Operational Semantics for Fixed-Point Logics on Constraint Databases

Stephan Kreutzer

LuFG Mathematische Grundlagen der Informatik,
RWTH Aachen, D-52056 Aachen
Email: kreutzer@informatik.rwth-aachen.de

Abstract. In this paper we compare the expressive power of various fixed-point logics on linear or dense order constraint databases. This comparison is not done on absolute terms, i.e. by comparing their expressive power for arbitrary queries, rather for definability of partially recursive queries. The motivation for choosing this benchmark comes from fixed-point logics as query languages for constraint databases. Here, non-recursive queries are of no practical interest.
It is shown that for linear constraint databases already transitive closure logic is expressive enough to define all partially recursive queries, i.e., transitive-closure logic is expressively complete for this class of databases. It follows that transitive-closure, least, and stratified fixed-point logic are equivalent with respect to this benchmark.

1 Introduction

Logics and query languages allowing the definition of fixed-points of definable operators have a long tradition both in (finite) model theory and database theory.

In finite model theory, the interest in fixed-point logics comes from questions connected with the definability or description of computations in logical formalisms. To obtain logics strong enough to describe properties interesting from a computational point of view, extensions of first-order logic by operators to define fixed-points have been considered. Among the logics obtained in this way are *transitive-closure logic* (FO(TC)), which extends first-order logic by an operator to define the transitive closure of definable graphs, and *least fixed-point logic* (FO(LFP)), which extends first-order logic by an operator to define the least fixed-point of operators defined by positive formulae. We give precise definitions of these logics in Section 2.

Query languages incorporating fixed-point concepts have also been studied in database theory. Here, the fixed-point constructs have not been added to full first-order logic but to conjunctive queries instead. To obtain more expressive logics than this query language, called *Datalog*, database theorists incorporated negation in various ways. One of these extensions is *Stratified Datalog*, where negation is allowed in a limited way. Stratified Datalog plays a rather prominent role in database theory as it has a good balance between expressive power and complexity.

Once the logics and query languages had been defined, it became obvious that fixed-point logics had corresponding Datalog variants and vice versa. For instance, Datalog is a logic equivalent to existential fixed-point logic (FO(EFP)), the restriction of least fixed-point logic to existential formulae.

Generally, it is clear that FO(TC) $\subseteq$ FO(SFP) $\subseteq$ FO(LFP) and, further, that FO(EFP) $\subseteq$ FO(SFP), where FO(SFP) is equivalent to Stratified Datalog[1]. Further, FO(TC) and FO(EFP) are incomparable. Whether these inclusions are strict depends on the kind of structures under consideration. It has been shown, that on arbitrary structures all inclusions are strict. The standard example separating FO(SFP) and FO(LFP) is the property of a binary relation of being a well-ordering. See [DG] for a very recent survey on fixed-point logics.

On finite structures, the situation depends on whether an ordering is available. If the structures are ordered, FO(SFP) and FO(LFP) coincide with PTIME and, if the order is given by a successor relation, the same is true for FO(EFP). Whether FO(TC) and FO(LFP) have the same expressive power on finite ordered structures depends on whether NLOGSPACE equals PTIME. See [EF95] for an extensive study of these questions on finite structures.

In [Kol91], Kolaitis showed that on unordered finite structures FO(SFP) is strictly weaker than FO(LFP). To prove this he used so-called game trees which can be defined in FO(LFP) but not in FO(SFP).

In this paper we are mainly interested in the expressive power of transitive-closure logic, stratified Datalog, and least fixed-point logic in the context of constraint databases (See [KLP00] for a detailed overview of constraint databases.) We give a precise definition of constraint databases in the next section. These structures lie somewhere between finite and arbitrary infinite structures. They usually have an ordering on their universe available, so that the separation methods for finite structures do not work, but on the other hand the database relations often provide too much structure to use the methods that work on general infinite structures. The problem here is, that - classically - logics are separated by showing that a class of structures with certain properties is definable in one logic but not in the other. In the constraint setting, the context structure is fixed in advance, whereas the relations that vary are first-order definable and thus have a rather simple structure. However, it can still be shown that on many interesting context structures, least fixed-point logic is more expressive than, for instance, stratified Datalog.

So far, much work on fixed-point logics in the context of constraint databases has focused on syntactic variants defined with regard to certain properties of the resulting languages, as polynomial time complexity, see e.g. [GK97,Kre01], or termination, see e.g. [GK00]. Further, the logics have been considered in terms of absolute definability, i.e., the main question was whether there are relations or properties definable in one logic but not in another.

Besides this classification, the expressive power can also be measured with respect to a fixed benchmark. This approach is taken here, where we consider

[1] Actually, FO(SFP) stands for *stratified fixed-point logic*, a name that will become clear later.

constraint databases over the real ordered group $(\mathbb{R}, <, +)$. As benchmark we take the class of partially computable queries. Precisely, we investigate which portions of this class of queries are definable in each of the logics mentioned above.

The motivation for choosing the class of partially recursive queries as benchmark comes from the use of fixed-point logics as query languages. Someone using the logics to ask queries about a given database will only be interested in computable queries, i.e., those whose evaluation terminates. Thus a query language that is more expressive than another only in means of the power to define non-recursive queries will not be considered as being expressively stronger.

When speaking about logics defining partially computable queries we first have to specify what it means for a formula to define a computable query. So far we only considered the standard model-theoretical semantics of fixed-point logics. Formally, the model-theoretical semantics for a logic $\mathcal{L}$ can be seen as a function

$$\mathrm{mod} : \varphi \in \mathcal{L} \longmapsto (\mathrm{mod}_\varphi : (A, \sigma) \longmapsto \mathcal{P}(A^*)),$$

taking formulae $\varphi \in \mathcal{L}$ to functions that map a database $\mathfrak{B} := (A, \sigma)$ to relations over its universe. The problem is that the functions mod_φ are not required to be computable. Further, in the constraint database setting, the problem arises that the resulting relations on A are not necessarily finitely representable in the context structure. Therefore, we have to give the logics an operational semantics, i.e., a total recursive function

$$\mathrm{op} : \varphi \in \mathcal{L} \longmapsto (\mathrm{op}_\varphi : (A, \sigma) \longmapsto \mathcal{P}(A^*)),$$

where the functions op_φ are partially recursive and take databases to finitely representable relations.

Clearly, not every such function can sensibly be called an operational semantics. One obvious condition that should be satisfied by any operational semantics for a logic is, that it is consistent with the model-theoretical semantics, i.e., for all databases $\mathfrak{B}$ and formulae φ where $\mathrm{op}_\varphi(\mathfrak{B})$ is defined, the two semantics should agree on the result, i.e., $\mathrm{mod}_\varphi(\mathfrak{B}) = \mathrm{op}_\varphi(\mathfrak{B})$.

Although the consistency condition is a necessary condition, it alone does not ensure that the semantics meets our expectations. For instance, the operational semantics mapping each formula to the everywhere undefined function would trivially satisfy the consistency requirement.

Thus, besides this consistency criterion also some kind of completeness condition is needed. Unlike for consistency, there is no canonical definition for an operational semantics to be complete. In this paper we take the view that an operational semantics for a logic is complete if it is most powerful possible, i.e. there is no other operational semantics and no partially computable query which is definable in the logic under the second but not under the first semantics.

In Section 3, we define operational semantics for the fixed-point logics mentioned above which satisfy both, the consistency and the completeness condition. The latter condition will be proved by showing that under the operational semantics we define for transitive-closure logic, all partially computable queries on linear constraint databases become definable in FO(TC). This shows that FO(TC) is expressively complete for this class of databases.

It follows from this that also least and stratified fixed-point logic are expressively complete and thus the three logics are equivalent with respect to the class of partially computable queries. This equivalence does not hold for existential fixed-point logic, where we show that there even exist queries definable in first-order logic but not in existential fixed-point logic.

Finally, in Section 4 we will consider constraint databases over the real line. Here it will be shown that the question about the expressive power of fixed-point logics can be reduced to the question for finite ordered databases. Thus, stratified Datalog and least fixed-point logic have the same expressive power and transitive-closure logic is equal to both if, and only if, NLogspace equals Ptime.

2 Preliminaries

Constraint Databases. Constraint Databases have been introduced by Kanellakis, Kuper, and Revesz [KKR90,KKR95] as a model for infinite relational databases that have a finite representation and are therefore accessible to algorithmic problems.

Let a signature τ and a τ-structure $\mathfrak{A}$ be given. *Constraint databases* are defined with respect to this fixed structure $\mathfrak{A}$. For a finite relational signature σ, a *σ-constraint database $\mathfrak{B}$ over* $\mathfrak{A}$, or simply *constraint database* if σ and $\mathfrak{A}$ are understood or irrelevant, is a σ-expansion of $\mathfrak{A}$ such that for all relation symbols $R \in \sigma$ there is a quantifier-free formula over $\mathfrak{A}$ defining $R^{\mathfrak{B}}$ in $\mathfrak{A}$. In these defining formulae elements of the universe may be used as parameters. The database relations in σ are called *finitely representable* or *constraint relations.*

In this paper we are specifically interested in *linear constraint databases*, i.e., constraint databases over the structure $(\mathbb{R}, <, +)$. Thus, linear constraint relations are defined by boolean combinations of linear equalities and inequalities. In this paper we only allow rational coefficients in these formulae defining database relations[2].

Fixed-point logics. As mentioned in the introduction, we are specifically interested in the expressive power of transitive-closure and least fixed-point logic as well as Datalog and stratified Datalog. We define FO(TC) and FO(LFP) first and then turn to the Datalog variants.

Definition 1 Transitive-closure logic (FO(TC)) *is defined as the extension of first-order logic by the following formula building rule. If $\varphi(\overline{x}, \overline{y})$ is a formula with free tuples of variables $\overline{x}, \overline{y}$ of equal length, then*

$$\psi := [\mathrm{TC}_{\overline{x},\overline{y}}\, \varphi](\overline{s}, \overline{t})$$

[2] After all, constraint databases have been defined in order to be finitely representable and it is debatable whether formulae with transcendental coefficients could sensibly be called finitely representable.

is also a formula, where $\overline{s}, \overline{t}$ are tuples of terms of the same length as $\overline{x}$ and $\overline{y}$. The free variables of ψ are the variables occurring free in $\overline{s}, \overline{t}$ and the free variables of φ other than $\overline{x}$ and $\overline{y}$.

Given a structure $\mathfrak{A}$, the semantics of FO(TC) is defined inductively with the meaning of the TC*-rule being that for some tuples $\overline{a}, \overline{b}$, $\mathfrak{A} \models ([\mathrm{TC}_{\overline{x},\overline{y}}\, \varphi])[\overline{a}, \overline{b}]$ if, and only if, $(\overline{a}, \overline{b})$ is in the transitive closure of the relation $\{(\overline{u}, \overline{v}) : \mathfrak{A} \models \varphi[\overline{u}, \overline{v}]\}$.*

Note that the result of a formula in FO(TC) on a linear constraint database does not have to be finitely representable anymore. For instance, the formula $\varphi(y) := [\mathrm{TC}_{x,y}\, x + 1 = y](0, y)$ defines the natural numbers on any linear constraint database. But clearly, the result is not finitely representable.

Definition 2 Least fixed-point logic FO(LFP) *is defined as the extension of first-order logic by the following formula building rule. If $\varphi(R, \overline{x})$ is a formula with free first-order variables $\overline{x} := x_1, \ldots, x_k$ and a free second-order variable R of arity k such that φ is positive in R, then*

$$\psi := [\mathbf{lfp}_{R,\overline{x}}\, \varphi](\overline{t})$$

is also a formula, where $\overline{t}$ is a tuple of terms of the same length as $\overline{x}$. The free variables of ψ are the variables occurring in $\overline{t}$ and the free variables of φ other than $\overline{x}$.

Given a structure $\mathfrak{A}$, the semantics of FO(LFP) *is defined inductively with the meaning of the* FO(LFP)*-rule being as follows. Define the stages R^α of an fixed-point induction as*

$$\begin{array}{ll} R^0 & := \varnothing \\ R^{\alpha+1} & := \{\overline{a} : \mathfrak{A} \models \varphi[R^\alpha, \overline{a}]\} \\ R^\lambda & := \bigcup_{\alpha<\lambda} R^\alpha. \end{array}$$

For a given structure $\mathfrak{A}$ and a tuple $\overline{a} \in A$, the formula $[\mathbf{lfp}_{R,\overline{x}}\, \varphi]$ becomes true for $\overline{a}$ if, and only if, $\overline{a} \in R^\alpha$, where α is the least α such that $R^\alpha = R^{\alpha+1}$. As φ is required to be positive in R such a stage α always exists.

We now turn to the definition of the Datalog variants. To harmonise notation, we will not deal with the Datalog variants directly but consider the corresponding fixed-point logics instead. In the case of Datalog this is the well known existential fixed-point logic.

Definition 3 Existential least fixed-point logic *is defined as the restriction of* FO(LFP) *to formulae without universal quantifiers and with negation being allowed only in front of atoms which are not built up from fixed-point variables.*

For stratified Datalog there are various syntactically different fixed-point extensions of FO equivalent to it. We follow the notation introduced by Kolaitis in [Kol91][3].

[3] Note that in this paper the logic was called *existential* fixed-point logic!

Definition 4 Stratified Fixed-Point Logic (FO(SFP)) *is defined as follows.*

- *Let* $\mathrm{FO(SFP)}_1$ *be defined as existential fixed-point logic.*
- *Define* $\mathrm{FO(SFP)}_{i+1}$ *as the class of existential fixed-point formulae which may use literals of the form* $\psi(\overline{x})$, *where* ψ *is an* $\mathrm{FO(SFP)}_l$ *formula with* $l \leq i$. *Note that these formulae may occur under the scope of negation symbols.*

Stratified Fixed-Point logic is defined as the union $\mathrm{FO(SFP)} := \bigcup_{i\in\omega} \mathrm{FO(SFP)}_i$ *of all* $\mathrm{FO(SFP)}_i$ *for* $i \in \omega$.

Clearly, FO(SFP) and stratified Datalog have the same expressive power and a stratified Datalog program with l strata corresponds to a formula in $\mathrm{FO(SFP)}_l$. We sometimes write $[\mathbf{sfp}_{R,\overline{x}}\, \varphi]$ instead of $[\mathbf{lfp}_{R,\overline{x}}\, \varphi]$. This happens in cases where we compare FO(SFP) and FO(LFP) and want to make it very clear in which logic we are.

It can easily be shown that a formula $[\mathrm{TC}_{\overline{x},\overline{y}}\, \varphi(\overline{x},\overline{y})](\overline{s},\overline{t})$ of FO(TC) is equivalent to the FO(SFP)-formula $[\mathbf{lfp}_{R,\overline{z}}\, (\overline{z} = \overline{s} \vee \exists \overline{z}'\, (R\overline{z}' \wedge \varphi(\overline{z}',\overline{z})))](\overline{t})$. Thus, $\mathrm{FO(TC)} \subseteq \mathrm{FO(SFP)}$. In fact, it can even be shown that every FO(SFP)-formula of the form $[\mathbf{lfp}_{R,\overline{x}}\, \varphi_0(\overline{x}) \vee \exists \overline{x}' \in R\, \varphi_1(\overline{x},\overline{x}')](\overline{t})$ such that R does not occur in φ_0 and φ_1 is equivalent to a formula in FO(TC), provided that φ_0 and φ_1 are equivalent to formulae in FO(TC).

It is also clear that $\mathrm{FO(SFP)} \subseteq \mathrm{FO(LFP)}$.

3 Finitely representable expansions of the real ordered group

In this section we consider constraint databases over the real ordered group $(\mathbb{R}, <, +)$ and show that transitive-closure, least, and stratified fixed-point logic define the same class of partially computable queries. This is proven by showing that all partially computable queries are already definable in FO(TC). For this, we first have to give the logics an operational semantics.

3.1 Operational semantics for fixed-point logics

Throughout this section, we denote by CDB the class of constraint databases over $(\mathbb{R}, <, +)$ and by CRel the class of finitely representable relations over $(\mathbb{R}, <, +)$. We don't distinguish between finitely representable relations and their representing formulae and denote both by CRel. It will always be clear from the context whether the relation or the formula is meant.

Definition 5 *For a given logic* $\mathcal{L}$, *an operational semantics* $op_{\mathcal{L}}$ *is defined as a total recursive function*

$$\mathrm{op} : \varphi \in \mathcal{L} \longmapsto (\mathrm{op}_\varphi : \mathrm{CDB} \longrightarrow \mathrm{CRel})$$

taking formulae $\varphi \in \mathcal{L}$ *to partially computable functions* op_φ *which take constraint databases over* $(\mathbb{R}, <, +)$ *to representations of finitely representable relations over* $(\mathbb{R}, <, +)$.

For $\varphi \in \mathcal{L}$ we denote by $mod_\varphi(\mathfrak{B})$ the set of elements defined by φ under the model-theoretical semantics for $\mathcal{L}$. An operational semantics op for $\mathcal{L}$ is consistent with the model-theoretical semantics *if, and only if, for all formulae φ and all databases $\mathfrak{B}$ such that $op_\varphi(\mathfrak{B})$ is defined, $mod_\varphi(\mathfrak{B}) = op_\varphi(\mathfrak{B})$.*

We now define an operational semantics for transitive-closure logic. This operational semantics closely resembles the usual definition of the model-theoretical semantics.

Definition 6 (Operational semantics for FO(TC)) *For each formula $\varphi \in$ FO(TC) we define a function $op_\varphi : \mathrm{CDB} \longrightarrow \mathrm{CRel}$ by induction on the structure of φ. Fix a quantifier elimination procedure for $(\mathbb{R}, <, +)$, i.e., an evaluation schema for first-order queries.*

- *If $\varphi \in \mathrm{FO}$, define $op_\varphi(\mathfrak{B}) := \varphi'$, where φ' is obtained from φ by first substituting each occurrence of a database relation symbol by the formula defining the relation in $\mathfrak{B}$ and then eliminating the quantifiers using the quantifier elimination method fixed above.*
- *If $\varphi := \varphi_1 \wedge \varphi_2$, define $op_\varphi(\mathfrak{B})$ as $((op_{\varphi_1}(\mathfrak{B})) \wedge (op_{\varphi_2}(\mathfrak{B}))$. For other boolean connectives the function op_φ is defined analogously.*
- *If $\varphi := \exists x \varphi_1$, define $op_\varphi(\mathfrak{B})$ as the result of applying the quantifier-elimination method fixed above to $op_{\varphi_1}(\mathfrak{B})$.*
- *Now suppose that φ is of the form $\varphi := [\mathrm{TC}_{\overline{x},\overline{y}}\, \psi(\overline{x}, \overline{y})](\overline{u}, \overline{v})$. We inductively define formulae σ_i, $i \in \omega$, as follows. Recall that we allowed the use of rational numbers as parameters in the formulae. Let I be the indices of parameters among $\overline{u} := u_1, \ldots, u_n$, i.e., u_i is a constant if, and only if, $i \in I$.*
  (i) $\sigma_0 := \psi(\overline{x}, \overline{y}) \wedge \bigwedge_{i \in I} u_i = x_i$.
  (ii) $\sigma_{i+1} := \exists \overline{x}'\, \sigma_i(\overline{x}, \overline{x}') \wedge \psi(\overline{x}', \overline{y})$.
  If there is no $j \in \omega$ such that σ_j and σ_{j+1} are equivalent in $\mathfrak{B}$, then $op_\varphi(\mathfrak{B})$ is undefined. Otherwise let i be the smallest such j and define
  $$op_\varphi(\mathfrak{B}) := \exists \overline{x} \exists \overline{y}\, ((\overline{u} = \overline{x}) \wedge (\overline{y} = \overline{v}) \wedge \sigma_i(\overline{x}, \overline{y})).$$

Finally, we define the operational semantics op for FO(TC) as the function taking formulae φ to op_φ.

Observe the difference between this definition of an operational semantics and the standard way to define the model-theoretical semantics as outlined in Definition 1 in the way the formula σ_0 is defined. The conjunct $\bigwedge_{i \in I}(u_i = x_i)$ reduces the computation of the transitive closure to the computation of all tuples which are reachable from tuples with some fixed components. Thus, by letting constants occur in the tuple $\overline{u}$ one gets some control over the process of building up the transitive closure. However limited this control might seem, we will show below that it is enough to allow the definition of arbitrary partially computable queries over databases from CDB, whereas it seems unlikely that this is also possible without this modification.

It is now an easy observation that the model-theoretical and the operational semantics for FO(TC) are consistent.

Proposition 7 *The operational semantics of Definition 6 is consistent with the model-theoretical semantics of FO(TC).*

3.2 Expressive completeness of transitive closure logic

We now turn to the definability of partially computable queries by formulae of FO(TC).

Definition 8 *A partially computable query Q is defined by a formula $\varphi \in$ FO(TC) if for all databases $\mathfrak{B}$, $Q(\mathfrak{B})$ is defined if, and only if, $op_\varphi(\mathfrak{B})$ is defined and in this case $Q(\mathfrak{B}) = op_\varphi(\mathfrak{B})$.*

We show now that all partially computable functions on constraint databases over $(\mathbb{R}, <, +)$ can be defined in FO(TC). The proof runs along the following line. We first show that the logic PFOL as introduced by Vandeurzen et. al. [Van99] is a subset of FO(TC). This enables us to use the results on finite representations definable in PFOL. We then show that the run of Turing-machines can be simulated in FO(TC).

Recall that PFOL was defined as an extension of FO by a restricted form of multiplication. Precisely, the logic allows the use of atoms $x \cdot p = y$, where p is a so-called *product variable.* These product variables have to be bound by a quantifier $\exists p \in \varphi_p$ or $\forall p \in \varphi_p$, where $\varphi_p(x)$ must be a formula defining a finite set. The semantics of a quantifier $Qp \in \varphi_p$ is the semantics of Q relativised to the set defined by φ_p.

To show that PFOL $\subseteq$ FO(TC) it suffices to prove that atoms of the form $x \cdot p = y$ can be defined in FO(TC) by a formula whose evaluation always terminates.

We show that atoms of this form can be defined by a formula *mult* in FO(TC) provided that the formula φ_p defines a set of rational numbers. In all cases where we use PFOL formulae below this will always be true. The formula *mult* makes use of two auxiliary formulae $\varphi_{nd}(p, n, d)$, stating that n and d are the numerator and denominator of the rational number p, and $\varphi_{im}(a, b, c)$, which defines $a \cdot b = c$, provided that b is an integer.

By the discussion above, we may use quantifiers $\exists p \in \varphi_p$ in our formulae as abbreviation for $\exists p \varphi_p(p)$, where φ_p is the unique formula binding p in the PFOL formula.

The formula φ_{im} is defined as

$$\varphi^{\psi}_{im}(x, y, z) := [\mathrm{TC}_{x,y,z;x',y',z'} \begin{matrix} x = x' \land y' = y - 1 \land \\ z' = z + x \land 0 \leq y \land \psi(y) \end{matrix}](x, y, 0, x, 0, z).$$

The formula is parameterised by the formula ψ which will be replaced by a concrete formula whenever we use φ_{im} below. The idea is, that ψ bounds the possible values for y from above, whereas the conjunct $0 \leq y$ bounds y from below. Thus, if there exists a number c such that ψ is not satisfied for any $c' > c$, then the evaluation of φ_{im} is guaranteed to terminate. We abbreviate $\varphi^{\psi}_{im}(a, b, c)$ as $a \cdot^{\psi}_{i} b = c$.

We now give the definition of the formula *mult* and define φ_{nd} below. The formula *mult* is defined as

$$\text{mult}(x, p, y) := \exists d \exists n\ \varphi_{nd}(p, n, d) \land (x \cdot_i^{\varphi_n} n = z \cdot_i^{\varphi_d} d),$$

where $x \cdot_i^{\varphi_n} n = y \cdot_i^{\varphi_d} d$ is an abbreviation for $\exists z(x \cdot_i^{\varphi_n} n = z \land z = y \cdot_i^{\varphi_d} d)$, and the formulae φ_n and φ_d are defined as $\varphi_n(x) := \exists d \exists p\, \varphi_{nd}(p, x, d)$ and $\varphi_d(x) := \exists n \exists p\, \varphi_{nd}(p, n, x)$.

Finally, to define φ_{nd} we assume a formula $\gamma(i, j, i', j')$ defining a Gödel enumeration of pairs (i, j) of natural numbers. Using this, we can set

$$\begin{aligned}\varphi_{nd}(p, n, d) := \varphi_p(p) \land d \cdot_i^{\varphi_p} p = n \land \\ [\text{TC}_{x,y,x',y'} \neg x = y \cdot_i^{\varphi_p} p \land \gamma(x, y, x', y')\ ](1, 0, n, d),\end{aligned}$$

where φ_p is the formula binding p in the PFOL formula.

Recall that the operational semantics above guarantees that the evaluation of the TC operator start with the pair $0, 1$. The conjunct $\neg x = y \cdot_i^{\varphi_p} p$ ensures that it terminates once a pair n, d of numerator and denominator for p is reached. Thus φ_{nd} defines exactly one pair n, d for each p. As the formula is used only for product variables p, it defines a finite set. This ensures that the formula φ_n and φ_d above define finite sets as well. Thus, for product variables p the formula $\text{mult}(x, p, y)$ terminates and defines the set $\{(a, b, c) : a \cdot b = c \text{ and } b \in \varphi_p\}$.

The proof of the following lemma is now straight-forward.

Lemma 9 *Each PFOL formula where all product variables are bound by formulae defining sets of rationals only is equivalent to a formula in FO(TC) whose evaluation always terminates.*

It has been shown by Vandeurzen [Van99] that there are PFOL queries *code* and *decode*, such that for a given databases $\mathfrak{B} := ((\mathbb{R}, <, +), S^{\mathfrak{B}})$, where S is k-ary, *code* defines a finite set $S^{\text{enc}} \subseteq \mathbb{R}^{k(k+1)}$ of $(k+1)$-tuples of points in $\mathbb{R}^k$, and $S^{\mathfrak{B}}$ can be recovered from this finite encoding S^{enc} by the formula *decode*, i.e., $S = \{\overline{a} : (\mathbb{R}, <, +) \models \text{decode}(S^{\text{enc}})\}$.

As all parameters occurring in the formulae defining the database relations are required to be rational, and therefore also all points in the encoding have rational coordinates, Lemma 9 implies that such an finite encoding of the database relation can also be defined in FO(TC).

We now turn to the simulation of the run of a Turing-machine $M := (Q, \Sigma := \{0, 1\}, q_0, \delta, \{q_f\})$ computing a given query. Here, Q is the set of states of M, Σ is the alphabet, q_0 the initial state, and q_f the unique halting state. δ is a set of rules of the form $(q, a) \to (q', a', m)$, where $q, q' \in Q$, $a, a' \in \Sigma$, and $m \in \{-1, 1, 0\}$. Such a rule states that if M is in state q and the head scans a position labelled a then M replaces a by a', goes into state q' and moves according to m the head to the left, to the right, or not at all.

We can assume w.l.o.g. that M operates on the encoding of the input database as defined above. Further, we assume that the machine halts with the head scanning the first position on the tape.

A configuration of M will be encoded as a tuple (x_l, x_r, t, s), where x_l and x_r are natural numbers encoding the tape content, $0 \leq t \in \mathbb{N}$ denotes the step counter, and s contains the current state of the Turing-machine. A tape content $a_0 a_1 \dots a_n$ will be encoded as follows. Let $0 \leq p \leq n$ be the current head position. The inscription $a_0 \dots a_{p-1}$ of the tape to the left of p is coded inversely, i.e., as $a_{p-1} \dots a_0$, in x_l by $x_l := \Sigma_{i=0}^{p-1} 2^i \cdot a_{p-1-i}$. The inscription $a_p \dots a_n$ of the tape to the right of p is coded in x_r by $x_r := \Sigma_{i=0}^{n-p} 2^i \cdot a_{p+i}$. As the machine only uses finitely many positions on the tape and all cells which have not been visited by the machine are defined to be 0, we can also think of x_r as the infinite sum $\Sigma_{i=0}^{\infty} 2^i \cdot a_{p+i}$.

The run of M will be simulated by the formula φ_M. We use bold face letters $\mathbf{q_0}, \mathbf{q_f}, \dots, \mathbf{a}, \mathbf{m}$ to denote fixed constants of the Turing-machine, e.g. states $\mathbf{q_0}$, symbols $\mathbf{a}$ of the alphabet, or $\mathbf{m}$.

$$\varphi_M := \exists t \exists x \; [\mathrm{TC}_{\substack{x_l, x_r, t, s; \\ x'_l, x'_r, t', s'}} \begin{pmatrix} (t = -1) \wedge \mathrm{init}) \vee \\ (t \geq 0 \wedge s \neq \mathbf{q_f} \wedge \mathrm{compute} \end{pmatrix}](\begin{matrix} -1, -1, -1, -1; \\ x, t, \mathbf{q_f}, 0 \end{matrix}).$$

The formulae *init* and *compute* are defined such that

1. $\mathit{init}(x'_l, x'_r, t', s')$ becomes true for the tuple x'_l, x'_r, t', s' coding the input configuration, i.e., $x'_l = 0$, x'_r codes the input, $s' = \mathbf{q_0}$, i.e., the machine is in the initial state, and, finally, $t' = 0$.
2. $\mathrm{compute}(x_l, x_r, t, s; x'_l, x'_r, t', s')$ becomes true for a pair of tuples, if the x'_l, x'_r, t', s' codes the successor configuration of the configuration coded in x_l, x_r, t, s.

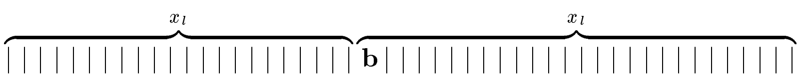
The conjunct $s \neq \mathbf{q_f}$ is needed to terminate the evaluation of the formula once the machine reaches the final state q_f.

We now turn to the definition of the formula *compute*. To define this, we first need three auxiliary formulae *move-right*$_a$, *move-left*$_a$, and *don't-move*$_a$ with free variables $\{x_l, x_r, x'_l, x'_r\}$ which define the transition from the tape content coded in (x_l, x_r) to the new tape content (x'_l, x'_r) if the machine writes the symbol a and moves to the right, to the left or doesn't move at all.

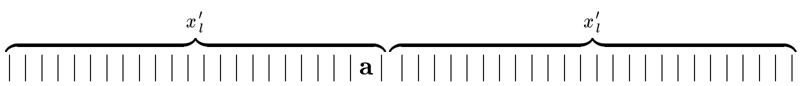
(i) *move-right* is defined as

$$\mathit{move\text{-}right}_{\mathbf{a}} := x'_l = 2x_l + \mathbf{a} \wedge x'_r = x_r \text{ div } 2.$$

Consider the following situation:



where the head scans the symbol b, the tape to the left is coded in x_l, and the tape to the right containing b is coded in x_r. As the head moves to the right, the pair (x'_l, x'_r) must code the new tape content as follows.



Thus, the position containing the b must be removed from the right side coded in x_r and x'_r resp., it must be added to the left side coded in x_l and x'_l resp., and, the symbol $\mathbf{b}$ must be replaced by $\mathbf{a}$. This is done by setting x'_r to x_r div 2, i.e., removing the position entirely, and setting x'_l to $2x_l + \mathbf{a}$. The same ideas are used in the next two formulae taking care of the head moving to the right or not moving at all.

(ii) *don't-move* is defined as

$$\textit{don't-move}_{\mathbf{a}} := x'_l = x_l \wedge x'_r = (x_r \text{ div } 2) \cdot 2 + \mathbf{a}.$$

(iii) *move-left* is defined as

$$\textit{move-left}_{\mathbf{a}} := x'_l = x_l \text{ div } 2 \wedge x'_r = ((x_r \text{ div } 2) \cdot 2 + \mathbf{a}) \cdot 2 + (x_l \bmod 2).$$

We are now ready to state the definition of the formula *compute*.

$$\begin{aligned} \text{compute} := {} & t' = t + 1 \wedge \exists c\,(c = x_r \bmod 2) \wedge \\ & \textstyle\bigvee_{(q,a)\to(q',a',m)\in\delta} s = \mathbf{q} \wedge s' = \mathbf{q}' \wedge c = \mathbf{a} \wedge p' = p + \mathbf{m} \wedge \\ & \qquad ((m = 1 \wedge \textit{move-right}_{\mathbf{a}'}) \vee \\ & \qquad (m = 0 \wedge \textit{don't-move}_{\mathbf{a}'}) \vee \\ & \qquad (m = -1 \wedge \textit{move-left}_{\mathbf{a}'})). \end{aligned}$$

We now turn to the definition of the formula init. Again we first need some auxiliary formulae. Recall from above that there is a formula *enc* which defines a representation $enc(S) \subseteq \mathbb{R}^{k(k+1)}$ of the input $S \subseteq \mathbb{R}^k$ by a finite set of tuples of points. We use this to define the initial configuration by letting the Turing-tape contain this set of tuples of points. To simplify notation, we assume an encoding $S' := enc(S)$ of the input S by a finite set of natural numbers, i.e., the tuples of points reduce to 1-tuples of points in $\mathbb{R}^1$ and, further, the coordinates of this "points" are natural numbers. Observe that such an encoding does not correspond to any possible input relation S but the extension to points and tuples of higher dimension and to rational coordinates will be straight forward. We comment on this below.

The formula init is defined as

$$\text{init}(x'_l, x'_r, t', s') := t' = 0 \wedge s' = \mathbf{q_0} \wedge x'_l = 0 \wedge \text{start}(x'_r),$$

where the formula start is defined as

$$\begin{aligned} \text{start}(x) := {} & \exists p = \max(S') \wedge \\ & [\mathrm{TC}_{p,x;p',x'} \left(\begin{pmatrix} p = x = -1 \wedge p' = \min(S') \wedge \\ \text{append}(1, p', x') \end{pmatrix} \vee \begin{pmatrix} p \in S' \wedge p' \in S' \wedge p' = \text{succ}(p) \wedge \\ \text{append}(x, p', x') \end{pmatrix} \right)](-1, -1, x', p). \end{aligned}$$

Here the formulae *max*, *min*, and *succ* are defined with respect to the lexicographical ordering of the points in the encoding S' and the formula append(x', p, x)

defines x' to code the tape inscription obtained from the inscription coded in x with the bit representation of p being appended at the end. It is defined as

$$\begin{aligned}\text{append} := x' = p' + x \wedge \\ \exists c\,[\mathrm{TC}_{x,p,x',p'} \begin{pmatrix} p' = 2 \cdot p \wedge x' = x \text{ div } 2 \wedge x > 0 \wedge \\ (\exists \hat{p} \in S'\, x \leq \hat{p}) \end{pmatrix}](x, p, 0, c).\end{aligned}$$

The formula first shifts the bit representation of the point p as many bits to the right as the number of bits needed for representation of x and stores the result in c. Then it simply adds x to c and gets the desired bit representation in x'. The part that might cause confusion is the conjunct $(\exists \hat{p} \in S'\, x \leq \hat{p})$. This is unnecessary for the computation of x' but guarantees that the evaluation of the formula terminates. This is achieved by binding the values for x by the largest point in the encoding S'. As S' is finite, the process of building up the transitive closure must be finite as well.

As mentioned above, the case that the encoding S' is unary does not happen for any input relation. Also it is unlikely, that the points in the encoding all have natural coordinates. But the formula can easily - although with a huge overhead in notation - be extended to rational numbers and encodings of higher arity. Termination of the evaluation process is also guaranteed for the general case, as all the computations needed to encode the input can be bounded by the values of points in the finite set S'.

Finally, we have to decode the result of the computation. For this we can use the PFOL-formula *decode* mentioned above. Further, we need some preprocessing to decode the output of the machine given in one single number x into tuples of points. But the inductions involved can all be bounded by the number x coding the output of the Turing machine.

Now, the proof of the following lemma is straight forward.

Lemma 10 *Let f be a query on constraint databases over $(\mathbb{R}, <, +)$ and let M be a Turing-machine computing it. Let $f_\varphi := op_{\varphi_M}$ be the function assigned by the operational semantics to the formula φ_M as constructed above. Then, for each database $\mathfrak{B} := ((\mathbb{R}, <, +), \sigma)$,*

(i) M halts on input $\mathfrak{B}$ if, and only if, $f_\varphi(\mathfrak{B})$ is defined, and
(ii) $f_\varphi(\mathfrak{B})$ defines the same set of elements as represented by the output of M on $\mathfrak{B}$.

Thus we have shown the following theorem.

Theorem 11 *Under the operational semantics defined above, FO(TC) defines exactly the partially computable queries on constraint databases over $(\mathbb{R}, <, +)$.*

The proof of the theorem also yields a negative answer to further decidability questions. For instance, one might ask whether it is decidable for a given FO(TC)-formula φ and a first-order formula ψ if for all databases $\mathfrak{B}$ such that $\mathrm{mod}_\varphi(\mathfrak{B})$ is defined $\mathrm{op}_\psi(\mathfrak{B}) \subseteq \mathrm{op}_\varphi(\mathfrak{B})$. Using the proof given above, it is an easy exercise to reduce the halting problem for Turing machines to this question, thus proving it undecidable.

Corollary 12 *Let* φ *be a FO(TC)-formula and* $\psi \in \mathrm{FO}$. *It is undecidable, whether for all databases* $\mathfrak{B} \in \mathrm{CDB}(\mathbb{R}, <, +)$ *such that* $\mathrm{mod}_\varphi(\mathfrak{B})$ *is defined,*

$$\mathrm{op}_\psi(\mathfrak{B}) \subseteq \mathrm{op}_\varphi(\mathfrak{B}).$$

3.3 Completeness of stratified Datalog and least fixed-point logic

Clearly, FO(SFP) is more expressive than FO(TC). Thus, Theorem 11 generalises to FO(SFP) and FO(LFP) in the sense that each partially computable query can be defined in these logics. However, a bit care has to be taken on whether the formulae terminate in the cases where the query is computable. Let $\varphi := [\mathrm{TC}_{\overline{x},\overline{y}}\psi(\overline{x},\overline{y})](\overline{u},\overline{v})$ be a FO(TC)-formula. Then φ can inductively be translated to the equivalent FO(SFP)-formula $\varphi^* := [\mathrm{FO(SFP)}_{R,\overline{x},\overline{y}}\psi^*(\overline{x},\overline{y}) \vee \exists\overline{z}\, R\overline{x}\overline{z} \wedge \psi^*(\overline{z},\overline{y})](\overline{u},\overline{v})$. However, under the standard operational semantics, this formula might not terminate although, given the operational semantics above, the FO(TC)-formula might. To avoid this we recursively translate formulae φ as above to $\varphi^* := [\mathrm{FO(SFP)}_{R,\overline{x},\overline{y}}(\overline{x} = \overline{u} \wedge \psi^*(\overline{x},\overline{y})) \vee \exists\overline{z}(R\overline{x}\overline{z} \wedge \psi^*(\overline{z},\overline{y}))](\overline{u},\overline{v})$. This closely resembles the operational semantics we used for FO(TC) and thus guarantees termination of the formulae.

3.4 Existential Fixed-Point Logic

In the previous sections we have seen that FO(TC), FO(SFP), and FO(LFP) all express the same class of partially recursive queries. Regarding existential fixed-point logic (FO(EFP)), it can easily be shown that this logic is much weaker than the other three. In fact, there are even first-order definable queries that are not expressible in existential fixed-point logic. An example is the boolean query that is true for all databases which are bounded, i.e. where there is a number c such that there is no point in the database with an coordinate greater than c. This can easily be expressed in first-order logic. As it is known that $\mathrm{FO} \cap \mathrm{FO(EFP)}$ is exactly the class of positive existential first-order formulae and that these formulae are preserved under extensions of the structure, it is an easy observation that this query cannot be expressed in existential fixed-point logic.

4 Dense linear orders

In this section we consider dense linear order databases, e.g. constraint databases over $(\mathbb{R}, <, +)$. Fixed-point logics on this class of databases have been studied in [BST98,GK99,Kre99] where it is shown that questions about fixed-point queries on dense order databases can be reduced to the corresponding questions on finite databases. In particular, in [GK99] it has been shown that for all dense linear order database $\mathfrak{B}$ there is a finite ordered database $\mathrm{inv}(\mathfrak{B})$ with universe B, called the *invariant of* $\mathfrak{B}$ such that

- there is a function $\widehat{\pi}$ from finite subsets $S \subseteq B$ to FO(LFP)-formulae over $(\mathbb{R}, <, +)$ and
- for each FO(LFP)-formula φ on $\mathfrak{B}$ there is a FO(LFP)-formula φ' on $\text{inv}(\mathfrak{B})$

with the property that if $S = \varphi'(\text{inv}(\mathfrak{B}))$ is the result of the evaluation of φ' in the invariant of $\mathfrak{B}$ and $P := \{\overline{a} : (\mathbb{R}, <, +) \models \widehat{\pi}(S)\}$ is the set of elements satisfying the formula $\widehat{\pi}(S)$, then

$$R = \varphi(\mathfrak{B}),$$

where $\varphi(\mathfrak{B})$ denotes the set of tuples satisfying φ in $\mathfrak{B}$.

Now, by the results mentioned in the introduction, it follows that the formula φ' on the finite ordered database is equivalent to a formula φ^* in stratified fixed-point logic. To obtain a stratified fixed-point formula equivalent to the original query φ we have to transform φ^* back to a formula over $\mathfrak{B}$. It follows immediately from the results proved in [GK99] that there is a stratified fixed-point formula ψ over $\mathfrak{B}$ defining the relation R as defined above.

Thus we have shown the following theorem.

Theorem 13 *Stratified fixed-point logic and fixed-point logic have the same expressive power on the class of finitely representable structures over the real line* $(\mathbb{R}, <)$.

5 Conclusion

In this paper we compared various fixed-point logics with respect to the fraction of partially computable queries on linear constraint databases they define. For this, we first had to equip the logics with an operational semantics, which allowed us to speak about computability of queries defined by these logics. We then showed that already transitive-closure logic is expressive enough to define all partially recursive queries on linear constraint databases. Thus, with respect to this benchmark, transitive-closure, least, and stratified fixed-point logic are equivalent. As mentioned in the introduction, this is contrary to the relationship of the logics in terms of absolute definability, i.e., where there are no restrictions on the class of queries under consideration.

The motivation for choosing the class of partially recursive queries as benchmark comes from the usage of fixed-point logics as query languages, where non-recursive queries are of no practical interest.

Acknowledgement I want to thank Jan Van den Bussche and Floris Geerts for the many helpful discussions we had on the topics of this paper and especially for pointing out the importance of the precise operational semantics used in the proofs. I also want to thank Dietmar Berwanger for the time he spent on the details of the proofs and the many helpful comments he made.

References

[BST98] O. Belegradek, A. Stolboushkin, and M. Taitslin. Extended order-generic queries. *Annals of Pure and Applied Logic*, 1998. To appear.

[DG] A. Dawar and Y. Gurevich. Fixed-point logics. *Bulletin of Symbolic Logic*. to appear.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.

[GK97] S. Grumbach and G. M. Kuper. Tractable recursion over geometric data. In *Principles and Practice of Constraint Programming*, number 1330 in LNCS, pages 450 – 462. Springer, 1997.

[GK99] E. Grädel and S. Kreutzer. Descriptive complexity theory for constraint databases. In *Computer Science Logic*, number 1683 in LNCS, pages 67 – 82. Springer, 1999.

[GK00] F. Geerts and B. Kuijpers. Linear approximation of planar spatial databases using transitive-closure logic. In *Proceedings of the 19th ACM Symp. on Principles of Database Systems (PODS), 2000*, pages 126–135. ACM Press, 2000.

[KKR90] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 299–313, 1990.

[KKR95] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and Systems Sciences*, 51:26–52, 1995. (An extended abstract appeared in the Proceedings of PODS'90).

[KLP00] G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer, 2000.

[Kol91] P. Kolaitis. The expressive power of stratified logic programs. *INFORMATION AND COMPUTATION*, 90:50–66, 1991.

[Kre99] S. Kreutzer. Descriptive complexity theory for constraint databases, 1999. Diplomarbeit an der RWTH Aachen.

[Kre01] S. Kreutzer. Query languages for constraint databases: First-order logic, fixed-points, and convex hulls. In *Proceedings of the 8th International Conference on Database Theory (ICDT)*, number 1973 in Lecture Notes in Computer Science, pages 248–262. Springer, 2001.

[Van99] L. Vandeurzen. *Logic-Based Query Languages for the Linear Constraint Database Model*. PhD thesis, Limburgs Universitair Centrum, 1999.