Adding Recursion to SPJRUD

Jef Wijsen

February 1, 2021

1 Introduction

In *Bases de Données I*, it was mentioned that the relational calculus cannot express transitive closure. Informally, this is because the relational calculus contains no recursion or looping constructs. We will now study the question of how to add some form of recursion to the relational calculus. An important concern is that the resulting language should still have a low complexity. We will express our findings in the SPJRUD algebra, which is equivalent to the relational calculus.

2 Time and Space Complexity

When we use the term *algorithm*, we mean an algorithm that terminates on every input. That is, a procedure that loops forever on some input is *not* an algorithm.

Let f be a function. We say that an algorithm *runs in* $\mathcal{O}(f(n))$ *time* if there exists a constant k such that on inputs of sufficiently large size n, the algorithm terminates after at most $k \cdot f(n)$ steps. We say that an algorithm *runs in* $\mathcal{O}(f(n))$ *space* if there exists a constant k such that on inputs of sufficiently large size n, the algorithm uses at most $k \cdot f(n)$ bits of auxiliary memory. Importantly, the space used to store the input or the output is not counted as auxiliary memory. For example, the only auxiliary memory used by the program

```
input i;
for j = 1 to i do
  print i;
```

is the memory needed for the counter j.

If f is a polynomial function, then an algorithm that runs in $\mathcal{O}(f(n))$ time is said to *run in polynomial time*. If f is a logarithmic function, then an algorithm that runs in $\mathcal{O}(f(n))$ space is said to *run in logarithmic space*. An algorithm that runs in polynomial time is also called a *polytime algorithm*. An algorithm that runs in logarithmic space is also called a *logspace algorithm*.

Every logspace algorithm can be made to run in polynomial time. Informally, this is because an algorithm that uses only $k \cdot \log n$ bits of auxiliary memory (for some input size n and constant k) cannot use more than $2^{k \cdot \log n} = n^k$ (a polynomial number) distinct auxiliary states. If ℓ is the number of lines of code of such an algorithm, then an execution with more than $\ell \cdot n^k$ steps must execute the same line of code twice with the same auxiliary state. Such an execution is in an infinite loop. But this would mean that we do not have an algorithm, because algorithms must terminate on every input.

A *decision problem* is a problem with possible outcomes "yes" and "no." The class **P** (also called **PTIME**) contains all decision problems that can be solved by a polytime algorithm. The class **L** (also called **LOGSPACE**) contains all decision problems that can be solved by a logspace algorithm. We thus have $\mathbf{L} \subseteq \mathbf{P}$; it remains an open problem whether the inclusion is strict.

3 Query Evaluation in SPJRUD

For every fixed SPJRUD expression E, we define $\mathsf{EVAL}(E)$ as the following problem:

INPUT: A database $\mathcal{I}$ and a tuple t.

QUESTION: Does t belong to $[\![E]\!]^{\mathcal{I}}$?

Recall from *Bases de Données I* that $[\![E]\!]^{\mathcal{I}}$ is the relation obtained by evaluating E on $\mathcal{I}$. Of course, in practice we will mostly be interested in computing $[\![E]\!]^{\mathcal{I}}$ rather than asking whether $[\![E]\!]^{\mathcal{I}}$ contains a particular tuple t. The reason for asking "$t \in [\![E]\!]^{\mathcal{I}}$?" is that we wanted to state our problem as a decision problem, to be able to refer to complexity classes for decision problems, like **L** and **P**. It can also be easily seen that if there exists a polytime algorithm for computing the complete answer to E, then $\mathsf{EVAL}(E)$ is in **P**; and if there exists a logspace algorithm for computing the complete answer to E, then $\mathsf{EVAL}(E)$ is in **L**.

In the database literature, the term *data complexity* is used when the complexity is measured in the size of the input database, for a fixed query, as in the above problem. This is different from *query complexity*, where the complexity is measured in the size of the query, for a fixed database. The following proposition considers the data complexity of evaluating SPJRUD expressions.

Proposition 1. *For every expression E in SPJRUD, there exists a logspace algorithm for the following problem: Given a database $\mathcal{I}$, return $[\![E]\!]^{\mathcal{I}}$. Therefore, $\mathsf{EVAL}(E)$ is in* **L** *for every expression E in SPJRUD.*

Proof sketch. Let E be an expression in SPJRUD. One first idea is to compute and store the results of subpexpressions in auxiliary relations. For example, $(R_1 \bowtie S_1) - (R_2 \bowtie S_2)$ could be computed as follows:

1. compute and store $R_1 \bowtie S_1$ in an auxiliary relation X_1;
2. compute and store $R_2 \bowtie S_2$ in an auxiliary relation X_2;
3. finally, compute and return each tuple in $X_1 - X_2$.

Alas, the auxiliary relations X_1 and X_2 may use more than logarithmic space.

The solution to this problem is simple: We do not explicitly store the intermediate results in auxiliary relations X_1 and X_2. Instead, we simulate X_1 and X_2 by remembering at all times two indexes i_1 and i_2 in X_1 and X_2, respectively. The indexes are stored in binary. Whenever the computation of the difference $(-)$ needs some tuple at position i_j in X_j $(j = 1, 2)$, that tuple is computed "on the flight."

How many bits are needed for each index? We can compute a constant k (which depends on E) such that for every database $\mathcal{I}$, no intermediate result in the computation of $[\![E]\!]^{\mathcal{I}}$ can contain more than $|\mathcal{I}|^k$ tuples. For example, for $i = 1, 2$, we have that $|R_i \bowtie S_i| \leq |R_i| \cdot |S_i| \leq |\mathcal{I}|^2$. Therefore, since an index of n bits can take 2^n distinct values, every index needs only $\mathcal{O}(\log |\mathcal{I}|)$ bits. How many indexes are needed? It suffices to have one index for every subexpression of E. Importantly, the number of indexes does not depend on $\mathcal{I}$, that is, the number of indexes is $\mathcal{O}(1)$.

In conclusion, the auxiliary memory needed to solve $\mathsf{EVAL}(E)$ is a constant number of indexes, each of logarithmic size, thus logarithmic space in total. □

The proof of Proposition 1 shows an important idea: logarithmic space is sufficient to store a constant number of indexes in the input.

4 Fixed Points

Let U be a finite set. A mapping $f : \mathcal{P}(U) \to \mathcal{P}(U)$ is said to be *inflationary* if for all $X \subseteq U$, $X \subseteq f(X)$. We say that f is *monotone* if for all $X, Y \subseteq U$, $X \subseteq Y$ implies $f(X) \subseteq f(Y)$. A set $X \subseteq U$ is a *fixed point of* f if $f(X) = X$.

Example 1. Let $U = \{a, b\}$ and f_1, f_2, f_3 as follows.

X	$f_1(X)$	$f_2(X)$	$f_3(X)$
$\emptyset$	$\{a, b\}$	$\emptyset$	$\{a, b\}$
$\{a\}$	$\{a\}$	$\{b\}$	$\{b\}$
$\{b\}$	$\{b\}$	$\{a\}$	$\{a\}$
$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\emptyset$

- f_1 is inflationary but not monotone, because $\emptyset \subseteq \{a\}$, but $f_1(\emptyset) \not\subseteq f_1(\{a\})$;
- f_2 is monotone but not inflationary, because $\{a\} \not\subseteq f_2(\{a\})$; and
- f_3 is neither inflationary nor monotone.

□

Property 1. *Define* $X^0 \coloneqq \emptyset$, *and for* $i = 0, 1, 2, \ldots$, $X^{i+1} \coloneqq f(X^i)$.

- *If* f *is inflationary or* f *is monotone, then for some* $n \leq |U|$, X^n *is a fixed point.*
- *Moreover, if* f *is monotone, then this fixed point* X^n *is included in every other fixed point of* f. *That is,* X^n *is the unique least fixed point of* f.

Proof. If X^n is a fixed point, then $X^{n+1} = f(X^n) = X^n$, and thus for all $j \geq n$, $X^j = X^n$.

Consider first the case that f is inflationary. Since $X^0 \subseteq X^1 \subseteq X^2 \subseteq \cdots$ and since U is finite, at least one inclusion must be an equality, at which point the sequence reaches a fixed point. Obviously, this fixed point is reached after at most $|U|$ steps.

Consider next the case that f is monotone. We show by induction on increasing i that for every $i = 0, 1, 2, \ldots$, we have $X^i \subseteq X^{i+1}$. This is obvious for $i = 0$, because X^0 is the empty set. For the induction step, $i \to i + 1$, the induction hypothesis is $X^i \subseteq X^{i+1}$. Since f is monotone, $f(X^i) \subseteq f(X^{i+1})$, thus $X^{i+1} \subseteq X^{i+2}$. So also in this case, $X^0 \subseteq X^1 \subseteq X^2 \subseteq \cdots$, which implies that the sequence must reach a fixed point. To show the second item, consider another fixed point Y, i.e., $f(Y) = Y$. If $X^i \subseteq Y$ for some i, then, since f is monotone, $f(X^i) \subseteq f(Y)$, thus $X^{i+1} \subseteq Y$. Since $X^0 \subseteq Y$ is obvious, it follows that every X^i is included in Y. Therefore, the fixed point reached by the sequence $X^0, X^1, X^2, \ldots$ is included in Y. □

Example 2. For f_1, f_2, f_3 as in Example 1, we obtain the following sequences:

- for f_1: $\emptyset$, $\{a, b\}$, $\{a, b\}$, $\{a, b\}$, ...
- for f_2: $\emptyset$, $\emptyset$, $\emptyset$, ...
- for f_3: $\emptyset$, $\{a, b\}$, $\emptyset$, $\{a, b\}$, $\emptyset$, $\{a, b\}$,...

Note that the sequence for f_1 reaches the fixed point $\{a, b\}$, but $\{a\}$ and $\{b\}$ are smaller fixed points of f_1. The sequence for f_3 does not converge. □

5 A Fixed Point Operator for SPJRUD

We now introduce the notion of fixed point into the SPJRUD algebra. We therefore need functions f mapping relations to relations, all with the same set of attributes. It is important that outputs and inputs of f have the same set of attributes, so that any output relation can be used again as an input relation. We start with an example.

Example 3. Let R and Δ be relation names such that $\mathsf{sort}(R) = \mathsf{sort}(\Delta) = \{A, B\}$. It is helpful to think of R as a database relation, and of Δ as some auxiliary relation. Consider

$$E \coloneqq R \cup \pi_{AB}\left(\rho_{B \mapsto C}\left(R\right) \bowtie \rho_{A \mapsto C}\left(\Delta\right)\right).$$

It can be verified that $\mathsf{sort}(E) = \{A, B\}$. Take the following relation for R:

R

A	B
1	2
2	3
3	4

We define f as the mapping that takes, as input, a relation for Δ, and returns, as output, the relation computed by E. As before, we define $\Delta^0 = \emptyset$, and for $i = 0, 1, 2, \ldots$, we define $\Delta^{i+1} = f(\Delta^i)$.

Δ^0

A	B

Δ^1

A	B
1	2
2	3
3	4

Δ^2

A	B
1	2
2	3
3	4
1	3
2	4

Δ^3

A	B
1	2
2	3
3	4
1	3
2	4
1	4

$\Delta^4 = \Delta^3$

The relation Δ^3 is a fixed point, and will be taken as the answer to E. Note that Δ^3 is the transitive closure of the input relation for R. □

Let S be a set of attributes. Let Δ be a fresh relation name. Let E be an algebra expression using Δ and relation names in the database schema such that $\mathsf{sort}(E) = \mathsf{sort}(\Delta) = S$. Although Δ is used in E, Δ is not a database relation but rather an auxiliary relation. The relation name Δ and the expression E must be of the same sort to allow for recursion: every relation returned by E will be a legal instance of Δ. We now extend the SPJRUD algebra with a new operator $\mathbf{fp}_{\Delta:S}\left(E\right)$ such that $\mathsf{sort}(\mathbf{fp}_{\Delta:S}\left(E\right)) = S$. Intuitively, the subscript $\Delta : S$ declares that Δ is a fresh relation name with $\mathsf{sort}(\Delta) = S$. It is understood that **fp**-operators can be nested: E can have a subexpression $\mathbf{fp}_{\Delta':S'}\left(E'\right)$.

Example 4. Let R be a relation name with $\mathsf{sort}(R) = \{A, B, C\}$. Then

$$E_1 \qu\coloneqq \quad \mathbf{fp}_{\Delta:ABC}\left(R \cup \pi_{ABC}\left(\rho_{B \mapsto D}\left(R\right) \bowtie \rho_{A \mapsto D}\left(\Delta\right)\right)\right)$$

is syntactically well-defined, with $\mathsf{sort}(\Delta) = \{A, B, C\}$. The new operator can be nested, as illustrated next.

$$\begin{aligned} E_2 \quad &\coloneqq \quad \pi_{AB}\left(E_1\right) \\ E_3 \quad &\coloneqq \quad \mathbf{fp}_{\Delta':AB}\left(E_2 \cup \pi_{AB}\left(\rho_{B \mapsto C}\left(E_2\right) \bowtie \rho_{A \mapsto C}\left(\Delta'\right)\right)\right) \end{aligned}$$

Can you explain what E_3 computes?

An example of a more intricate nesting is $\mathbf{fp}_{\Delta:A}\left(\Delta \cup \left(R - \mathbf{fp}_{\Delta':A}\left(\Delta' \cup \left(R - \Delta\right)\right)\right)\right)$, where Δ occurs in the **fp**-subexpression $\mathbf{fp}_{\Delta':A}\left(\Delta' \cup \left(R - \Delta\right)\right)$ and $\mathsf{sort}(R) = \{A\}$. □

We are now ready to define the formal semantics of $\mathbf{fp}_{\Delta:S}(E)$, i.e., we define what is $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}}$ for a database $\mathcal{I}$. So let $\mathcal{I}$ be a database. Let f be the following function whose argument is any relation X over S:

$$f(X) \coloneqq [\![E]\!]^{\mathcal{I}_{\Delta\to X}}. \tag{1}$$

Here, $\mathcal{I}_{\Delta\to X}$ denotes the database that maps Δ to X, and maps every other relation name R to $R^{\mathcal{I}}$. Note that f depends on $\mathcal{I}$. Now we define $\Delta^0 \coloneqq \emptyset$, and for $i = 0, 1, 2, \ldots$, define $\Delta^{i+1} \coloneqq f(\Delta^i)$. If the sequence $(\Delta^i)_{i=0}^{\infty}$ reaches a fixed point Δ^n, then $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}} \coloneqq \Delta^n$; otherwise $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}}$ remains undefined. In general, given a database $\mathcal{I}$, the answer to $[\![E]\!]^{\mathcal{I}}$ is undefined if some (not necessarily proper) **fp**-subexpression of E is undefined on $\mathcal{I}$.

Example 5. Let $\mathsf{sort}(R) = \mathsf{sort}(\Delta)$. Let $E \coloneqq R - \Delta$. Let $\mathcal{I}$ be a database. Let f be the function defined by (1). We have $f(\emptyset) = R^{\mathcal{I}}$ and $f(R^{\mathcal{I}}) = \emptyset$. Consequently, if $R^{\mathcal{I}} \neq \emptyset$, then $[\![\mathbf{fp}_{\Delta:A}(E)]\!]^{\mathcal{I}}$ is undefined; otherwise, if $R^{\mathcal{I}} = \emptyset$, then $[\![\mathbf{fp}_{\Delta:A}(E)]\!]^{\mathcal{I}} = \emptyset$. □

Alas, there is something unsatisfactory about the preceding semantics: we want to avoid situations where $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}}$ is undefined, but we cannot tell *a priori* whether such a situation can occur.

Proposition 2. *The following problem is undecidable: Given an expression* $\mathbf{fp}_{\Delta:S}(E)$*, decide whether* $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}}$ *is defined for every database* $\mathcal{I}$*.*

Proof. Let $\mathsf{sort}(\Delta) = \emptyset$. Let F be an arbitrary SPJRUD expression with $\mathsf{sort}(F) = \emptyset$. Let $E \coloneqq F - \Delta$. For every database $\mathcal{I}$ and $X \in \{\{\}, \{\{\}\}\}$, either $[\![E]\!]^{\mathcal{I}_{\Delta\to X}} = \{\}$ (the empty relation) or $[\![E]\!]^{\mathcal{I}_{\Delta\to X}} = \{\{\}\}$ (the singleton relation containing the empty tuple). If $[\![F]\!]^{\mathcal{I}} = \{\}$ for every database $\mathcal{I}$, then $[\![E]\!]^{\mathcal{I}_{\Delta\to\{\}}} = \{\}$ for every database $\mathcal{I}$, and thus $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}}$ is equal to $\{\}$ (and thus defined) for every database $\mathcal{I}$.

Now assume that $[\![F]\!]^{\mathcal{J}} = \{\{\}\}$ for some database $\mathcal{J}$. Then, for the the function f defined by $f(X) \coloneqq [\![E]\!]^{\mathcal{J}_{\Delta\to X}}$, we obtain $f(\{\}) = \{\{\}\}$ and $f(\{\{\}\}) = \{\}$, which implies that $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{J}}$ is undefined.

Consequently, $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}}$ is defined for every database $\mathcal{I}$ if and only if $[\![F]\!]^{\mathcal{I}} = \{\}$ for every database $\mathcal{I}$. The latter condition is undecidable by Trakhtenbrot's theorem, which states (paraphrasing somewhat) that there exists no algorithm for the following problem: Given an expression E in SPJRUD, is there a database $\mathcal{I}$ such that $[\![E]\!]^{\mathcal{I}} \neq \{\}$? This concludes the proof. □

That is, there exists no algorithm that, on input $\mathbf{fp}_{\Delta:S}(E)$, can tell us whether $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}}$ will be defined for all databases $\mathcal{I}$. This is unsatisfactory, because whenever we write a query in some database application, we want to be sure that the query will succeed on *every* input database. We cannot accept a situation where a query succeeds on some databases, but fails on others.

To solve this problem, we have to restrict the syntax of E. To understand the following definition, note that in $R - (S - T)$, the relation name R is in the scope of zero $-$-signs, S is in the scope of one $-$-sign, and T is in the scope of two $-$-signs.

Definition 1. Let E be an expression using **fp**-operators and operators in SPJRUD. We say that E is *positive in* Δ if all occurrences of Δ in E occur within the scope of an even number of $-$-signs; symmetrically, E is *negative in* Δ if all occurrences of Δ in E occur within the scope of an odd number of $-$-signs.

Note that $\Delta - \Delta$ is neither positive nor negative in Δ, because the first occurrence of Δ in $\Delta - \Delta$ occurs within the scope of 0 (an even number) $-$-signs, and the second occurrence within the scope of 1 (an odd number) $-$-sign. On the other hand, a subexpression in which Δ does not occur, is both positive and negative in Δ.

Proposition 3. *Let* $F = \mathbf{fp}_{\Delta:S}(E)$ *be an expression using* **fp***-operators and operators in SPJRUD. Let* $\mathcal{I}$ *be an arbitrary database, and let* f *be the mapping defined by* $f(X) = [\![E]\!]^{\mathcal{I}_{\Delta\to X}}$*. Then,*

1. *f is inflationary if for every (not necessarily proper)* **fp**-*subexpression* $\mathbf{fp}_{\Delta':S'}(E')$ *of* F*, we have that* E' *is a union of the form* $\Delta' \cup E''$*; and*

2. *f is monotone if for every (not necessarily proper)* **fp**-*subexpression* $\mathbf{fp}_{\Delta':S'}(E')$ *of* F*, we have that* E' *is positive in* Δ'*.*

Proof. To prove the first item, assume that for every (not necessarily proper) **fp**-subexpression $\mathbf{fp}_{\Delta':S'}(E')$ of F, we have that E' is a union of the form $\Delta' \cup E''$. In particular, we can assume $E = \Delta \cup E'$ for some E'. Since $\Delta^{i+1} \coloneqq f(\Delta^i) = [\![\Delta \cup E']\!]^{\mathcal{I}_{\Delta\to\Delta_i}} = \Delta^i \cup [\![E']\!]^{\mathcal{I}_{\Delta\to\Delta_i}}$, we have $\Delta^i \subseteq \Delta^{i+1}$. Note here that, by an inductive argument on the structure of E, $[\![E']\!]^{\mathcal{I}_{\Delta\to\Delta_i}}$ is well-defined.

To prove the second item, assume that for every (not necessarily proper) **fp**-subexpression $\mathbf{fp}_{\Delta':S'}(E')$ of F, we have that E' is positive in Δ'. Let $\Delta_0 \subseteq \Delta_1$ be two relations over $\mathsf{sort}(\Delta)$. We will show that for every (not necessarily proper) subexpression G of E:

- if G is positive in Δ, then $[\![G]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![G]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$; and
- if G is negative in Δ, then $[\![G]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \supseteq [\![G]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$.

The proof uses induction on the structure of G, which includes the cases of SPJRUD operators and **fp**-operators.

Case $G = \Delta$**.** Then G is positive in Δ. Obviously, $\Delta_0 = [\![\Delta]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![\Delta]\!]^{\mathcal{I}_{\Delta\to\Delta_1}} = \Delta_1$.

Case $G = R$ **with** $R \neq \Delta$**.** Then G is both positive and negative in Δ. Clearly, $R^{\mathcal{I}} = [\![R]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} = [\![R]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$.

Case $G = \sigma_{A=c}(G')$**.** We show the desired result for the case that G is positive in Δ (the case that G is negative in Δ is symmetrical). Then, G' is positive in Δ. By the induction hypothesis, $[\![G']\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![G']\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$. It follows $\sigma_{A=c}\left([\![G']\!]^{\mathcal{I}_{\Delta\to\Delta_0}}\right) \subseteq \sigma_{A=c}\left([\![G']\!]^{\mathcal{I}_{\Delta\to\Delta_1}}\right)$. Thus, $[\![\sigma_{A=c}(G')]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![\sigma_{A=c}(G')]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$.

Case $G = \sigma_{A=B}(G')$**.** Left as an exercise.

Case $G = \pi_X(G')$**.** Left as an exercise.

Case $G = G_1 \bowtie G_2$**.** Left as an exercise.

Case $G = \rho_{A\mapsto B}(G')$**.** Left as an exercise.

Case $G = G_1 \cup G_2$**.** Left as an exercise.

Case $G = G_1 - G_2$**.** We show the desired result for the case that G is positive in Δ (the case that G is negative in Δ is symmetrical). Then, G_1 is positive in Δ, and G_2 is negative in Δ. By the induction hypothesis, $[\![G_1]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![G_1]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$ and $[\![G_2]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \supseteq [\![G_2]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$. Then, $[\![G_1]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} - [\![G_2]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![G_1]\!]^{\mathcal{I}_{\Delta\to\Delta_1}} - [\![G_2]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$. Thus, $[\![G_1 - G_2]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![G_1 - G_2]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$.

Case $G = \mathbf{fp}_{\Delta':S'}(G')$**.** Then, G' is positive in Δ'. Let h be the function defined by $h(X) \coloneqq [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_0})_{\Delta'\to X}}$. Let g be the function defined by $g(Y) \coloneqq [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_1})_{\Delta'\to Y}}$. Let $X^0 = Y^0 = \emptyset$ and for all $i = 0, 1, \ldots$, define $X^{i+1} \coloneqq h(X^i)$ and $Y^{i+1} \coloneqq g(Y^i)$. By the induction hypothesis, h and g are monotone, and thus, by Property 1, both sequences will reach a fixed point. We now distinguish two subcases.

Case that G is positive in Δ. Then G' is also positive in Δ. Then, by the induction hypothesis, $X^1 = [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_0})_{\Delta'\to\emptyset}} \subseteq [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_1})_{\Delta'\to\emptyset}} = Y^1$. Then, by applying the induction hypothesis twice (the first inclusion holds true because G' is positive in Δ; the second inclusion holds true because g is monotone):

$$\begin{aligned} X^2 = [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_0})_{\Delta'\to X^1}} &\subseteq [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_1})_{\Delta'\to X^1}} \\ [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_1})_{\Delta'\to X^1}} &\subseteq [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_1})_{\Delta'\to Y^1}} = Y^2 \end{aligned}$$

It follows $X^2 \subseteq Y^2$. By repeated application of the same reasoning, $X^n \subseteq Y^n$ for all n, hence the fixed point reached by h will be contained in the fixed point reached by g. Thus, $[\![\mathbf{fp}_{\Delta':S'}(G')]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![\mathbf{fp}_{\Delta':S'}(G')]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$.

Case that G is negative in Δ. Then G' is also negative in Δ. Then, by the induction hypothesis, $X^1 = [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_0})_{\Delta'\to\emptyset}} \supseteq [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_1})_{\Delta'\to\emptyset}} = Y^1$. Then, by applying the induction hypothesis twice (the first inclusion holds true because G' is negative in Δ; the second inclusion holds true because h is monotone):

$$\begin{aligned} Y^2 = [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_1})_{\Delta'\to Y^1}} &\subseteq [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_0})_{\Delta'\to Y^1}} \\ [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_0})_{\Delta'\to Y^1}} &\subseteq [\![G']\!]^{(\mathcal{I}_{\Delta\to\Delta_0})_{\Delta'\to X^1}} = X^2 \end{aligned}$$

It follows $Y^2 \subseteq X^2$. By repeated application of the same reasoning, $Y^n \subseteq X^n$ for all n, hence the fixed point reached by g will be contained in the fixed point reached by h. Thus, $[\![\mathbf{fp}_{\Delta':S'}(G')]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \supseteq [\![\mathbf{fp}_{\Delta':S'}(G')]\!]^{\mathcal{I}_{\Delta\to\Delta_1}}$.

In particular, since E is positive in Δ, we obtain $f(\Delta_0) = [\![E]\!]^{\mathcal{I}_{\Delta\to\Delta_0}} \subseteq [\![E]\!]^{\mathcal{I}_{\Delta\to\Delta_1}} = f(\Delta_1)$. Thus f is monotone. This concludes the proof. □

We now define SPJRUD+FP as the relational algebra that extends SPJRUD with expressions $\mathbf{fp}_{\Delta:S}(E)$, possibly nested, where E must have one of the two syntactic forms in Proposition 3. That is, an expression F is syntactically legal in SPJRUD+FP if either[1]

1. for every subexpression $\mathbf{fp}_{\Delta:S}(E)$ of F, E is of the form $\Delta \cup E'$; or
2. for every subexpression $\mathbf{fp}_{\Delta:S}(E)$ of F, E is positive in Δ.

Of course, given $\mathbf{fp}_{\Delta:S}(E)$, we can decide whether E is of the form $\Delta \cup E'$ or whether E is positive in Δ. Then, by Proposition 3 and Property 1, we know that $[\![\mathbf{fp}_{\Delta:S}(E)]\!]^{\mathcal{I}}$ will be defined for every database $\mathcal{I}$. So we have solved the problem raised by Proposition 2, at the price of some syntactic restrictions. The complexity rises from **L** to **P**, as shown next.

Proposition 4. *For every expression E in SPJRUD+FP, there exists a polytime algorithm for the following problem: Given a database $\mathcal{I}$, return $[\![E]\!]^{\mathcal{I}}$. Therefore, $\mathsf{EVAL}(E)$ is in* **P** *for every expression E in SPJRUD+FP.*

Proof sketch. Consider the complexity of our new operator $\mathbf{fp}_{\Delta:S}(E)$. By Property 1, a fixed point will be reached after at most n steps, where n is the cardinality of the set U. For $\mathbf{fp}_{\Delta:S}(E)$, this set is the set of tuples over S that can be constructed from constants in $\mathcal{I}$. The cardinality of this set is $\mathcal{O}(|\mathcal{I}|^{|S|})$, a number that is polynomial in the size of $|\mathcal{I}|$ (because in this complexity analysis, E is fixed, and thus S is fixed). □

[1]Note that we exclude hybrid expressions, like $\mathbf{fp}_{\Delta:A}(\Delta \cup (R - \Delta)) - \mathbf{fp}_{\Delta':A}(\Delta' - R)$, which combine both forms. This is because in database theory, it is common to separate logics with **ifp**-operators (inflationary **fp**) from logics with **lfp**-operators (least **fp**), which correspond, respectively, to the first and second syntactic form in Proposition 3.

6 Fixed Point Operator in Relational Calculus

Let φ be a domain-independent formula in relational calculus with free variables $x_1, \ldots, x_k$. Let Δ be a k-ary relation name. Then, we introduce

$$\psi \coloneqq [\mathbf{fp}_{\Delta:x_1,\ldots,x_k}(\varphi)](y_1, \ldots, y_k) \tag{2}$$

as a formula with free variables $y_1, \ldots, y_k$, all distinct. Note that the square brackets [and] are fixed by the syntax.

We now give the semantics of (2) by providing an equivalent expression in SPJRUD+FP. In *Bases de Données I*, we have seen that for every domain-independent relational calculus expression φ with free variables $x_1, \ldots, x_k$, there exists an algebra expression E in SPJRUD with $\mathsf{sort}(E) = \{A_{x_1}, \ldots, A_{x_k}\}$ such that for every database $\mathcal{I}$, for all constants $c_1, \ldots, c_k$,

$$\mathcal{I} \models \varphi(c_1, \ldots, c_k) \text{ if and only if } \{A_{x_1} : c_1, \ldots, A_{x_k} : c_k\} \in [\![E]\!]^{\mathcal{I}}$$

Informally, E and φ are equivalent, and the free variables of φ correspond one-to-one to the attributes in $\mathsf{sort}(E)$. We define the semantics of (2) as follows:

$$\mathcal{I} \models \psi(c_1, \ldots, c_k) \text{ if } \{A_{y_1} : c_1, \ldots, A_{y_k} : c_k\} \in \left[\!\!\left[\rho_{A_{x_1},\ldots,A_{x_k} \mapsto A_{y_1},\ldots,A_{y_k}}\left(\mathbf{fp}_{\Delta:A_{x_1},\ldots,A_{x_k}}(E)\right)\right]\!\!\right]^{\mathcal{I}}.$$

Informally, the semantics of (2) computes a fixed point (if it exists) and then renames variables, so that the free variables of ψ are $y_1, \ldots, y_k$, which one-to-one correspond to $A_{y_1}, \ldots, A_{y_k}$.

Example 6. The formula $\varphi \coloneqq [\mathbf{fp}_{\Delta:x_A,x_B}(R(x_A, x_B) \vee \exists x_C (R(x_A, x_C) \wedge \Delta(x_C, x_B)))](x_D, x_E)$ has free variables x_D and x_E, and computes the transitive closure of the relation R. Here, we used variables $x_A, x_B, x_C, \ldots$ to make clear the correspondence with attributes. In relational calculus, we are generally not interested in named attributes, and will simply write:

$$\varphi \coloneqq [\mathbf{fp}_{\Delta:x,y}(R(x, y) \vee \exists z (R(x, z) \wedge \Delta(z, y)))](u, v).$$

To illustrate that such formulas can be nested, let $\nu \coloneqq \exists x (R(u, x) \vee R(x, u))$, a formula with free variable u that computes the active domain of R. Then, the formula

$$\forall u \forall v ((\nu(u) \wedge \nu(v)) \rightarrow \varphi)$$

tests whether R has a directed path from every node to every other node. □

Now one may wonder why in (2) we introduce free variables $y_1, \ldots, y_k$ instead of using $x_1, \ldots, x_k$. One reason is that with a little extension of our syntax, we can allow some y_i's to be equal to one another, or to be equal to constants. For example, for $k = 3$, we could write $[\mathbf{fp}_{\Delta:x_1,x_2,x_3}(\varphi)](y_1, c, y_1)$ with the meaning $\exists y_2 \exists y_3 \left([\mathbf{fp}_{\Delta:x_1,x_2,x_3}(\varphi)](y_1, y_2, y_3) \wedge y_2 = c \wedge y_3 = y_1\right)$.[2]

To guarantee the existence of fixed points, we have to impose some syntactic restrictions on the formula φ in (2), equivalent to the restrictions in Proposition 3. The allowed forms are the following:

- φ is of the form $\Delta(x_1, \ldots, x_k) \vee \varphi'(x_1, \ldots, x_k)$; or
- φ uses only $\forall$, $\exists$, $\wedge$, $\vee$, $\neg$, and every occurrence of Δ occurs under the scope of an even number of negations.

Concerning the second item, we have to rewrite, for example, $S(x) \wedge \forall y (\Delta(x) \rightarrow \neg R(x, y))$ into $S(x) \wedge \forall y (\neg\Delta(x) \vee \neg R(x, y))$ or $S(x) \wedge \neg\exists y (\Delta(x) \wedge R(x, y))$ to see that Δ occurs under the scope of an odd number of negations. Note incidentally that it makes no difference whether one uses universal or existential quantifiers (or both).

[2] Something similar happened in *Bases de Données I*, where the Safe Relational Calculus introduced $R(y_1, \ldots, y_k)$, with $y_1, \ldots, y_k$ distinct variables, while in practice we like to use constants and repeating variables in such formulas. For example, we like to write $R(x, c, y, y)$ instead of $\exists u \exists v (R(x, u, y, v) \wedge u = c \wedge y = v)$.

7 Transitive Closure Logic

Informally, transitive closure logic is the extension of first-order logic with a transitive closure operator. We define SPJRUD+TC as a sublanguage of SPJRUD+FP. In SPJRUD+TC, whenever we write $\mathbf{fp}_{\Delta:S}(E)$, then E must be of the syntactically restricted form

$$E' \cup \pi_S\left(\rho_{\vec{B}\mapsto\vec{C}}\left(E'\right) \bowtie \rho_{\vec{A}\mapsto\vec{C}}(\Delta)\right),$$

where for some k,

- $\vec{A} = A_1, \ldots, A_k$ and $\vec{B} = B_1, \ldots, B_k$ and $\vec{C} = C_1, \ldots, C_k$ are sequences of attributes, all distinct;
- for every $i \in \{1, \ldots, k\}$, $A_i, B_i \in S$ and $C_i \notin S$;
- $\mathsf{sort}(E') = \mathsf{sort}(\Delta) = S$.

Notice that S can contain attributes not in $\{A_1, \ldots, A_k, B_1, \ldots, B_k\}$, and that E' can itself contain an **fp**-operator of the restricted form. It is convenient to introduce a syntactic shorthand for the above expression:

$$\mathbf{tc}_{\vec{A};\vec{B}}(E).$$

Why would we care about the language SPJRUD+TC which sits between the languages SPJRUD and SPJRUD+FP? The reason is complexity. We will not discuss the following proposition in detail, but limit ourselves to saying that **NL** is a complexity class such that $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P}$.

Proposition 5. *For every expression E in SPJRUD+TC,* $\mathsf{EVAL}(E)$ *is in* **NL**.

8 Discussion

- SPJRUD+FP is a theoretically well-founded extension of SPJRUD. With some effort, all definitions and results in this paper can be translated into relational calculus.[3]
- SPJRUD+FP allows recursion.
- Every expression in SPJRUD+FP can be evaluated in polynomial time in the size of the input databases. That is, every query in this language is guaranteed to terminate after at most polynomially many steps. There is no risk to write infinite loops.
- The syntax is not as nice as Datalog (see later).
- Are there even more expressive algebras/logics that can express problems in **NP**? Yes, see the forthcoming course *Knowledge Representation & Reasoning*, which will be taught from 2019-2020 on. Note, however, that unless $\mathbf{P} = \mathbf{NP}$, a query that solves a problem in $\mathbf{NP} \setminus \mathbf{P}$ will take exponential time in the size of the input database, which is completely impractical (databases are usually "large").

[3] I used algebra because I believe that most students find it easier to take off in relational algebra than in relational calculus (right?).

9 Exercises

1. Let $E = \mathbf{fp}_{\Delta:A}(\Delta \cup (R - \mathbf{fp}_{\Delta':A}(\Delta' \cup (R - \Delta))))$.

   (a) Obviously, E is of the first form in Proposition 3. Argue that E is also of the second form.

   (b) Compute $[\![E]\!]^{\mathcal{I}}$ for a database $\mathcal{I}$ containing the following relation:

R	A
	0
	1

   Note that $[\![E]\!]^{\mathcal{I}}$ has to be computed "from the outside in."

2. Complete the proof of Proposition 3.

3. Complete the proof of Proposition 4.

4. Proposition 1 tells us that for evaluating an SPJRUD expression, the only auxiliary memory needed is a constant number of indexes, all of logarithmic size. Why would one need more auxiliary memory for evaluating an **fp**-operator?

5. Let φ be a formula in the relational calculus, using $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$. Let φ' be the formula obtained from φ by pushing negations inward as far as possible, using De Morgan's laws and the related quantification laws—and canceling double negations. Show that Δ occurs under the scope of an odd number of negations in φ if and only if Δ occurs under the scope of an odd number of negations in φ'.

6. Let $\mathsf{sort}(R) = \mathsf{sort}(\Delta) = \{A\}$. Let $E = \Delta \cup (R - \pi_A(R \bowtie \rho_{A \mapsto B}(\Delta)))$. Then, $\mathbf{fp}_{\Delta:A}(E)$ belongs to SPJRUD+FP, because it is of the first form in Proposition 3 (but not of the second form).

   (a) Show that $\mathbf{fp}_{\Delta:A}(E) \equiv R$.

   (b) Let $\mathcal{I}$ be a database such that $R^{\mathcal{I}} = \{\{A:0\}, \{A:1\}\}$. Show that the mapping f defined by $f(X) \coloneqq [\![E]\!]^{\mathcal{I}_{\Delta \to X}}$ has a fixed point that is strictly included in $[\![\mathbf{fp}_{\Delta:A}(E)]\!]^{\mathcal{I}}$, which shows that f is not monotone (by Property 1).

7. Give an expression $\mathbf{fp}_{\Delta:S}(E)$ such that the mapping f defined by $f(X) \coloneqq [\![E]\!]^{\mathcal{I}_{\Delta \to X}}$ is monotone but not inflationary.

8. Show that the expressive power of SPJRUD+FP does not decrease if we require that all **fp**-operators must be of the form $\mathbf{fp}_{\Delta:S}(\Delta \cup E)$, i.e., the first form in Proposition 3.[4]

9. Let R be a binary relation that encodes a directed graph. Which vertices are in the answer of the following query?

$$\{z \mid [\mathbf{fp}_{\Delta:x}(\exists y(R(x,y) \vee R(y,x)) \wedge \forall y(R(y,x) \to \Delta(y)))](z) \wedge \exists x R(x,z)\}$$

10. Let R be ternary relation name with $\mathsf{sort}(R) = \{A, B, C\}$. Let S be a unary relation name with $\mathsf{sort}(S) = \{A\}$. An R-tuple $\{A:p, B:q, C:r\}$ encodes the propositional formula $p \wedge q \to r$. An S-tuple $\{A:p\}$ encodes that p has truth value **true**. Write an expression E in SPJRUD+FP such that $\mathsf{sort}(E) = \{A\}$. The expression E must return $\{A:q\}$ for every propositional variable q that must be **true** in every model of the formulas in R, given the truth values in S.

[4] It is also known [Lib04, Corollary 10.12] that the expressive power of SPJRUD+FP does not decrease if we require that all **fp**-operators must be of the second form in Proposition 3.

For example, for

R	A	B	C
	p	q	r
	r	s	t
	q	u	v

S	A
	p
	q
	s

, the result is

E	A
	p
	q
	r
	s
	t

.

It is known that, unless $\mathbf{NL} = \mathbf{P}$, E cannot be expressed in SPJRUD+TC. The question whether $\mathbf{NL} = \mathbf{P}$ is an important open question in complexity theory.

11. Similar to Exercise 10, but now R is a binary relation with $\mathsf{sort}(R) = \{A, B\}$ and a tuple $\{A : p, B : q\}$ encodes the propositional formula $p \to q$. In this case, the requested expression E can be written in SPJRUD+TC.

12. Let R be a relation name with $\mathsf{sort}(R) = \{A, B\}$ used to encode a directed graph: a tuple $\{A : a, B : b\}$ denotes a directed edge from vertex a to vertex b. Let $V \coloneqq \pi_A(R) \cup \rho_{B \mapsto A}(\pi_B(R))$, the set of vertices. Assume $\mathsf{sort}(\Delta) = \{A\}$. Let

$$E \coloneqq V - \rho_{B \mapsto A}(\pi_B((V - \Delta) \bowtie R)).$$

Verify that E returns each vertex a such that the starting point of every edge that ends in a belongs to Δ.

Since the only occurrence of Δ in E occurs within the scope of two (an even number) of $-$-signs, the expression $\mathbf{fp}_{\Delta:S}(E)$ belongs to SPJRUD+FP. Let

$$F \coloneqq V - \mathbf{fp}_{\Delta:S}(E).$$

Write a short English sentence equivalent to F: "F gets every vertex a that ..."

13. Let M with $\mathsf{sort}(M) = \{A, B\}$ encode a rooted tree with root 0 such that for some even number k, every path from 0 to any leaf node has length k. In the following example relation, $k = 2$. The path $0 \to 2 \to 5$ is a path from the root to a leaf. Let W with $\mathsf{sort}(W) = \{A\}$ encode a (possibly empty) set of leaf nodes.

M	A	B
	0	1
	0	2
	1	3
	1	4
	2	5
	2	6

W	A
	5
	6

Consider now the following game between two players, called α and β, which consists in moving a pebble down the tree. The players alternate moves. The game starts when player β puts the pebble on the root node 0. Then, player α chooses a node in the *child* axis of the pebble's node, and moves the pebble to that node. Then, player β chooses a node in the *child* axis of the pebble's node, and moves the pebble to that node. Then again, player α chooses a node in the *child* axis of the pebble's node, and moves the pebble to that node. And so on. The game ends when a leaf node is reached. Since all paths from the root to any leaf contain an even number of edges, the last move will be made by player β.

Player α wins the game if the pebble's node belongs to W at the end of the game; otherwise β wins. Write an expression for the following question, called the GAME query:

> Does player α has a winning strategy, i.e., can she always win the game, no matter how β plays?

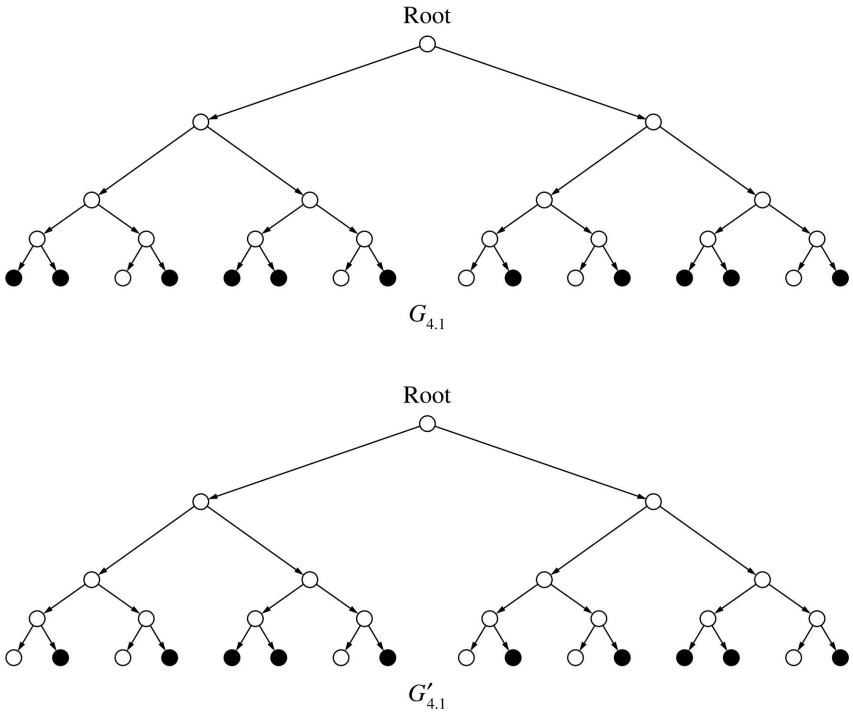

Figure 1: Game trees. The black nodes are in W. Image copied from [AHV95]

For the example relation above, player α has a winning strategy: move the pebble from 0 to 2. Then, player β must move the pebble to 5 or 6, which both belong to W.

Figure 1 shows two other databases. The directed edges represent M-tuples, and the black nodes are W-tuples. In the database $G_{4,1}$, player α has a winning strategy: always choose the left branch. In the database $G'_{4,1}$, player α has no winning strategy. Indeed, in $G'_{4,1}$, player β wins every game by always choosing the left branch.

Hint: Construct a unary relation Δ with $\mathsf{sort}(\Delta) = \{A\}$ which stores the winning positions for player α: if player β moves the pebble to a node in Δ, then player α can win the game, no matter how player β plays later on. Assume now that player β has just moved the pebble to node x. Then player α has a winning strategy if the following holds true:

$$W(x) \vee \exists y \left(M(x,y) \wedge \forall z \left(M(y,z) \rightarrow \Delta(z)\right)\right).$$

When you translate this in SPJRUD+FP, you will understand that, when it comes to readability, relational calculus is often preferable over algebra. ;)

Aside: The GAME query is of interest in database theory, where it is used to show that SPJRUD+FP is strictly more expressive than Stratified Datalog, a language that will be introduced later on. Indeed, it is known [Kol91] that the GAME query is not expressible in Stratified Datalog.

14. **Difficult exercise.** Show that SPJRU+FP has the same expressive power as Datalog.[5] Note that SPJRU+FP has no difference operator.

15. **Difficult exercise.** A difference $E_1 - E_2$ is called *atomic* if E_2 is a single relation name. Let SPJRU$\underline{\text{A}}$+FP be the restriction of SPJRUD+FP to expressions in which all difference operators

[5] Datalog will be defined later on.

are atomic. For example, if R and S are relation names such that $A \in \mathsf{sort}(R) = \mathsf{sort}(S)$, then $R - S$ belongs to SPJRU$\underline{\text{A}}$+FP, but $R - \sigma_{A=c}(S)$ does not. Show that SPJRU$\underline{\text{A}}$+FP has the same expressive power as Semipositive Datalog.[6]

References

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[Kol91] Phokion G. Kolaitis. The expressive power of stratified programs. *Inf. Comput.*, 90(1):50–66, 1991.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[6] Semipositive Datalog will be defined later on.