In the right side, every $\hat{D}_l$ is written as a conjunction of the cunits by Step (1). Since $\hat{D}$ contains no disjunctions, the above subsumption requires that $\hat{D}$ be subsumed by some disjunct, *i.e.*, $\exists l : \hat{D} \subseteq \hat{D}_l$ in terms of the cunits.

That is, it must be either $\hat{D} = \hat{D}_l$ or $\hat{D} \subset \hat{D}_l$. We show that either case will respectively satisfy Condition (1) or (2) of Theorem 7, and thus *SafeConj* returns safe conjunctions.

(1) $\hat{D} = \hat{D}_l$: That is, $\hat{D} = (\hat{x}_{1u_1}) \cdots (\hat{x}_{eu_e})$ equals some $\hat{D}_l$ in terms of the cunits, as rewritten by Step (1). *SafeConj* ensures that such a conjunction will satisfy Condition (1) of Theorem 7: Suppose that m is a matching such that $m \supseteq \hat{D}_l$. Because $\hat{D}_l$ is a simple conjunction, there exists some cunit $\hat{x}$ (which is also a simple conjunction) in *DNF*(m) such that $\hat{x} \supseteq \hat{D}_l$. Since $\hat{x}$ must be a subconjunction of $\hat{D}_l$, referring to Step (1), $\hat{x}$ must either itself be an included $\hat{x}_{lk}$ (*i.e.*, $\hat{x} = \hat{x}_{lk}$), or be a subconjunction of it (*i.e.*, $\hat{x} \supseteq \hat{x}_{lk}$). In either case, because $m \supseteq \hat{x}$, it follows that $m \supseteq \hat{x}_{lk}$, and thus Condition (1) of Theorem 7 holds.

(2) $\hat{D} \subset \hat{D}_l$: Note that both sides are simple conjunctions of some cunits, which the subsumption is based on. In other words, $\hat{D}_l$ is a subconjunction of $\hat{D}$ in terms of the cunits: *i.e.*, $\forall \hat{x}_{lk}$ (a cunit) in $\hat{D}_l$, $\exists \hat{x}$ in $\hat{D}$, such that $\hat{x}_{lk} = \hat{x}$ (there may be extra cunits in $\hat{D}$ not appearing in $\hat{D}_l$). Because $\hat{D}_l$ satisfies Condition (1) as explained above, $\hat{D}$ will satisfy Condition (2) of Theorem 7.

(liveness:) Assume that, for Q, there exists a conjunction $Q = T_1 \wedge \cdots \wedge T_n$, which is non-trivial, *i.e.*, $n > 1$, and is safe (according to Theorem 7).

SafeConj will start with the DNF of Q; suppose that *DNF*$(Q) = \hat{D}_1 \vee \cdots \vee \hat{D}_m$. Note that *DNF*$(Q)$ can be derived based on *DNF*(T_i): Let *DNF*$(T_i) = \hat{D}_{i1} \vee \cdots \vee \hat{D}_{im_i}$. We will show separately later the following statement that every $\hat{D}_l$ is a combination of $\hat{D}_{ik_i}$'s, *i.e.*,

a. $\forall \hat{D}_l, \exists \hat{D}_{ik_i}$ in each T_i, such that $\hat{D}_l = \hat{D}_{1k_1} \cdots \hat{D}_{nk_n}$.

Each $\hat{D}_l$ thus consists of a portion from each T_i; let's denote the T_i portion of $\hat{D}_l$ by $\hat{D}_l(T_i)$, *i.e.*,

$$\hat{D}_l(T_i) = \hat{D}_{ik_i}.$$

SafeConj in Step (1) will rewrite each $\hat{D}_l$ in terms of the cunits. Suppose that it rewrites $\hat{D}_l = \prod \hat{x}_{lk}$, where $\hat{x}_{lk}$ is a cunit from some matching. For the proof, we assume that the algorithm includes *all* the cunits appearing in $\hat{D}_l$, *i.e.*, without omitting those subconjunctions in Step (1). These subconjunction cunit will simply create redundant conjuncts in Step (2), which will be "invisible" anyway after conjunct merging in Step (3), as we will discuss later. (That is, *SafeConj* omits these redundant cunits simply as an "optimization.")

The safety of $T_1 \cdots T_n$ (by Theorem 7) assures that every cunit in $\hat{D}_l$ appears *within* (*i.e.*, as a subconjunction of) some T_i portion (see later for a proof):

b. $\forall \hat{x}_{lk}$ in $\hat{D}_l$, $\exists T_i$, such that $\hat{x}_{lk} \supseteq \hat{D}_l(T_i)$.

Step (2) will generate *candidate conjuncts* of the form $\check{E}_j = \hat{x}_{1k_1} \vee \cdots \vee \hat{x}_{mk_m}$, which contains one $\hat{x}_{lk_l}$ from each $\hat{D}_l$. Our proof assumes that Step (2) does not remove redundant conjuncts, which can result from, among others, not omitting redundant cunits (as just mentioned). A redundant conjunct will become "invisible" after conjunct grouping (which we will construct by Eq. 25 later), and thus effectively be omitted as they are in *SafeConj*. Note that if $\check{E}$ is redundant, *every* factor X in *CNF*$(\check{E})$ subsumes some factor Y in another conjunct $\check{E}'$, *i.e.*, $X \supseteq Y$.

Every such X must be invisible, and thus $\check{E}$ will be effectively omitted. If $\check{E}$ is grouped with $\check{E}'$, X will be invisible since $XY = Y$. If they are in different groups, since we can show that conjuncts constructed from different groups have no redundancies (see Statement (c) later), X cannot be "visible" as a factor in the overall CNF of the group, or otherwise redundancies must exist between the groups of E and E'.

To prove that *SafeConj* will form a non-trivial conjunction, we must show that there *exists* a non-trivial grouping with the required properties. When a valid grouping exists, Step (3) can (at least) find it to return a non-trivial conjunction. We will construct such a grouping to show that it exists: This grouping, defined below, consisting of groups $\mathcal{E}_1, \ldots, \mathcal{E}_n$, which correspond to the safe conjuncts $T_1, \cdots, T_n$.

$$\mathcal{E}_i = \{\check{E} = \hat{x}_{1k_1} \vee \cdots \vee \hat{x}_{mk_m} \,|\, \forall l : \hat{x}_{lk_l} \supseteq \hat{D}_l(T_i)\} \quad (25)$$

It is instructive to note that each group $\mathcal{E}_i$ collects the candidate conjuncts that generate T_i:

c. Let C_i be the conjunction of all the candidate conjuncts in $\mathcal{E}_i$, *i.e.*, $C_i = \prod_{\check{E} \in \mathcal{E}_i} \check{E}$. $C_i = T_i$.
(As a lemma, since $T_1 \cdots T_n$ has no conjunction redundancies, neither does $C_1 \cdots C_n$.)

We show that the grouping of $\mathcal{E}_1, \ldots, \mathcal{E}_n$ has the properties that *SafeConj* requires (and thus the grouping is a solution that the algorithm can at least use in its Step (3)):

d. The grouping covers all *essential candidate conjuncts* in $\mathcal{E}=\{\check{E}_1, \ldots, \check{E}_e\}$: *i.e.*, $\forall \check{E} \in \mathcal{E}, \exists \mathcal{E}_i$, *s.t.* $\check{E} \in \mathcal{E}_i$.
e. The grouping is disjoint in terms of the essential conjuncts.

Statement (d) shows that $\mathcal{E}_1, \ldots, \mathcal{E}_n$ is indeed a grouping of the essential conjuncts (as in *SafeConj*), since it does not miss any. (While it may contain redundant conjuncts, we explained that they are invisible after grouping and thus effectively omitted, as they are in *SafeConj*.) It is also a disjoint grouping, as Statement (e) shows.

Finally, the grouping is non-trivial, since $n > 1$ as we assume that $T_1, \ldots, T_n$ is non-trivial. The grouping will generate n final conjuncts $C_1, \ldots, C_n$, as given in Statement (c). Since such a non-trivial grouping exists, *SafeConj* can (at least) find it to output a non-trivial safe conjunction (although *SafeConj* may generate other non-trivial grouping, if any).

(a) We derive $\hat{D}_l$ of $\mathit{DNF}(Q)$ in terms of $\hat{D}_{ij}$ in $\mathit{DNF}(T_i)$ as follows, since $Q = T_1 \wedge \cdots \wedge T_n = \prod_{i=1}^{n} \mathit{DNF}(T_i)$,

$$Q = \prod_{i=1}^{n} \hat{D}_{i1} \vee \cdots \vee \hat{D}_{im_i} = \sum_{k_i \in [1:m_i]} \hat{D}_{1k_1} \cdots \hat{D}_{nk_n}. \quad (26)$$

Eq. 26 rewrites Q as a disjunction of simple conjunctions (of the constraints in Q): Since every $\hat{D}_{ik_i}$ is a simple conjunction, so is every disjunct $\hat{D}_{1k_1} \cdots \hat{D}_{nk_n}$. Consequently, Eq. 26 must contain all the (non-redundant) disjuncts $\hat{D}_l$ in the DNF of Q (although in addition Eq. 26 may also have some extra, redundant disjuncts). That is, $\forall \hat{D}_l$ in $\mathit{DNF}(Q)$, $\exists \hat{D}_{ik_i}$ in $\mathit{DNF}(T_i)$, such that $\hat{D}_l = \hat{D}_{1k_1} \cdots \hat{D}_{nk_n}$.

(b) We show that every cunit $\hat{x}_{lk}$ appearing in $\hat{D}_l$ must appear *within* some T_i portion. We showed in (a) that every $\hat{D}_l$ is of the form $\hat{D}_l = \hat{D}_{1k_1} \cdots \hat{D}_{nk_n}$. Because $Q = T_1 \cdots T_n$ is safe, $\hat{D}_l$ must satisfy Condition (1) of Theorem 7. Note that it cannot satisfy Condition (2) because otherwise $\hat{D}_l$ is redundant in $\mathit{DNF}(Q)$ and thus can be deleted.

Let m be the matching containing the cunit $\hat{x}_{lk}$, *i.e.*, $\hat{x}_{lk}$ is a simple conjunction in $\mathit{DNF}(m)$. Since $\hat{x}_{lk}$ appears (as a subconjunction) in $\hat{D}_l$, it follows that $\hat{x}_{lk} \supseteq \hat{D}_l$ and thus $m \supseteq \hat{D}_l$. By Condition (1) of Theorem 7, it must hold that, for some $\hat{D}_{ik_i}$, $m \supseteq \hat{D}_{ik_i}$. Since $\hat{D}_{ik_i}$ is a simple conjunction (with no disjunctions), there must exist some cunit $\hat{y}$ (as a simple conjunction) in $\mathit{DNF}(m)$, such that $\hat{y} \supseteq \hat{D}_{ik_i}$. It must hold that $\hat{x}_{lk} = \hat{y}$ and thus $\hat{x}_{lk} \supseteq \hat{D}_{ik_i}$. Otherwise, if $\hat{x}_{lk}$ and $\hat{y}$ are distinct, since $\hat{x}_{lk} \supseteq \hat{D}_l$ and $\hat{y} \supseteq \hat{D}_l$, they are both simpler than the *same* $\hat{D}_l$ term in $\mathit{DNF}(Q)$, a contradiction. (As Section 6 discussed, any matching of Q cannot have distinct disjuncts $\hat{x}_{lk}$ and $\hat{y}$ that are both simpler than the same $\hat{D}_l$ in $\mathit{DNF}(Q)$.)

(c) We show that the candidate conjuncts in group $\mathcal{E}_i$ will form a conjunction C_i equal to T_i. As in Eq. 25, a candidate conjunct $\check{E}$ in $\mathcal{E}_i$ selects from each $\hat{D}_l$ one cunit $\hat{x}_{lk_l}$ that appears within $\hat{D}_l(T_i)$. Note that, since we did not omit redundant cunits and redundant candidate conjuncts in forming the grouping (see above), every cunit in $\hat{D}_l(T_i)$ will appear in some $\check{E}$ in $\mathcal{E}_i$. Consequently, the conjunction of all $\check{E}$ in $\mathcal{E}_i$ will "recover" (in conjunctions) all the cunits selected from the same $\hat{D}_l$, *i.e.*,

$$\begin{aligned} C_i &= \wedge(\{\check{E} = \hat{x}_{1k_1} \vee \cdots \vee \hat{x}_{mk_m} \mid \forall l : \hat{x}_{lk_l} \supseteq \hat{D}_l(T_i)\}) \\ &= \sum_{l=1}^{m} \wedge(\{\hat{x}_{lk} \mid \hat{x}_{lk} \supseteq \hat{D}_l(T_i)\}). \end{aligned} \quad (27)$$

Each disjunct above, as the conjunction of all the cunits in $\hat{D}_l(T_i)$, will simply equal to $\hat{D}_l(T_i)$: Note that every cunit in $\hat{D}_l(T_i)$ is a subconjunction of it and thus cannot contain any extra constraints not in $\hat{D}_l(T_i)$. Since at least every *single* constraint is a cunit (because an atomic constraint must be a matching, as required by the rule completeness), every constraint in $\hat{D}_l(T_i)$ must be covered by some cunit. Consequently, we obtain that $\wedge(\{\hat{x}_{lk} \mid \hat{x}_{lk} \supseteq \hat{D}_l(T_i)\}) = \hat{D}_l(T_i)$, and thus

$$C_i = \sum_{l=1}^{m} \hat{D}_l(T_i).$$

Note that every $\hat{D}_l(T_i)$ represents a $\hat{D}_{ik}$ in $\mathit{DNF}(T_i) = \hat{D}_{i1} \vee \cdots \vee \hat{D}_{im_i}$. In addition, as derived in (a), every $\hat{D}_{ik}$ must appear as $\hat{D}_l(T_i)$ in some $\hat{D}_l$ (otherwise, $\hat{D}_{ik}$ is redundant in T_i). That is, C_i given above contains all and only those $\hat{D}_{ik}$, *i.e.*,

$$C_i = \sum_{k=1}^{m_i} \hat{D}_{ik} = T_i.$$

(d) We show that the grouping covers all essential conjuncts, *i.e.*, every $\check{E}$ not included must be redundant. Any candidate conjunct has the form $\check{E} = \hat{x}_{1k_1} \vee \cdots \vee \hat{x}_{mk_m}$. As Statement (b) asserts, for each $\hat{x}_{lk_l}$ selected from $\hat{D}_l = \hat{D}_{1k_1} \cdots \hat{D}_{nk_n}$, $\hat{x}_{lk_l}$ is part of some $\hat{D}_{ik_i}$, *i.e.*, $\hat{x}_{lk_l} \supseteq \hat{D}_{ik_i}$. That is, *every* $\hat{x}_{lk_l}$ in $\check{E}$ *touches* some $\hat{D}_{ik_i}$ of $\mathit{DNF}(T_i)$. (It is instructive to note that, if $T_1 \cdots T_n$ were not safe, some cunit may not be part of any $\hat{D}_{ik_i}$ and thus spans across multiple T_i portions. An essential conjunct containing such cunits will thus be missed in the grouping.)

Since $\check{E}$ must select a $\hat{x}_{lk_l}$ from *every* $\hat{D}_l$ (as a combination of $\hat{D}_{ik_i}$ from each T_i), we can then show (which we omit here) that the selection must "fully" touch *some* T_i, *i.e.*: $\forall \hat{D}_{ik_i}$ in $\mathit{DNF}(T_i)$, $\check{E}$ contains some $\hat{x}_{lk_l}$ such that $\hat{x}_{lk_l} \supseteq \hat{D}_{ik_i}$.

If $\check{E}$ touches *only* a particular T_i, *i.e.*, $\check{E}$ selects from the T_i portion for every $\hat{D}_l$, then $\check{E}$ is included in group $\mathcal{E}_i$, according to Eq. 25. Conversely, if $\check{E}$ is not in any $\mathcal{E}_i$, then it must touches some entire T_i (as explained above) and, in addition, part of some other T_j. Such $\check{E}$ cannot be essential, as it must subsume some $\check{E}'$ in $\mathcal{E}_i$ that corresponds to only the T_i portion of $\check{E}$. That is, for every factor X in $\mathit{CNF}(\check{E})$, there exists some factor Y in $\mathit{CNF}(\check{E}')$, such that $X \supseteq Y$. We thus conclude that every essential conjunct must be covered in some $\mathcal{E}_i$.

(e) We show that any two groups $\mathcal{E}_i$ and $\mathcal{E}_j$ must be disjoint in terms of the essential conjuncts. Let C_i and C_j be the conjunctions generated by $\mathcal{E}_i$ and $\mathcal{E}_j$ respectively (see Statement (c)). Suppose $\mathcal{E}_i$ and $\mathcal{E}_j$ share a common essential conjunct $\check{E}_r$. Since $\check{E}_r$ is non-redundant as required by Step (2) of *SafeConj*, *CNF*(C_i) constructed from $\mathcal{E}_i$ and *CNF*(C_j) from $\mathcal{E}_j$ will both contain the "unique" factor X identified in Step (2) that makes $\check{E}_r$ essential. Note that X must appear in both *CNF*(C_i) and *CNF*(C_j) since it does not subsume any other conjunction factors. Consequently, C_i and C_j contain a conjunction redundancy. Since $C_i = T_i$ and $C_j = T_j$ (as Statement (c) asserts), the redundancy occurs across T_i and T_j. Such a redundancy cannot exist for the safe conjunction $T_1 \cdots T_n$ by Theorem 7, a contradiction. □

Theorem 4 (*SafeDisj* Correctness)
Given a query Q and the associated semantic units,

– (soundness) *SafeDisj will rewrite Q into a safe disjunction, and*
– (liveness) *if there exists a non-trivial safe disjunction for Q, then the disjunction that SafeDisj returns is also non-trivial.*

□

Proof Since *SafeDisj* is a dual algorithm of *SafeConj*, the proof parallels that of Theorem 3. □

Theorem 5 (Unique Safe Decomposition)
Given a query Q and a mapping specification K w.r.t. a closeness criterion $\mathcal{F}$, if a safe decomposition exists:

– (uniqueness) *the safe decomposition is* unique *for Q, i.e., any other distinct decomposition of Q must be unsafe, and*
– (correctness) *Algorithm NFB will find the unique safe decomposition and output $\mathcal{S}(Q)$ w.r.t. $\mathcal{F}$.*

□

Proof **(uniqueness)** Suppose there exists a safe decomposition $\mathcal{A}(m_1, \ldots, m_u)$ for Q, in terms of the semantic units $m_1, \ldots, m_u$ that K defines. We show that any decomposition different from $\mathcal{A}$ is unsafe, and thus $\mathcal{A}$ is uniquely safe.

As a safe decomposition, $\mathcal{A}$ is a rewriting of Q in which every composition (conjunction or disjunction) is safe, by definition. To begin with, consider the root composition $\mathcal{A} = \mathcal{A}_1 \odot \cdots \odot \mathcal{A}_n$, where $\odot$ is either $\vee$ or $\wedge$.

Since the composition is safe, other decompositions must also agree on separating components A_i. That is, according to Lemma 1 if $\odot$ is a disjunction or otherwise Lemma 2, every decomposition B, as also a rewriting of Q using $m_1, \ldots, m_u$, must implicitly separate the components: $B(m_1, \ldots, m_u)$ = $B_1 \odot \cdots \odot B_n$ such that $\forall B_i$: $B_i(m_1, \ldots, m_u)$ = $\mathcal{A}_i(m_1, \ldots, m_u)$.

If $\mathcal{A}_i$ is not already a semantic unit, it will contain compositions, which must also be safe since the entire decomposition $\mathcal{A}$ is safe. We can similarly (as in the above "root" case) show that B_i must agree on the (root) composition of $\mathcal{A}_i$, both as rewritings using $m_1, \ldots, m_u$. Recursively repeating this process (*i.e.*, traversing $\mathcal{A}$'s query tree), we conclude that B must agree on *every composition* that $\mathcal{A}$ has.

Consequently, to be different from $\mathcal{A}$, decomposition B must handle some semantic units (at the leaves of $\mathcal{A}$'s query tree) differently. In other words, B must break a semantic unit m as $m \equiv y_1 \odot \cdots \odot y_l$, *i.e.*, a composition of more than one component y_i. Such a composition cannot be safe: Since m is a semantic unit defined by some rule in K, by the completeness of K, $\mathcal{S}(m)$ *cannot* be synthesized from any such non-trivial decompositions (otherwise the rule defining m should not be given). Thus, $\mathcal{S}(m) \neq \mathcal{S}(y_1) \odot \cdots \odot \mathcal{S}(y_l)$, *i.e.*, as a semantic unit, m must remain atomic in mapping, and the composition is therefore unsafe. It follows that a decomposition can be safe only when every m is preserved as an atomic unit. In other words, any decomposition B different from $\mathcal{A}$ must be unsafe, and thus $\mathcal{A}$ is uniquely safe.

(correctness) Let $\mathcal{A}$ be the safe decomposition as given in the above. We show that *NFB*, by calling *SafeDecom*, will find the safe decomposition; the fact that a safe decomposition will lead to $\mathcal{S}(Q)$ was shown in Section 6.2.

We show that, if $\mathcal{A}$ is not already an atomic unit, the initial call *SafeDecom*$(Q, m_1, \ldots, m_u)$ will rewrite Q into a non-trivial composition: Note that there exists a nontrivial safe composition (conjunctive or disjunctive) for Q, *i.e.*, $\mathcal{A}_1 \odot \cdots \odot \mathcal{A}_n$ (where $n > 1$) as discussed above. Since *SafeDecom* will call *SafeConj* and, if it does not make progress, then *SafeDisj*, at least one of them will return a non-trivial composition, according to the liveness properties in Theorem 3 and Theorem 4. Let this composition be $Q = N_1 \odot \cdots \odot N_m$.

Since $Q = N_1 \odot \cdots \odot N_m$ is a safe composition (by the soundness properties in Theorem 3 and 4), according to Lemma 1 and 2, $\mathcal{A}$ (as a rewriting of Q) must also separate these N_i, *i.e.*, $\mathcal{A} = \mathcal{A}'_1 \odot \cdots \odot \mathcal{A}'_m$ such that $\mathcal{A}'_i = N_i$.

Since this composition of $\mathcal{A}'_i$ (or N_i) is safe (and thus separable), the decomposition $\mathcal{A}$ represents separately decomposing each $\mathcal{A}'_i$. First, if $\mathcal{A}'_i$ is not a semantic unit, $\mathcal{A}'_i$ will contain compositions, which must also be safe since the entire decomposition $\mathcal{A}$ is safe. There thus exists a safe composition for N_i, since one exists for $\mathcal{A}'_i$ and $\mathcal{A}'_i = N_i$. Consequently, *SafeDecom* will continue to make progress to decompose N_i, similarly to the "root" composition above. Recursively, *SafeDecom* will decompose the components until they are semantic units (which we discuss next), and $\mathcal{A}$ will agree on the compositions in the constructed structure. Second, if $\mathcal{A}'_i$ is a semantic unit, *SafeDecom* will not further decompose it and will simply return the atomic unit.

Overall, *SafeDecom* will construct a decomposition that agrees with $\mathcal{A}$ on all its compositions as well as atomic units. That is, *NFB* will find the unique safe decomposition $\mathcal{A}$. □