関係データ交換フレームワークにおける
弱適切なポリシの導出アルゴリズムの実装と性能評価

2019SC017 石田潤　2019SC048 岡田哲典

指導教員：石原靖哲

1　はじめに

近年，急速な情報化社会の進行により企業間における取引が電子データ化し，ある構造を持つデータを異なるデータに変換するデータ交換の技術が必要とされている．しかし，データを交換する際には，プライバシやセキュリティを確保する必要がある．

通常のデータ交換フレームワークでは，参加者はソース側とターゲット側の 2 種類である．しかし，実際の状況では，それらが交換したデータを使用する他の参加者が存在する．本研究では，以上の 3 種類の参加者がデータのやり取りを行う状況について議論を行う．

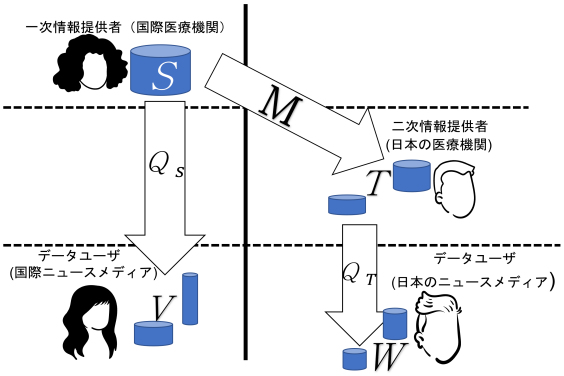

図 1 データ交換フレームワーク

1.1　問題設定

[1] では，一次情報提供者，二次情報提供者，そしてそれらが公開したデータを受け取るデータユーザの 3 種類の参加者がデータのやり取りを行う状況を想定している．そこでは，一次情報提供者がソースデータを，二次情報提供者がターゲットデータを所持し，それらのデータをデータユーザに公開する（図 1）．一般に，一次情報提供者はデータユーザに対するデータ公開ポリシ Q_S を持ち，二次情報提供者もまた，ポリシ Q_T を持つ．[2] では，Q_T が**弱適切なポリシ**であるための条件として，次の二つを定義している．

1. Q_T は秘匿性を満たす．すなわち，Q_T は Q_S が公開する情報のみ公開する．
2. Q_T は何らかの情報を公開する．

例 1. 世界中の傷病者に関するデータを収集する医療機関（以降，*WMI* と記す）と，*WMI* から提供された日本の傷病者に関するデータを管理する医療機関（以降，*JMI* と記す）が存在する状況を考える．*WMI* はその一部のデータを国際ニュースメディアに公開し，*JMI* はその一部のデータを日本のニュースメディアに公開する．この例において，*WMI* は一次情報提供者，*JMI* は二次情報提供者，ニュースメディアはデータユーザに当たる．*WMI* は関係スキーマ $S(ID, Age, Gender, Nationality)$ 上のデータを保持しているとする．これは，S という名前のテーブルに，*ID*，年齢，性別，国籍のデータを保持していることを表す．このとき，*WMI* と *JMI* の間のデータ交換メカニズムは次の連言問合せで表現される．

$$\mathbf{M}: \quad T(ID, Age, Gen) \text{ :- } S(ID, Age, Gen, \text{“}JPN\text{”}).$$

これは，テーブル S から国籍が *JPN* であるデータをすべて取り出し，テーブル T に *ID*，年齢，性別を格納することを表す．*WMI* は，ニュースメディアに対して傷病者の *ID* の情報を秘匿したいと考え，Q_S として次のような問合せを与えているとする．

$$Q_S: \quad V(Age, Gen, Nat) \text{ :- } S(ID, Age, Gen, Nat).$$

ここで，*JMI* がデータ公開ポリシとして，次のような連言問合せ Q_T を採用した場合を考える．

$$Q_T: \quad W(Age, Gen) \text{ :- } T(ID, Age, Gen).$$

このとき，Q_S が公開するデータ，すなわち，年齢と性別のみを Q_T は公開するため秘匿性を満たし，かつ何らかの情報を公開するので，Q_T は「弱適切なポリシである」という．一方で，Q_S が傷病者の *ID* と国籍の情報を秘匿したい場合の連言問合せを考える．

$$Q_S: \quad V(Age, Gen) \text{ :- } S(ID, Age, Gen, Nat).$$

一見すると，Q_T が公開する年齢と性別のデータと同じであり秘匿性を満たすように考えられるが，日本のニュースメディアは日本の傷病者に関するデータのみを得ているため，国際ニュースメディアが得ることのできない，傷病者の国籍の情報を一部得ていることになる．この場合，Q_T は Q_S が公開しない情報を公開するので秘匿性を満たしていない．よって，「弱適切なポリシは存在しない」という．

1.2　先行研究の課題と本研究の成果

我々の先行研究 [2] では，与えられた Q_S と $\mathbf{M}$ に対して，弱適切なポリシという概念を導入した．さらに，弱適切なポリシが存在するための十分条件を判定するアルゴリズムを提案し，その判定が必要条件の判定にもなるための制約を与えた．[2] で提案されたアルゴリズムとは，Q_S を

使った問合せ N を総当たりで発生させ，$N \circ Q_S$ と $Q_T^N \circ \mathbf{M}$ が等価となるような Q_T^N が存在するかを判定するアルゴリズムである．ただし，$\circ$ は問合せの合成を表す．しかし，[2] では Q_S を使った問合せ N を総当たりで発生させることから，$N \circ Q_S$ の総数が増加し，$N \circ Q_S$ と $Q_T^N \circ \mathbf{M}$ の等価判定に莫大な処理時間を要する恐れがあり，実用の場面で役に立つのかを確認する必要がある．

そこで，本研究では [2] で提案されたアルゴリズムの実装と性能評価を行う．実装と性能評価には，Python と Prolog を組み合わせてプログラムの作成に取り組んだ．次に，Q_S の引数の数を増加させたときの処理時間を計測することで性能評価を行った．貸与ノート PC での計測の結果，引数が 1〜5 個の場合には 1 つの $N \circ Q_S$ あたりに掛かる処理時間の平均は 0.0284 秒であった．しかし，引数を 6 個以上にすると $N \circ Q_S$ が 27175 個を超えた所でエラーが出力された．そのため，本研究で作成したプログラムでは Q_S の頭部の引数が 6 個以上の場合において処理時間を計測することはできなかったが，1 つの $N \circ Q_S$ あたりに掛かる処理時間から引数 8 個の場合には約 44 日間，引数 9 個の場合には約 1146 日間の処理時間が掛かると推測される．弱適切なポリシは，一度求めれば Q_S と $\mathbf{M}$ が変化しない限り再度求める必要がない．よって，[2] で提案されたアルゴリズムは Q_S の頭部の引数が 8 個程度までなら実用的な時間内に動作することがわかった．

2 諸定義

2.1 関係データベース

関係スキーマ $R[n]$ は関係データの構造を表すものであり，形式的には関係名 R とアリティと呼ばれる非負整数 n から成る．関係スキーマ上の実データは**タプル** t の集合で与えられる．$R[n]$ 上のタプルは n 個のエントリをもつ．各エントリは，**ドメイン**と呼ばれる，定数の可算集合 Dom の要素をとる．**関係インスタンス** I はタプルの有限集合である．

データベーススキーマ $\mathbf{R}$ は相異なる関係名をもつ関係スキーマの有限なリスト $\mathbf{R} = \langle R_1[n_1], \ldots, R_k[n_k] \rangle$ であり，関係データベースの構造を表す．**データベースインスタンス** $\mathbf{I}$ はリスト $\langle I_1, \ldots, I_k \rangle$ である．ただし，各 I_i は $R_i[n_i]$ の関係インスタンスである．$\mathbf{I}$ の i 番目の関係インスタンスがタプル $(a_1, \ldots, a_{n_i})$ を含むとき，$R_i(a_1, \ldots, a_{n_i}) \in \mathbf{I}$ と記す．

$\mathbf{R} = \langle R_1[n_1], \ldots, R_k[n_k] \rangle$ をデータベーススキーマとする．変数の可算集合 Var を固定する．$\mathbf{R}$ における**原子式**は $R_i(x_1, \ldots, x_{n_i})$ と表される．ただし，各 x_i は Dom ∪ Var に属する．$\mathbf{R}$ における**連言式**は原子式のリストであり，空の場合 $\top$ と記す．$\mathbf{R}$ における**連言問合せ**（以降，CQ と記す）Q は次の形式で与えられる規則である:

$$Q: \quad V(y_1, \ldots, y_m) \text{ :- } R_1, \ldots, R_\ell.$$

ただし，V はアリティ m の，$\mathbf{R}$ に属さない関係名であり，各 y_i は Dom ∪ Var に属する．$R_1, \ldots, R_\ell$ は $\mathbf{R}$ における連言式である．$V(y_1, \ldots, y_m)$ を Q の**頭部**，$R_1, \ldots, R_\ell$ を Q の**体部**と呼ぶ．すべての $y_1, \ldots, y_m$ が相異なる変数であり，かつ体部にも現れるとき，Q は**安全**であるという．Q の体部の原子式の数が一つのとき，Q は *join-free* であるという．Q の体部に同じ関係名が 2 回以上現れないとき，Q は *self-join-free* であるという．$\mu : \text{Var} \to \text{Dom}$ を変数割当てとする．$\mathbf{R}$ 上の $\mathbf{I}$ における Q の **答え** $Q(\mathbf{I})$ は次のように表される．

$$Q(\mathbf{I}) = \{(\mu(y_1), \ldots, \mu(y_m)) \mid \text{各 } R_i(x_1, \ldots, x_{n_i}) \text{ に対し } R_i(\mu(x_1), \ldots, \mu(x_{n_i})) \in \mathbf{I}\}$$

Q が安全であるならば $Q(\mathbf{I})$ は有限であり，したがって $Q(\mathbf{I})$ は $V[m]$ 上の関係インスタンスである．

安全な CQ Q の体部に現れる変数のうち，頭部にも現れる変数を *distinguished* な変数と呼ぶ．CQ Q の**代入** θ は，Q に現れる *distinguished* な変数を Var ∪ Dom の要素に写す写像である．CQ Q や原子式 φ に θ を適用して得られる CQ や原子式をそれぞれ $Q\theta$，$\varphi\theta$ と記す．

$\mathbf{A}$ をデータベーススキーマ $\mathbf{R}$ 上の連言式とする．$adom(\mathbf{A})$ を $\mathbf{A}$ に現れるすべての定数の集合（アクティブドメインと呼ぶ）とし，$var(\mathbf{A})$ を $\mathbf{A}$ に現れるすべての変数の集合とする．

2.2 問合せの等価性と書き換え

$\mathcal{I}(\mathbf{R})$ を $\mathbf{R}$ 上のすべてのデータベースインスタンスの集合とし，Q_1 と Q_2 を $\mathbf{R}$ 上の CQ とする．任意の $\mathbf{I} \in \mathcal{I}(\mathbf{R})$ に対し $Q_1(\mathbf{I}) \subseteq Q_2(\mathbf{I})$ であるとき，Q_1 は Q_2 に含まれるといい，$Q_1 \subseteq Q_2$ と記す．$Q_1 \subseteq Q_2$ であることと Q_2 から Q_1 への準同型写像が存在することは同値である．$Q_1 \subseteq Q_2$ かつ $Q_2 \subseteq Q_1$ であるとき，Q_1 と Q_2 は**等価**であるといい，$Q_1 \equiv Q_2$ と記す．

$W()$:- $\top$ の形式で与えられる nullary な（アリティが 0 の）CQ を $Q_\perp$ とする．安全な CQ Q について，$Q \not\equiv Q_\perp$ であるとき，Q は *information-revealing* であるという．これは，何らかの情報を公開するポリシに相当する．

$\mathbf{M}$ を安全な CQ のリストとする．$\mathbf{M}$ を**使った** CQ R とは，R の体部に現れるのは $\mathbf{M}$ の頭部に現れる関係名のみであるような CQ である．安全な CQ Q に対し，$\mathbf{M}$ を使った Q の *CQ-rewriting* [3] とは次のような CQ R である．

- R は $\mathbf{M}$ を使った CQ である．
- $R \circ \mathbf{M} \equiv Q$ が成り立つ．ただし，$\circ$ は問合せの合成を表す．

2.3 弱適切なターゲットポリシ

安全な CQ Q_S と安全な CQ のリスト $\mathbf{M}$ が与えられているとする．Q_S を使った $Q_T \circ \mathbf{M}$ の CQ-rewriting が存在するとき Q_T は秘匿性を満たすという．Q_T が秘匿性を満たし，かつ information-revealing であるとき，Q_T は Q_S と $\mathbf{M}$ に関して**弱適切**なポリシであるという．

3　弱適切なポリシの導出アルゴリズム

相崎らの提案アルゴリズムへの入力は，安全な CQ Q_S と安全な CQ のリスト $\mathbf{M}$ である．Q_S が self-join-free な CQ であり，かつ $\mathbf{M}$ が join-free な CQ のリストであるとき，提案アルゴリズムは健全かつ完全である（すなわち，アルゴリズムが導出したポリシは必ず弱適切であり，かつ，弱適切なポリシが存在するなら必ず導出できる）ことを [2] で示している．以下では特に断らない限り安全な CQ のみを考える．アルゴリズムは以下の 3 ステップから成る．

1. 値域が $adom(\mathbf{M}) \cup adom(Q_S) \cup dvar(Q_S) \cup \{\clubsuit\}$ であるような Q_S の代入の全体集合 Θ を求める．ここで，$dvar(Q_S)$ は Q_S における distinguished な変数の集合であり，$\clubsuit$ は $adom(\mathbf{M})$ や $adom(Q_S)$ には属さない定数である．
2. Q_S を使った，join-free かつ nullary な CQ の集合

$$\mathcal{N} = \{V_T()\text{:-}H\theta \mid \theta \in \Theta\}$$

を求める．ここで H は Q_S の頭部の原子式である．
3. 各 $N \in \mathcal{N}$ について，$\mathbf{M}$ を使った $N \circ Q_S$ の CQ-rewriting Q_T^N が存在するかを調べ，存在するならそれを求める．求めた Q_T^N が information-revealing であれば，それを弱適切なポリシとして出力する．どの $N \in \mathcal{N}$ についても information-revealing な Q_T^N が存在しなければ，「Q_S と $\mathbf{M}$ に関して弱適切なポリシは存在しない」と出力する．

4　弱適切なポリシの導出アルゴリズムの実装

4.1　実装の概要

本研究では，SWI-Prolog の version8.4.3-1 と Python の version3.9.0 を実装に用いる．3 節のアルゴリズムでは連言問合せを使用するため，問合せの評価のプログラムを書く必要がない Prolog を用いた．しかし，実装上の問題点として弱適切なポリシの存在判定の処理は Prolog のみでは難しいため，文字列の操作を容易に行える Python を組み合わせた．Prolog と Python の連携は PySwip というライブラリを用いて行った．

3 節のアルゴリズムを実装するにあたり，連言問合せの構文解析を行う関数と変数を凍結（一時的に定数として扱うための処理を指す）する関数，与えられた $N \circ Q_S$ と $\mathbf{M}$ に対して CQ-rewriting の存在判定と導出をする関数の 3 つの関数を用意した．そして，それら 3 つの関数を適宜呼び出すことで 3 節のアルゴリズムの実装を行った．なお，本研究で実装したプログラムは，現時点では問合せ $\mathbf{M}$ が 2 つの問合せから成る場合のみを想定している．

4.2　作成した関数

弱適切なポリシの導出アルゴリズムの実装にあたり作成した 3 つの関数を以下に示す．

4.2.1　連言問合せの構文解析を行う関数

この関数は，問合せを文字列として読み込み，変数，空行，定数，空行，頭部，空行，体部の順に並べ替えたものをファイルに書き出すプログラムである．文字列の分割を行うことで並び替えを行う．

4.2.2　変数を凍結する関数

この関数は，問合せ式の体部の変数を凍結し，体部の変数を凍結した変数に置き換えるプログラムである．4.2.1 節の関数によって構文解析された文字列が書かれたファイルを読み込み，空行を目印にして変数，定数，頭部，体部を別々の配列に格納する．そして，変数のみをシングルクォートで囲み凍結の処理を行う（Prolog 上ではシングルクォートで囲まれた文字列を定数として扱うため）．返り値として変数を凍結した変数に置き換えた体部の他に，頭部の文字列と配列，体部の文字列と配列を返す．

4.2.3　CQ-rewriting の存在判定と導出をする関数

この関数は，CQ R を求めたのち，それが CQ-rewriting であるかどうかを判定する．まず，4.2.2 節の関数を呼び出し凍結を行った $N \circ Q_S$ の体部をアサートする．そして，その体部に対して $\mathbf{M}$ で問い合わせることで CQ R を求める．その後，Prolog を用いてその CQ R と $\mathbf{M}$ の合成を求め，それが $N \circ Q_S$ と等価であるかどうかを判定する．等価性判定は，一方の CQ の凍結した体部に対して他方の CQ で問い合わせることで行う．これを両方に対して行いどちらの結果も True であるとき等価であるとする．返り値として CQ-rewriting が存在する場合は True，存在しない場合は False を返す．

4.3　実装したプログラム

3 節で示した，3 ステップに基づいてプログラムの説明を行う．アルゴリズムを実装するにあたり，Q_S と $\mathbf{M}$ の問合せを書き込んだファイルを用意する（q.pl，t1.pl，t2.pl）．

1. まず，値域 $adom(\mathbf{M}) \cup adom(Q_S) \cup dvar(Q_S) \cup \{\clubsuit\}$ を求める．4.2.2 節の関数を呼び出し，$\mathbf{M}$ の定数と Q_S の変数，定数を clover という文字列と共に配列 ran に格納する．次に，全体集合 Θ を求める．Q_S の頭部の引数の数を取得し，その個数分の組からなる直積集合を求める．この直積集合は配列 theta に格納され，要素はタプルとなる．
2. $N \circ Q_S$ を求める際には連言問合せの形にしておく必要はないので，$\{V_T()\text{:-}H\theta \mid \theta \in \Theta\}$ を求める．Q_S の頭部の変数をステップ 1 で求めた配列 theta のタプルの要素に置き換え，配列 htheta に格納する．
3. ステップ 2 で求めた配列 htheta と Prolog の述語を用いて $N \circ Q_S$ を求め，ファイル（noqslist.pl）に一行ずつ書き出していく．次に，各 $N \in \mathcal{N}$ について，$\mathbf{M}$ を使った $N \circ Q_S$ の CQ-rewriting Q_T^N が存在するかを調

べる．そのために，まず noqslist.pl を一行ずつ読み込み，読み込んだ $N \circ Q_S$ をファイル（noqs.pl）に書き込む．次に，noqs.pl と t1.pl，t2.pl を引数として 4.2.3 の関数を呼び出す．返り値が True 場合には，その時の CQ-rewriting Q_T^N をファイル（true.pl）に書き込む．返り値が False の場合には何もしない．

4.4 動作確認

本研究で実装したプログラムが正常に動作しているかの確認を以下の問合せについて行う．

$$\begin{aligned} Q_S: &\quad V_S(A,B) \text{ :- } S(A,B). \\ \mathbf{M}: &\quad T_1(X) \text{ :- } S(X,Y), \\ &\quad T_2(W) \text{ :- } S(Z,W). \end{aligned}$$

確認は Prolog を用いて行った．あらかじめ，引数に clover という定数を 1 つずつ持ち，関係名をそれぞれ t1，t2 とした事実を定義しておく．そして，true.pl に得られた連言問合せを 1 つずつアサートし，vt() で問合せた結果が true になるかを確認した．その結果，どの連言問合せにおいても true が得られ，正しく動作していることが確認できた．

5 性能評価

4 節で作成したプログラムの処理時間を計測することで性能評価を行う．弱適切なポリシは，一度求めれば Q_S と **M** が変化しない限り再度求める必要がないため，処理時間にリアルタイム性を必要としていない．そのため，実験に用いる PC の性能も考慮して，処理時間が 3 か月以内である場合，[2] で提案されたアルゴリズムは実用的であるとする．使用した PC は，東芝 Dynabook RX73/JBE であり，プロセッサは intel Core i5-7200U CPU 2.50GHz，RAM は 16GB，OS は Windows10 である．

5.1 評価方法

動作検証で用いた問合せが与えられた場合を例として，提案アルゴリズムの性能評価を行う．評価方法は Q_S の引数に以下のように変数を 1 個ずつ加えていった場合の処理時間を計測した．

$$Q_S: \quad V_S(X_1,\ldots,X_m) \text{ :- } S(X_1,\ldots,X_m).$$

また，処理時間の計測には Python の time モジュールを使用し，処理の開始と終了の時刻の差を求めることで処理時間を計測した．

5.2 結果と考察

計測の結果，引数が 2 個の場合の処理時間は 0.3297 秒，引数が 3 個の場合の処理時間は 1.8468 秒，引数が 4 個の場合の処理時間は 15.3466 秒，引数が 5 個の場合の処理時間は 186.3671 秒であった．この結果から引数が 1〜5 個の場合には 1 つの $N \circ Q_S$ あたりに掛かる処理時間の平均は 0.0284 秒であった．また，引数を 6 個以上にすると $N \circ Q_S$ が 27175 個を超えた所でスタックに積まれた文字列が多すぎるという意味のエラーが出力された．そこで，引数の変化によって多量のプログラムが定義されてしまっていることがエラーの原因だと考え，プログラムの数に変化があるかどうかを調べた．その結果，プログラムの数に変化は見られなかった．さらに，SWI-Prolog の version によって変化があるかを調べたところ，version8.2.4-1 では $N \circ Q_S$ を 56653 個まで求めることが可能となった．そのため，このエラーが起きた原因は SWI-Prolog にあると考えられる．

結果として，本研究で作成したプログラムでは Q_S の頭部の引数が 6 個以上の場合において処理時間を計測することはできなかった．しかし，1 つの $N \circ Q_S$ あたりに掛かる処理時間から引数 6 個の場合には 2 時間以上，引数 7 個の場合には 45 時間以上，引数 7 個の場合には約 45 時間，引数 8 個の場合には約 44 日間，引数 9 個の場合には約 1146 日間の処理時間が掛かると推測される．推定処理時間からアルゴリズムは引数が 8 個の場合までは処理時間の要求条件を満たし，実用的に動作することがわかった．

今後は引数が 6 個以上にも対応が可能となるようなプログラムに洗練化する必要がある．

6 まとめ

本研究では，弱適切なポリシの導出アルゴリズムの実装と性能評価を行った．Prolog では実現の難しい変数を凍結し置き換える処理や，文字列の操作などについては Python を用いて実現した．問合せの評価を行う箇所においては，Python で Prolog を扱うためのライブラリである PySwip を用いることで実現した．また，作成したプログラムの処理時間を Python の time モジュールを用いて計測した．弱適切なポリシは，一度求めれば Q_S と **M** が変化しない限り再度求める必要がない．よって，[2] で提案されたアルゴリズムは Q_S の頭部の引数が 8 個程度までなら実用的な時間内に動作することがわかった．

今後の課題としては，本研究で実装したプログラムでは扱えない **M** の問合せが 2 つ以外の場合においても弱適切なポリシを求めることのできる仕様に変更する点と，プログラムが正常に作動するための $N \circ Q_S$ の総数の上限を改善する点である．

参考文献

[1] Yasunori Ishihara. Toward appropriate data publishing in relational data exchange framework. In *Fourth Workshop on Software Foundations for Data Interoperability, CCIS 1281*, pp. 131–137, 2020.

[2] 相崎聖也, 石原靖哲．関係データ交換フレームワークにおける弱適切なデータ公開ポリシの存在を判定するアルゴリズム．電子情報通信学会論文誌 D, doi:10.14923/transinfj.2022PDP0011, 2022.

[3] Foto N Afrati. Determinacy and query rewriting for conjunctive queries and views. *Theoretical Computer Science*, Vol. 412, No. 11, pp. 1005–1021, 2011.