Translating μ-RA into SQL

3rd October 2019

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)

  This information can be inferred.

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)
  This information can be inferred.
- How to translate fixpoints ?

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)

  This information can be inferred.
- How to translate fixpoints ?
  - We first decompose fixpoints : $\mu(X = \varphi) = \mu(X = \kappa \cup \psi)$ where κ does not contain X and ψ depends linearly of X, i.e. every tuple of ψ is obtained from exactly one tuple of X.

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)

  This information can be inferred.
- How to translate fixpoints ?
  - We first decompose fixpoints : $\mu(X = \varphi) = \mu(X = \kappa \cup \psi)$ where κ does not contain X and ψ depends linearly of X, i.e. every tuple of ψ is obtained from exactly one tuple of X.
    Theorem: Decomposition is always possible.

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)
  This information can be inferred.
- How to translate fixpoints ?
  - We first decompose fixpoints : $\mu(X = \varphi) = \mu(X = \kappa \cup \psi)$ where κ does not contain X and ψ depends linearly of X, i.e. every tuple of ψ is obtained from exactly one tuple of X.
    Theorem: Decomposition is always possible.
    Property: Decomposed fixpoints can be computed iteratively by binding X initially to κ and then to only the new tuples generated each time.

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)

  This information can be inferred.
- How to translate fixpoints ?
  - We first decompose fixpoints : $\mu(X = \varphi) = \mu(X = \kappa \cup \psi)$ where κ does not contain X and ψ depends linearly of X, i.e. every tuple of ψ is obtained from exactly one tuple of X.
    Theorem: Decomposition is always possible.
    Property: Decomposed fixpoints can be computed iteratively by binding X initially to κ and then to only the new tuples generated each time.
  - Fixpoint terms can be translated into recursive SQL if ψ contains exactly one occurrence of X.

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)
  This information can be inferred.
- How to translate fixpoints ?
  - We first decompose fixpoints : $\mu(X = \varphi) = \mu(X = \kappa \cup \psi)$ where κ does not contain X and ψ depends linearly of X, i.e. every tuple of ψ is obtained from exactly one tuple of X.
    Theorem: Decomposition is always possible.
    Property: Decomposed fixpoints can be computed iteratively by binding X initially to κ and then to only the new tuples generated each time.
  - Fixpoint terms can be translated into recursive SQL if ψ contains exactly one occurrence of X.
    This is always the case for raw translated UCRPQs.

- Mostly straightforward
  (μ-RA is a variant of RA, and SQL is based on RA)
- Need type information to transform μ-RA's anti-projections into SQL's native projections.
  (Type of a relation = the names of its attributes)
  This information can be inferred.
- How to translate fixpoints ?
  - We first decompose fixpoints : $\mu(X = \varphi) = \mu(X = \kappa \cup \psi)$ where κ does not contain X and ψ depends linearly of X, i.e. every tuple of ψ is obtained from exactly one tuple of X.
    Theorem: Decomposition is always possible.
    Property: Decomposed fixpoints can be computed iteratively by binding X initially to κ and then to only the new tuples generated each time.
  - Fixpoint terms can be translated into recursive SQL if ψ contains exactly one occurrence of X.
    This is always the case for raw translated UCRPQs.
  - When they cannot, we can use the DBMS's procedural language to execute a `WHILE` loop computing the fixpoint into a temporary table.

So what exactly do we feed PostgreSQL?

$$
\begin{aligned}
\llparenthesis X \rrparenthesis &= \texttt{SELECT * FROM X} \\
\llparenthesis |c_i \to v_i|_{1 \leqslant i \leqslant n} \rrparenthesis &= \texttt{SELECT } v_1 \texttt{ AS } c_1, \ldots, v_n \texttt{ AS } c_n \\
\llparenthesis \sigma_{\mathfrak{f}}(\varphi) \rrparenthesis &= \texttt{SELECT * FROM } (\llparenthesis \varphi \rrparenthesis) \texttt{ WHERE } \mathfrak{f} \\
\llparenthesis \widetilde{\pi}_a(\varphi) \rrparenthesis &= \texttt{SELECT <other cols> FROM } (\llparenthesis \varphi \rrparenthesis) \\
\llparenthesis \rho_a^b(\varphi) \rrparenthesis &= \texttt{SELECT a AS b, <other\_cols> FROM } (\llparenthesis \varphi \rrparenthesis) \\
\llparenthesis \beta_a^b(\varphi) \rrparenthesis &= \texttt{SELECT a AS b <all\_cols> FROM } (\llparenthesis \varphi \rrparenthesis) \\
\llparenthesis \varphi \cup \psi \rrparenthesis &= \texttt{SELECT * FROM } (\llparenthesis \varphi \rrparenthesis) \texttt{ UNION SELECT * FROM} (\llparenthesis \psi \rrparenthesis) \\
\llparenthesis \varphi \bowtie \psi \rrparenthesis &= \texttt{SELECT * FROM } (\llparenthesis \varphi \rrparenthesis) \texttt{ NATURAL JOIN } (\llparenthesis \psi \rrparenthesis) \\
\llparenthesis \mu(X = \kappa \cup \psi) \rrparenthesis &= \texttt{X} \textit{ where we add before the query the statement:}
\end{aligned}
$$

```
CREATE TEMPORARY RECURSIVE VIEW X (<type>) AS
    SELECT * FROM ⦇κ⦈
  UNION
    SELECT * FROM ⦇ψ⦈;
```

Or, if X appears more than once in ψ:

```
CREATE TEMPORARY TABLE TMP AS
 (SELECT * FROM ⦇κ⦈);
DO $$BEGIN
 CREATE TEMPORARY TABLE X AS
  (SELECT * FROM TMP);
 WHILE EXISTS (SELECT 1 FROM X) LOOP
  CREATE TEMPORARY TABLE NEW_LINES AS
   (SELECT * FROM ⦇ψ⦈ EXCEPT SELECT * FROM TMP);
  INSERT INTO TMP (SELECT * FROM NEW_LINES);
  DROP TABLE X;
  ALTER TABLE NEW_LINES RENAME TO X;
 END LOOP;
END;$$
DROP TABLE X;
ALTER TABLE TMP RENAME TO X;
```