Guarded Quantification in Least Fixed Point Logic

Gregory McColm* Department of Mathematics University of South Florida Tampa, FL 33620 (813) 974-9550, fax (813) 974-2700 mccolm@math.usf.edu URL http://www.math.usf.edu/~mccolm

May 14, 2002

Abstract

We develop a variant of Least Fixed Point logic based on First Order logic with a relaxed version of guarded quantification. We develop a Game Theoretic Semantics of this logic, and find that under reasonable conditions, guarding quantification does not reduce the expressibility of Least Fixed Point logic. But guarding quantification increases worst-case time complexity.

^{*}This research was partially supported by NSF grant CCR 940-3463.

Contents

1	Introduction		
	1.1	The Notion of Guards	4
	1.2	Outline of the Paper	6
2	Gua	ards in Logics	7
	2.1	Guarded Quantification	9
	2.2	Guard versus database relations	13
	2.3	Least fixed point logic with guards	17
	2.4	Distance in Guard Relations	20
3	Gar	ne Programs	26
	3.1	Playing the Game	26
	3.2	Winning the Game	30
	3.3	The Topology of Games	34
	3.4	Uniform Connectivity and Games	39
4	Usi	ng Game Programs	43
	4.1	Non-Recursive Games and First Order Logic	44
	4.2	Recursive Games and LFP Logic	45
	4.3	Least fixed point logic	50
		4.3.1 (Sub)Programs for Simulating Quantification	53
		4.3.2 The Simulation Works	57
	4.4	Closure Under Negation	60
5	Exc	elsior	61

1 Introduction

For over two decades, finite model theorists have looked at relational databases as relational structures. After all, if relational databases were relational structures, then the logics of finite model theory should be helpful in understanding relational databases. This approach seemed justified by the identification of certain logics (assisted by various gadgets) with traditional computational complexity classes, e.g., NP with Σ_1^1 in [F74], PTIME with the Least Fixed Point logic (with successor) of [Mo74] and [AhU79] in [I86] and [V82], and the various additional identifications in [I87] and elsewhere. In a certain sense, the logics of finite model theory seem to describe the power of database query systems fairly well. But there are complications.

The primordial logic of finite model theory is First Order logic (FO), the basic logic of classical model theory. In addition, positive existential FO logic is the logic corresponding to Structured Query Language (SQL) — and if you added negation to SQL, the corresponding logic is FO itself (see [AbiHV95]). But there are at least two problems with FO from an algorithmic point of view:

- There is a classical problem: FO is unable to express recursion, and (thus) cannot express popular queries like reachability in graphs.
- As [Gu97] points out, quantification in FO logic is too easy compared with exhaustive searches through databases.

Notice that the second problem is not about the *power* of quantification — database query systems are usually allowed to search through an entire database, given time to do so — but that it is done in a single, innocuous step that too easily sweeps the complexities under the rug. It might be helpful to have a logic that captures the descriptive and computational complexity of database queries more precisely.

In this article, we will develop such a precise logic, and compare the time complexity of algorithms in this more precise logic with that of classical fixedpoint logic.

1.1 The Notion of Guards

In this article, we propose an approach for dealing explicitly with the the second problem (the complexity of quantification) and thus implicitly with the first (the need for recursion). We will develop a Least Fixed Point (LFP) logic using a system of guards similar to those of [Com83] and [AnBN96]. As LFP has recursion built into it, we will be able to deal explicitly with issues involving recursion. And as the guards provide a plausible restriction on quantification, we will be able to look at issues arising from such restrictions. In addition, we will follow the approach of [Mc95a] in using Hintikka type semantics for games to describe computations, and we will find that these "Eloise-Abelard" (or "Angel-Demon" or "Assertor-Denier") correspond to this guarded LFP logic.

Here are two motivating images.

I. The original motivating image of [AnBN96] is that of a universe of many worlds, appropriate for a modal-type logic. From a particular world, only some other worlds are accessible, and those worlds are the ones that a "guard relation" permits one to go to. The two basic quantifications of modal logic, " \Box = it is necessary that ..." and " \Diamond = it is possible that" can be represented as universal and existential quantifications, respectively, over the worlds accessible from the current world. This motivational image has been developed further in the work of [GraW99], etc.

Guards have been used in modal-type logics elsewhere, e.g., in the communication, process, and game logics. And we are very interested in games in this paper.

The connection between games and logic goes back to Peirce (see [Hil82]), but the idea was popularized by such works as [Hin72] (see [HiS97]), [Mo72] and [Ac75]. Suppose that we have a two-player game of perfect information, and we want to assert that one or the other player has a winning strategy. We can use the guards to indicate the legal moves, with additional relations to indicate whose turn it is, and, if the game is over, who has won. Then "Player \mathcal{E} has a winning strategy" is an assertion that can be expressed in a "game logic." "Game logics" were explored for their own sake in [Pa85], but also for the sake of studying validity in [Pl81], [He94], [BacW98, II.14], etc. and for the sake of studying communication and processes in [AbaLW89], [Mo91], [Mi99], etc. (see [FHMV95]). In this article, we will explore a framework for studying many kinds of two-player games of perfect information. In such a game, there might be several pieces to move, with one player trying to get the pieces into one kind of configuration, and the other trying to prevent the first player from winning. (Imagine a game of Chess, in which White wants to win, while Black would be satisfied with a draw.) Most board games have some kind of bounds on the legal moves, and we will use guard relations to bound the range of legal moves.

As the games may go on indefinitely, we will be spending much of our time on fixed points. Assertions about who wins in a two-player game in a Modal-type game logic can be naturally expressed in fixed point logics (see, e.g., [BarM96, IV.12]). And this leads us closer to another motivating image.

II. Imagine a simple-minded data storage device, in which we could store, say, a digraph. Imagine that hashing is not an issue. There are at least two naive ways one could store the digraph.

- We might store the digraph by putting the names of the vertices into registers of the device, and assigning, for each arc, a register with two pointers (one for the source and one for the sink).
- We could store an edge by having a pointer to the sink's register in the the source's register. This approach would take less space than the first, but would impose an upper bound on the outdegrees of the vertices.

(There are other options as well.) Then we naively imagine the CPU following pointers as it navigates through the database. The relation consisting of all these navigational pointers might be called the *guard relation*.

Now for a complication. One problem with the CPU metaphor is that navigation through the digraph requires the digraph's own arcs, which might be distributed unhelpfully (e.g., if the digraph is not connected, one could be cut off from some of the digraph). In some of the extant guarded quantification literature, there are several relations, and in this paper we explicitly distinguish *relations used for computations* (i.e., *guard relations*) from *database relations* (i.e., relations representing the *inputted structure*). Returning to the game metaphor, we have some relations — the guard relations — used for moves, and some relations — the database relations — used at the end of the game to determine who won. In both metaphors, the guard relations do not represent any part of the input structure in itself; they are used for computations only.

In the first, modal-type, motivational image, the guard relation is an integral part of the structure. This is typical of the usual versions of "guarded quantification." In the second, database-style, motivational image, the guard relation must merely satisfy certain utilitarian criteria (connectivity, perhaps it enables some arithmetic, etc.).

The logical device that we will use to describe this "guarding" is a generalization of the "bounded quantification" of classical recursion theory (see [Ro67]) and descriptive set theory (see [Mo80]). Such restrictions on quantification entered finite model theory with the "connected quantification" of [Com83], which investigated the special preservation properties of FO sentences whose guards were actually edge relations of graphs. We believe that an investigation of guarded quantification, looking at various kinds of guards, will enable us to dissect the notion of quantification itself. We are separating out the properties of quantification over immediate neighbors from "global" quantification (iover the entire structure), and we hope to find which properties of quantification are preserved.

In this paper, we will look at such a variant of what [AnBN96] called the Third Fragment: a 'guarded' version of FO itself. This notion of "guarded quantification" will be more relaxed than the one currently most popular in finite model theory, as the motivation is different. (Thus it might be called "weakly guarded" quantification.) And we will develop a guarded Least Fixed Point logic (a guarded version of the logic "FO + LFP"), in which some of the work of the quantification is represented by recursions on the guarded quantifications.

1.2 Outline of the Paper

In this paper, we will start with a long Section 2 on how our guarded quantification works, and a description of guarded LFP logic. We will also look at a technical difficulty that arises when the guard relation is of arity > 2.

In Section 3, we will recast guarded quantification logic in terms of games, using

the game-theoretic framework of [Mc95a]; this system is a descendent of [Mo72] and [HaK84], but belongs in the family of games described by the Game Theoretic Semantics developed by Hintikka (see [HiS97]). In such a game, there are two players, whom we may romantically call "Eloise" and "Abelard": Eloise is trying to prove that a certain statement is TRUE on a given structure, while Abelard is trying to prove that it is FALSE. The rules of a game make up a "game program", and we develop a Datalog-like language of game programs similar to that of [Mc95a], and we find that we can precisely capture guarded First Order logic (or guarded LFP logic) with these game programs, depending on whether recursion is permitted.

We will continue this exploration in Section 4 by comparing logics with game programs. Thus we get a guarded version of the theorem of [HaK84]:

Theorem 4.2 Guarded FO + positive LFP logic captures the guarded game logic.

In addition, by a coarse measure, there is no change in expressive power:

Theorem 4.3 Assuming an innocuous assumption, on finite structures, guarded FO + (positive) LFP and unguarded FO + (positive) LFP have the same expressive power.

However, we will find that using guards increases time complexity and, in a manner that will prove irritating in [Mc*], space complexity as well.

2 Guards in Logics

In this section, we will define the structures that we will be working on, and the guarded logics that we will be working with. We will also discuss some of the basic properties of these logics on these structures. We start with a few basic definitions, mostly following the nomenclature of [Mo74]; see also [EbF95] and [I99].

Definition 2.1 A relational database is a tuple $\mathfrak{D} = \langle D; R_1, \ldots; c_1, \ldots \rangle$, where $D = |\mathfrak{D}|$ is some set (the domain of \mathfrak{D} , of cardinality $||\mathfrak{D}||$), where each R_i is a relation on D and each c_k is a constant in D.

More logical papers call these semantic tuples (relational) *elementary structures*. We will resort to calling these things *structures* when we don't want to think of them as

databases (in the sense that what they contain is not so much information as navigational guidance for computation). Notice that we use semicolons to separate the domain from the relations and the relations from the constants. We will tend to use capital fraktur font (\mathfrak{M} , \mathfrak{D} , \mathfrak{R} , ...) for structures, capital italic font for sets and relations, and lowercase italic font for elements and both capital and lowercase italic font for integers. Let $[n] = \{1, 2, \ldots, n\}$; let $[0] = \emptyset$.

Definition 2.2 A schema is a tuple $\sigma = \langle (R_1, a_1), \ldots; c_1, \ldots \rangle$, where each R_i is a symbol standing for a relation of arity a_i , and each c_k is a constant symbol standing for an element. We say that a database $\mathfrak{D} = \langle D; R_1^{\mathfrak{D}}, \ldots; c_1^{\mathfrak{D}}, \ldots \rangle$ is of schema $\sigma = \langle (R_1, a_1), \ldots; c_1, \ldots \rangle$ if:

- for each $i, R_i^{\mathfrak{D}} \subseteq D^{a_i}$, and
- for each $j, c_j^{\mathfrak{D}} \in D$.

We say that \mathfrak{D} is a σ -structure.

More logical papers call these syntactic tuples *signatures* rather than *schemas*. We will often look at classes of databases of a common schema.

Notice that in Definition 2.1, each "R" was a relation, while in Definition 2.2, each "R" was a symbol for a relation: applied to the database \mathfrak{D} , "R" is a symbol that can represent the relation $R^{\mathfrak{D}}$. Usually, we will be sloppy, and not distinguish between symbols and the objects that they represent.

Remark 2.1 In "real life," relational databases are multi-sorted structures, i.e., they have many domains. Thus we would have a database $\mathfrak{D} = \langle D_1, \ldots; R_1, \ldots; c_1, \ldots \rangle$, where each D_h is a domain, each R_i is a subset of some product of the domains, and each constant c_j is an element of a domain. The schema would be changed accordingly. And in "real life," many databases also have built in functions. But we will avoid such complexities in this paper, and merely observe that the definitions and theorems can all be generalized in some natural way. For an introduction to the foundations of database theory, see the article [Ka91], the book [AbiHV95], or the tome [U88, 89]. In this paper, we will want to add relations, functions and constants to databases (and corresponding symbols to database shema).

Definition 2.3 Given a σ -structure $\mathfrak{M} = \langle D; R_1, \ldots; c_1, \ldots \rangle$, and given a relation $S \subseteq D^s$, call $(\mathfrak{M}, S) = \langle D; R_1, \ldots, S; c_1, \ldots \rangle$ an expansion of \mathfrak{M} by adding the relation S, and say that it is of the expanded schema $(\sigma, (S, s))$.

The nomenclature for adding a constant is similar, as is the nomenclature for adding a tuple of relations and constants.

2.1 Guarded Quantification

In this section, we define guarded quantification, and the corresponding first order logic.

In this paper, the boolean operators \land , \lor and \neg behave as usual. Our version of "guarded quantification" is as follows.

First, we expand the database schema σ by adding one or more guard relations, which are of arity at least 2, and some guard constants, as follows.

Definition 2.4 A guard schema is a tuple $\rho = \langle (P_1, a'_1), \ldots; d_1, \ldots \rangle$ such that:

- each P_i is a relation symbol, and each a'_i is a positive integer greater than 1: we call a'_i the arity of the relation symbol P_i ; and
- each d_i is a constant symbol.

A guard structure of guard schema ρ is a tuple $\mathfrak{R} = \langle D; P_1^{\mathfrak{R}}, \ldots; d_1^{\mathfrak{R}}, \ldots \rangle$, where $|\mathfrak{R}| = D$ is the domain of the structure, $P_i^{\mathfrak{R}} \subseteq D^{a'_i}$ for each i and $d_j^{\mathfrak{R}} \in D$ for each j.

Again, we will *usually* not distinguish between guard relations and guard relation symbols, between guard constants and guard constant symbols.

Remark 2.2 Notice that technically, there is no difference between a database schema and a guard schema: given a tuple out of context, we could not tell whether it was a database schema (i.e., whether it contained input information about the domain) or a guard schema (i.e., whether it contained computation guidance for the domain). Similarly, out of context, we cannot differentiate between database and guard structures. But if we keep them in context, we should be able to avoid confusion.

We put the database and guard schemas together so that we can apply them both to the same domains.

Definition 2.5 Let $\sigma = \langle (R_1, a_1), \ldots; c_1, \ldots \rangle$ be a database schema, and let $\rho = \langle (P_1, a'_1), \ldots; d_1, \ldots \rangle$ be a guard schema. Denote the joint database-guard schema of σ and ρ as:

$$(\sigma, \rho) = \langle (R_1, a_1), \dots, (P_1, a'_1), \dots; c_1, \dots, d_1, \dots \rangle$$

Let $\mathfrak{D} = \langle D; R_1^{\mathfrak{M}}, \ldots; c_1^{\mathfrak{M}}, \ldots \rangle$ be a σ -structure and $\mathfrak{R} = \langle D; P_1^{\mathfrak{R}}, \ldots; d_1^{\mathfrak{R}}, \ldots \rangle$ a ρ -structure, both of a common domain $|\mathfrak{D}| = |\mathfrak{R}| = D$. Then the joint database-guard structure of joint schema (σ, ρ) is the tuple

$$(\mathfrak{D},\mathfrak{R}) = \langle D; R_1^{\mathfrak{M}}, \ldots; c_1^{\mathfrak{M}}, \ldots; P_1^{\mathfrak{R}}, \ldots; d_1^{\mathfrak{R}}, \ldots \rangle.$$

We will often just call $(\mathfrak{D}, \mathfrak{R})$ a joint structure.

We will tend to denote database structures by the letter \mathfrak{D} , guard structures by the letter \mathfrak{R} , and joint structures by the letter \mathfrak{M} . Notice that we distinguish between the database relations and constants on the one hand, and the guard relations and constants on the other. On the other hand, we can consider (σ, ρ) as an expansion of σ , and thus $(\mathfrak{D}, \mathfrak{R})$ as an expansion of \mathfrak{D} , in the sense of Definition 2.3.

Definition 2.6 Let \mathcal{D} be a set of databases of a common database schema, and let \mathcal{R} be a set of guard structures of a common guard schema. Then \mathcal{R} is a guard system for \mathcal{D} if, for each $\mathfrak{D} \in \mathcal{D}$, there exists $\mathfrak{R} \in \mathcal{R}$ such that $|\mathfrak{D}| = |\mathfrak{R}|$.

We now describe how guarded quantification works. The guard constants $d_1^{\mathfrak{R}}, \ldots$ will provide us with places to begin to look during quantifications, while the guard relations $P_1^{\mathfrak{R}}, \ldots$ will bound our search. Here is the critical idea of guarded quantification. **Definition 2.7** Fix a domain D and an integer $a \ge 2$. Given an a-ary guard relation $P \subseteq D^a$ and an (a-1)-tuple $\mathbf{x} \in D^{a-1}$, and $a \ y \in D$, we say that \mathbf{x} accesses y through P if $P(\mathbf{x}; y)$.

Definition 2.8 We define guarded quantification as follows. If P is a guard relation, and \mathbf{x} a tuple of variables and guard constants, and if \mathbf{z} is a tuple of variables (including y) then

$$(\exists y: P(\mathbf{x}; y))\theta(\mathbf{z})$$

is the assertion that there exists y such that $P(\mathbf{x}; y)$ and $\theta(\mathbf{z})$ are both true. Similarly,

$$(\forall y: P(\mathbf{x}; y))\theta(\mathbf{z})$$

is the assertion that every y satisfying $P(\mathbf{x}; y)$ also satisfies $\theta(\mathbf{z})$.

Think of $(\exists y: P(\mathbf{x}; y))\theta(\mathbf{z}, y)$ as $\exists y[P(\mathbf{x}; y) \land \theta(\mathbf{z}, y) \text{ and think of } (\forall y: P(\mathbf{x}; y))\theta(\mathbf{z}, y)$ as $\forall y[P(\mathbf{x}; y) \to \theta(\mathbf{z}, y).$

The simplest class of guarded formulas are the guarded FO formulas:

Definition 2.9 The guarded FO formulas, which we denote by FO_* , are constructed as follows.

- **1.** The atomic formulas:
- If x is a variable and c is a database constant, then x = c is a FO_{*} formula. We will regard equality as both a database and a guard relation.
- If \mathbf{x} is a tuple of variables, and if R is a database relation symbol, then $R(\mathbf{x})$ is a FO_{*} formula.

(The idea is this: in order to ask if a tuple \mathbf{x} is listed in R, we must have accessed all of \mathbf{x} , and we do not assume that database constants are automatically accessible. However, given an accessed value, we can certainly ask if this is in fact the constant in question.)

2. Conjunctions, disjunctions, and negations are defined as usual to get: $\varphi \wedge \psi$, $\varphi \lor \psi$, and $\neg \varphi$.

3. For any FO_{*} formula ψ , and guard relation P, the formulas

 $(\exists y: P(\mathbf{x}'; y))\psi(\mathbf{x}'', y)$ and $(\forall y: P(\mathbf{x}'; y))\psi(\mathbf{x}'', y)$

are (guarded quantification) FO_{*} formulas (where \mathbf{x}' is a tuple of variables and guard constants, while \mathbf{x}'' is a tuple of variables). Notice that \mathbf{x}'' might be an empty tuple, so that if \mathbf{x}' consists of guard constants alone, we would have a nontrivial FO_{*} formula with no free variables; as usual, such a formula is called a *sentence*. In these formulas we say that y is *bound* by the quantification: variables unbound by any quantifications are *free*.

Remark 2.3 The above system is a relaxation of the guarded logics in the literature. Here are the major differences:

- We distinguish between guard relations and database relations (and those that are both). We similarly distinguish between guard constants and database constants.
- In quantifications (Qy: P(x; y))ψ(x'; y), we permit free variables in x' that do not occur in x. This bit of permissiveness permits us to express notions like "the graph is a 4-clique." (In fact, as we shall see, this permissiveness permits a great deal.)
- Some guarded quantification logics have, instead of a single atomic formula serving as a guard, a conjunction of several atomic formulas: e.g.,

$$(Qy: P_1(\mathbf{x}_1; y), \ldots, P_k(\mathbf{x}_k; y))\theta(\mathbf{z}).$$

(The usual restriction is that the variables of \mathbf{z} are all among $\mathbf{x}_1, \ldots, \mathbf{x}_k, y$, and the interpretation is, if $Q = \exists$, that there is a y satisfying $P_1(\mathbf{x}_1; y) \land \cdots \land P_k(\mathbf{x}_k; y) \land \theta(\mathbf{z})$ — with a similar interpretation if $Q = \forall$. The nice thing about permitting these conjunctions is that the restriction on the free variables of \mathbf{z} no longer causes irritating number-of-variables problems: e.g., it is now possible to represent "the graph is a 4-clique.").

It is possible to capture queries defined by these conjunctive guarded quantifications using the system of this paper. Define,

$$(\exists y: P_1(\mathbf{x}_1; y), \dots, P_k(\mathbf{x}_k; y)) \theta(\mathbf{z})$$

$$\equiv (\exists y_1: P_1(\mathbf{x}_1; y_1)) \cdots (\exists y_k: P_k(\mathbf{x}_k; y_k)) [y_1 = \dots = y_k \land \theta(\mathbf{z})]$$

and

$$(\forall y: P_1(\mathbf{x}_1; y), \dots, P_k(\mathbf{x}_k; y)) \theta(\mathbf{z})$$

$$\equiv (\forall y_1: P_1(\mathbf{x}_1; y_1)) \cdots (\forall y_k: P_k(\mathbf{x}_k; y_k)) [y_1 = \dots = y_k \to \theta(\mathbf{z})]$$

We can define satisfaction in the usual way: given a joint structure \mathfrak{M} of joint schema κ , and given a FO_{*} sentence θ also of joint schema κ , " $\mathfrak{M} \models \theta$ " means that \mathfrak{M} satisfies θ .

There is a useful fact (which we will need) about FO formulas that is also true of FO_{\star} formulas: we can push negations down to the atomic level.

Proposition 2.1 Every FO_{*} formula θ is equivalent to a FO_{*} $\hat{\theta}$ whose negations modify only atomic subformulas.

Idea of proof. This merely involves repeated applications of De Morgan's Laws:

$$\neg(\varphi \lor \psi) \equiv \neg \varphi \land \neg \psi,$$
$$\neg(\varphi \land \psi) \equiv \neg \varphi \lor \neg \psi,$$
$$\neg(\exists y : P(\mathbf{x}'; y))\psi \equiv (\forall y : P(\mathbf{x}'; y))\neg \psi,$$
$$\neg(\forall y : P(\mathbf{x}'; y))\psi \equiv (\exists y : P(\mathbf{x}'; y))\neg \psi.$$

2.2 Guard versus database relations

Sometimes we may want to use the database relations themselves as guards. Imagine that we are given a graph and some vertices in the graph: we may want to quantify over the neighbors of the given vertices. There is nothing to prevent us from using some of the database relations as guard relations, as is done in [Com83] and [AnBN96] and, indeed, most of the literature. When we want to do this, we just list the relation symbol twice in the joint signature (σ, ρ) : once in σ and once in ρ . But in this article, we will permit guard relations that are not database relations, and vice versa, with the proviso:

We will take equality to be a guard relation as well as a database relation.

- (Remember: the database contains the information while the guards guide computation.) Thus we have two extreme situations to deal with:
 - 1. All the guard relations and constants could be database relations and constants.
 - 2. No guard relations (other than equality) or constants would be database relations or constants.

(And, of course, we could have *some* guard relations and constants be database relations and constants.) As mentioned Subsection 1.1, (1) is the usual situation in the literature, while (2) is the usual situation in this paper. As an example of the situation (1), consider the following.

Example 2.1 Let \mathfrak{G} be a connected graph, with edge relation Edge. Let Edge be the guard relation as well, and imagine that there is one guard constant d. Then to determine if \mathfrak{G} is a clique, i.e., if

$$\mathfrak{G} \models \forall x \forall y (x = y \lor \mathrm{Edge}(x, y)),$$

determine the truth value of the FO_{\star} sentence

$$(\forall x: \operatorname{Edge}(d, x))(\forall y: \operatorname{Edge}(x, y))(\forall z: d = z)\operatorname{Edge}(z, y).$$

Notice that if \mathfrak{G} is not connected, all this says is that d's component in \mathfrak{G} is a clique.

We now turn to the situation where there are guard relations that are not database relations, and vice versa.

We should also note that the guard relations are themselves definable using quantification with guards.

Proposition 2.2 Guard relations and constants are expressible in FO_{\star} , as follows.

For each guard relation P, there is a FO_{*} formula θ_P such that for all joint structures $\mathfrak{M}, \mathfrak{M} \models \forall \mathbf{x} [P(\mathbf{x}) \longleftrightarrow \theta_P(\mathbf{x})].$

For each guard constant d, there is a FO_{*} formula θ_d such that for all joint structures $\mathfrak{M}, \mathfrak{M} \models \forall x (x = d \longleftrightarrow \theta_d(x)].$ **Proof.** Define

$$\theta_P(\mathbf{x}; y) \equiv (\exists z: P(\mathbf{x}; z))(y = z)$$

which is a FO_{\star} formula. For any guard constant d,

$$\theta(x) \equiv (\exists y: y = d)(y = x)$$

is a FO_ \star formula. \blacksquare

Let's look at some examples.

Example 2.2 The successor relation can be used as a guard. Let \mathcal{D} be a set of databases of a common schema σ . We have a guard schema consisting of a binary relation succ and a constant 0. If $\mathfrak{D} \in \mathcal{D}$, then a guard structure of schema $\rho = \langle \text{succ}; 0 \rangle$ will be a structure $\mathfrak{R} = \langle |\mathfrak{D}|; \text{succ}^{\mathfrak{R}}; 0^{\mathfrak{R}} \rangle$ such that:

- For each x ∈ |𝔅|, there exists at most one y such that (𝔅,𝔅) ⊨ succ(x, y): for all but one x, such a y exists. Let s be a partial function such that for each x ∈ |𝔅|, s(x) is the unique y such that 𝔅 ⊨ succ(x, y) − if s(x) exists.
- 2. For each $x \in |\mathfrak{D}|$, there exists an n such that n iterations of s from 0 produces x, i.e., $s^{n}(0) = x$. Thus the only $x \in |\mathfrak{D}|$ such that s(x) does not exist is the succ-maximum element of \mathfrak{D} .

For example, suppose that \mathcal{D} is a set of structures whose domains are the sets [n], $n = 1, 2, 3, \ldots$

• One guard system for a built in successor would be the set of structures

$$\langle [n]; \mathtt{succ}_n; 1 \rangle,$$

for each positive integer n, where $\operatorname{succ}_n(x, y) \equiv x + 1 = y$ for all x < n.

• Another guard system, for a successor "independent of" the given structure, might be the set of structures

$$\langle [n]; \mathtt{succ}_{n,\tau}; \tau(1) \rangle,$$

for each positive integer n and permutation τ : $[n] \to [n]$, where $\operatorname{succ}_{n,\tau}(x,y) \equiv \tau(x) + 1 = \tau(y)$ for all $x, \tau(x) < n$.

Expanding on Example 2.2, the representation of PTIME in [Mo83] — see [Mc89] — as the 'recursive' queries over structures $\langle \{0, \ldots, n-1\}; R, \texttt{succ}, \texttt{pred}; 0 \rangle$, R a relation encoding the structure, might be regarded as an example using successor and predecessor as guard relations (and 0 as the guard constant — if we use =0 rather than =). But there are other examples as well.

Example 2.3 Consider a database whose domain is of 2^m elements. We could take an *m*-hypercube as a symmetric quard relation, with one of the vertices as a quard constant.

This sort of thing works for databases of very special sizes. How about:

Example 2.4 Fix a natural integer $n = m_1 m_2 \cdots m_k$, and suppose that \mathfrak{D} was a database of n elements. We could apply a directed toroidal k-dimensional grid as the guard relation as follows. Each element of $|\mathfrak{D}|$ might be identified with a tuple $(\alpha_1, \ldots, \alpha_k) \in \mathbb{N}^k$, where $\alpha_j < m_j$ for each $j \in [k]$ and $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$. The guard relation would allow quantifications from a vertex $(\alpha_1, \ldots, \alpha_k)$ to a vertex $(\beta_1, \ldots, \beta_k)$ iff for some $i, j \neq$ $i \implies \alpha_j = \beta_j$, while $\alpha_i + 1 \cong \beta_i \mod m_i$. As a guard constant, we use the vertex $(0, \ldots, 0)$.

The following is a chestnut: see [Mo83].

Example 2.5 The guard relation might be a binary tree, where each vertex v has at most two successors (which might be identical) $\mathfrak{l}(v)$ and $\mathfrak{r}(v)$, and quantification from v means checking $\mathfrak{l}(v)$ and $\mathfrak{r}(v)$ only. Thus we get two guard relations left and right, being the graphs of the functions \mathfrak{l} and \mathfrak{r} respectively. The guard constant would be the root of the tree.

This last example leads us to the special case: suppose that the guard relation is a function. Returning to the motivation of a computer with tuples stored in registers and guards represented by pointers, this certainly makes sense: a register can store only so many pointers. So we could imagine a collection of pointer functions f_1, \ldots such that given $\mathbf{x}, f_i(\mathbf{x})$ gives you the *i*th pointer on \mathbf{x} 's register.

2.3 Least fixed point logic with guards

We will work mostly with guarded versions of the LFP logics of [Mo74] and [AhU79]. These are extensions of FO logic, but here we use guarded quantification. We construct guarded positive LFP Logic (which we denote FO_{\star} + pos LFP) and unguarded positive LFP Logic (which we denote FO + pos LFP) as follows.

An operative system of formulas is a sequence $\varphi = \varphi_0, \ldots, \varphi_{\nu}$ of second order formulas

$$\varphi_i(x_{i,1},\ldots,x_{i,r_i},S_0,\ldots,S_\nu), \qquad i=0,\ldots,\nu,$$

where, for each j, S_j ranges over r_j -ary relation variables. (We permit 0-ary relation variables, i.e., variables ranging over TRUE and FALSE.)

Now suppose that each of these formulas φ_i is monotone, i.e., on any joint structure \mathfrak{M} , if $S_j \subseteq T_j$ for each j, then for any \mathbf{x} from \mathfrak{M} , and any i,

$$\mathfrak{M} \models \varphi_i(\mathbf{x}, S_0, \ldots, S_{\nu}) \rightarrow \varphi_i(\mathbf{x}, T_0, \ldots, T_{\nu}).$$

Then we can carry out a recursion (as described in [Mo74] and [Mo83]) as follows. The Least Fixed Point of the system φ is constructed by setting $\varphi_i^0 = \emptyset$ for each *i*, and then, for each *n*, we define

(2.1)
$$\varphi_i^{n+1}(\mathbf{x}) \equiv \varphi_i(\mathbf{x}, \varphi_0^n, \dots, \varphi_\nu^n),$$

and it follows from an induction on n that

so that on a finite structure \mathfrak{M} , there exists an n such that for all j, $\mathfrak{M} \models \varphi_j^n = \varphi_j^{n+1}$. Then if $\varphi_i^{\infty} = \bigcup_n \varphi_i^n$ for each i, the tuple $(\varphi_0^{\infty}, \ldots, \varphi_{\nu}^{\infty})$ is the 'Least Fixed Point' of the system φ . If all the φ_i are \mathcal{L} -expressible for some logic \mathcal{L} , we say that φ_0^{∞} is $(\mathcal{L} + \text{pos LFP})$ -expressible.

(We will tend to use S_0, \ldots, S_{ν} or T_0, \ldots, T_{ν} as relation-valued recursion variables, or as relations, depending on the context. We will also use the notation $\varphi^{\kappa} = (\varphi_0^{\kappa}, \ldots, \varphi_{\nu}^{\kappa})$.)

Incidentally, this is why we are sticking to finite structures throughout most of this paper: in infinite structures, these recursions can continue for transfinitely many iterations. We will use this number-of-iterations measure as a time complexity measure. (More machine-oriented time-complexity issues in guarded quantification are discussed in [GraW99].)

Definition 2.10 Suppose that we have an operative system φ , a joint structure \mathfrak{M} , and that $\mathfrak{M} \models \varphi_i^n(\mathbf{x}) \land \neg \varphi_i^{n-1}(\mathbf{x})$. Then n is the stage of \mathbf{x} , denoted $|\mathbf{x}|_{\varphi_i,\varphi}^{\mathfrak{M}}$ (if φ is understood, write $|\mathbf{x}|_{\varphi_i}^{\mathfrak{M}}$; if \mathfrak{M} is understood, write $|\mathbf{x}|_{\varphi_i,\varphi}$; if both φ and \mathfrak{M} are understood, write $|\mathbf{x}|_{\varphi_i}$). If $\neg \varphi_i^{\infty}(\mathbf{x})$, write $|\mathbf{x}|_i = \infty$.

By Formula 2.2, $m < n \implies (\varphi_i^m(\mathbf{x}) \to \varphi_i^n(\mathbf{x}))$ for all \mathbf{x} .

Definition 2.11 Suppose that we have an operative system φ , and let \mathfrak{M} be a joint structure. Then

$$\|\varphi\|^{\mathfrak{M}} = \sup_{\mathbf{x},i} \{ |\mathbf{x}|_{\varphi_i} \colon \mathbf{x} \text{ from } \mathfrak{M} \& i \in \{0, 1, \dots, \nu\} \}$$

is the closure ordinal of φ in \mathfrak{M} .

(Closure ordinals on infinite structures are discussed in [Mo74], [Bar77], and [Mc90a].)

We can get the formulas $\varphi_i(\mathbf{x}, S_0, \ldots, S_{\nu})$ to be monotone by requiring that they be S_j -positive for each j. Given S, the S-positive formulas are constructed by the following recursion:

- 1. If S does not occur in θ , then θ is S-positive.
- 2. If θ and ϕ are S-positive, then so are $\theta \land \phi$, $\theta \lor \phi$, $(\exists y: P(\mathbf{x}; y))\theta$, and $(\forall x: P(\mathbf{x}; y))\phi$, where P is a guard relation.

In essence, φ is S-positive if there are no negations "in front of" any occurrence of S in φ . It is straightforward to prove (see [Mo74]) that if φ_i is S_j -positive for each j, then each φ_i is monotone in each relational argument, so we can find least fixed points of the system $\varphi_0, \ldots, \varphi_{\nu}$.

We will call an operative system φ of formulas, each being S_j -positive for each j, a "positive operative system."

Let's construct a system of formulas for graph reachability:

 $\operatorname{REACH}(x, y) \equiv$ "there is a path along edges from x to y".

In this example, the guard structures are complete digraphs, i.e., for any x, y, there is a P-arc from x to y.

Example 2.6 Imagine that Edge is the edge relation we want to get REACH for, while *P* is the (complete) binary guard relation. One operative system of formulas, with guarded quantification, for generating graph reachability as a least fixed point, is the following:

$$\varphi_0(x, y, S_0, S_1) \equiv x = y \lor S_1(x, y, y)$$

$$\varphi_1(x, y, z, S_0, S_1) \equiv [\operatorname{Edge}(x, z) \land S_0(z, y)] \lor (\exists w: P(z; w)) S_1(x, y, w).$$

Here, REACH = φ_0^{∞} . Notice that $\|\varphi_0, \varphi_1\|^{\mathfrak{M}}$ is quadratic in $\|\mathfrak{M}\|$.

Notice that if the quantification had been unguarded, i.e., if $\varphi_1(x, y, z, S_0, S_1) \equiv [\text{Edge}(x, z) \wedge S_0(z, y)] \vee \exists w S_1(x, y, w)$, the effect would have been the same. The least fixed point logic FO + pos LFP of [Mo74] and [AhU79] uses strictly unguarded quantification, but is otherwise developed the same way as FO_{*}+ LFP..

Notice that in the above example, we did not use any guard constants. However, since a quantification in a sentence requires *something* for the first k arguments of the ((k + 1)ary) guard relation in the outmost quantification, sentences require guard constants.

If P is merely strongly connected — i.e., for any x, y, one can go along P-arcs from x to y — the algorithm would require that in the second line, we would need to search for an appropriate w by traversing an indeterminate number of P-arcs. A single quantification $(\exists w: P(-; w))$ would not be sufficient. We will look at this situation later.

Definition 2.12 Fix a database schema σ and a guard schema ρ .

The logic FO_{*}+ pos LFP on (σ, ρ) defines the set of queries φ_0^{∞} , for φ_0 being from a positive operative system $\varphi = \varphi_0, \ldots \varphi_{\nu}$ of formulas with guarded quantifications, whose guard constants and relations are from ρ and whose database constants and relations are σ .

The logic FO + pos LFP on $\sigma \cup \rho$ defines the set of queries φ_0^{∞} , for φ_0 being from a positive operative system $\varphi = \varphi_0, \ldots, \varphi_{\nu}$ of formulas with unguarded quantifications, whose constants and relations are from $\sigma \cup \rho$.

We have defined the logics this way because we will prove that on finite structures, given a uniformly connected guard system, FO_{*} + pos LFP on (σ, ρ) has the same expressive power as FO + pos LFP on $\sigma \cup \rho$.

The logic FO_{*} + pos LFP consists of the "Least Fixed Points" when \mathcal{L} is the set of formulas in FO_{*} that are S-positive for each second order variable S. Notice that FO_{*} + pos LFP is the least logic containing FO_{*} and closed under disjunction and conjunction, guarded quantifications, and LFP inductions. The logic FO_{*} + LFP is the least logic containing FO_{*} and closed under boolean operations, guarded quantifications, and LFP inductions: unlike FO_{*} + pos LFP, FO_{*} + LFP is explicitly closed under negations. It will turn out in Subsection 4.3 that on finite structures, all FO_{*} + pos LFP expressible queries are FO_{*} + LFP expressible; in Subsection 4.4, we will see that this is *not* true for infinite structures.

2.4 Distance in Guard Relations

There is one pathology that we encountered in Example 2.1 which we will have to deal with: suppose that from the guard constants, one cannot reach all vertices of the database. Fix a guard structure \mathfrak{R} of domain D. Without loss of generality, suppose that there is one guard relation P, besides equality. Let $\mathbf{d} = d_1, \ldots$ be the guard constants, let $D_0 = \{d_1, \ldots\}$, let $D_{m+1} = \{y: \exists x_1, \ldots, (P(x_1, \ldots; y) \land \bigwedge_h x_h \in D_m)\}$ for each integer $m \ge 0$, and let $D_{\infty} = \bigcup_m D_m$.

Definition 2.13 A guard structure of domain D is of radius m if it satisfies $D_{m-1} \neq D = D_m$; it is connected if $D = D_\infty$. A guard system is of radius m if m is the supremum of the radii of its structures, and it is connected if all of its structures are connected.

Notice that on a finite database, a guard structure is connected iff it is of finite radius.

This is a perfectly natural notion. Unfortunately, this is not quite the notion we will need. In this section, we will present the uglier notion that we will need, and show that the natural notion above is insufficient. First, here is the uglier notion. **Definition 2.14** Fix a guard structure \mathfrak{R} . Fix an integer p > 1. Suppose that for each $y \in |\mathfrak{R}|$, if y is not itself a guard constant, then there exists a sequence $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n$ of p-tuples from $|\mathfrak{R}|$ such that the following is true (if $\mathbf{x}_i = x_{i,1}, ..., x_{i,p}$ for each i):

- The tuple \mathbf{x}_0 consists of guard constants.
- For each i ∈ [n] and j ∈ [p − 1], x_{i,j} is either a guard constant or is equal to x_{i−1,j'} for some j'.
- For each i ∈ [n], there is a guard relation P and a tuple x' from x_{i,1},..., x_{i,p-1} such that P(x'; x_{i,p}).
- The sequence of tuples ends with $x_{n,p} = y$.

If this is true, call \mathfrak{R} p-uniformly connected. If \mathcal{R} is a class of p-uniformly connected (guard) structures for some one integer p, call \mathcal{R} uniformly connected.

Definition 2.14 is the notion that we will use in this paper.

Remark 2.4 We could have defined p-uniform connectedness slightly differently, e.g., as follows. Suppose that for each $y \in |\mathfrak{R}|$, if y is not itself a guard constant, then there exists a sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$ of tuples from $|\mathfrak{R}|$, such that the following is true:

- For each i, \mathbf{x}_i is a tuple $x_{i,1}, \ldots, x_{i,q_i}$, where $q_i \leq p$.
- For each i and each j ∈ [q_i], x_{i,j} either occurs in x_{i-1}, or can be accessed from guard constants and elements of x_{i-1} via a guard relation.
- The element y occurs in \mathbf{x}_n .

If these three conditions hold for all $y \in |\Re|$, call \Re p-uniformly^{*} connected. It is not hard to prove that if there are ν guard constants, then any p-uniformly connected guard structure is p-uniformly^{*} connected, and that any p-uniformly^{*} connected structure is $(2p + \nu)$ -uniformly connected. And we would get a notion of "uniform^{*} connectivity" that would coincide with uniform connectivity. The rest of this subsection is devoted to the difference between connected and uniformly connected guard systems. First the good news: these notions coincide when all the guard relations are binary.

Theorem 2.1 Let ρ be a guard schema whose relations are all binary. Then all connected ρ -structures are 2-uniformly connected.

Proof. Fix a connected guard structure \mathfrak{R} , with binary guard relations P_1, \ldots As in Definition 2.13, let D_0 be the set of guard constants, and for each m, let

$$D_{m+1} = \left\{ y \in |\mathfrak{R}| - \bigcup_{i \leq m} D_m \colon \exists x \left(x \in D_m \land \bigvee_i P_i(x; y) \right) \right\}.$$

As \mathfrak{R} is connected, if $y \in |\mathfrak{R}|$, there exists x_0, x_1, \ldots, x_n such that:

- The vertex x_0 is a guard constant.
- For each $i < n, x_i \in D_i, x_{i+1} \in D_{i+1}$, and there exists j_i such that $P_{j_i}(x_i, x_{i+1})$.
- The sequence ends with $x_n = y$.

Thus \Re is 2-uniformly connected.

This means that in most of the examples one would play with, the nice Definition 2.13 would suffice. However, when dealing with messier models, we may want ternary (or worse) guard relations. This brings us to the bad news: if there are guard relations of arity > 2 in a guard schema ρ , then some connected ρ -guard systems are connected but not uniformly connected.

Theorem 2.2 For any p, there exists a guard structure with a 3-ary guard relation and a guard constant that is connected but not p-uniformly connected.

Proof. We will prove this by induction on p. We start with a variant of Definition 2.14

Definition 2.15 Let \mathfrak{R} be a guard structure and let $y \in |\mathfrak{R}|$. A sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$ accesses y if (letting $q_i = \text{length}(\mathbf{x}_i)$ for each i):

- For each i, $\mathbf{x}_i = x_{i,1}, \ldots, x_{i,q_i}$ is a tuple from $|\mathfrak{R}|$...
- For each i and j, if $j \in [q_i 1]$, then $x_{i,j} = x_{i-1,k}$ for some $k \in [q_{i-1}]$. (Here, $[0] = \emptyset$.)
- For each *i*, there exists a tuple \mathbf{x}' of guard constants and elements from $x_{i,1}, \ldots, x_{i,q_i-1}$ such that for some guard relation $P, \mathfrak{R} \models P(\mathbf{x}'; x_{i,q_i}).$
- The sequence ends with $x_{n,q_n} = y$.

In addition, y is accessed within q variables if there is a sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$ accessing y in which $\max_i q_i \leq q$.

If

 $p = \min\{q: "y \text{ is accessed within } q \text{ variables"}\},\$

say that accessing y requires p variables.

Observe that if y can be accessed within q(y) variables for each y, and $p = \max_{y \in \mathfrak{R}} q(y)$, then \mathfrak{R} is p-uniformly connected.

We now construct the structures that will have vertices requiring many variables to access. Fix p. Let $\mathfrak{T}_p = \langle T_p; \operatorname{Arc}_p; x_p \rangle$ be the complete binary tree of root x_p , vertices T_p , arc relation Arc_p , and height p. Let L_p be the leaves in T_p , and each vertex in $T_p - L_p$ has precisely two successors. Following Example 2.5, label the two successors of a vertex $t \in T_p - L_p$ by it and $\mathfrak{r}t$: consider these the *left* and *right* successors of t (but notice that it and $\mathfrak{r}t$ are *not* distinguishable in the language of \mathfrak{T}_p). Extend \mathfrak{T}_p by adding a vertex $d \notin T_p$, and let

$$P_p = \{(\mathfrak{l}t, \mathfrak{r}t, t) \colon t \in T_p - L_p\} \cup \{(d, d, t) \colon t \in L_p\},\$$

and let $\mathfrak{R}_p = \langle T_p \cup \{d\}; P_p; d \rangle$. Note that $x_p \in |\mathfrak{R}_p|$, but x_p is not the guard constant d. Figure 1 displays \mathfrak{R}_1 .



To prove the Theorem, it suffices to prove the following Claim by induction on p.

Claim 2.1 In \mathfrak{R}_p , accessing x_p requires p + 2 variables.

The basis of the induction is: p = 1. Here, if $x = x_1$,

$$\mathfrak{R}_1 = \langle \{x, \mathfrak{l}x, \mathfrak{r}x, d\}; \{(d, d, \mathfrak{l}x), (d, d, \mathfrak{r}x), (\mathfrak{l}x, \mathfrak{r}x, x)\}; d \rangle.$$

To access x, one uses the tuple $(\mathfrak{l}x, \mathfrak{r}x, x)$, so accessing x requires at least 3 variables. But 3 variables are sufficient: to access x via $(d, d, \mathfrak{l}x)$, $(d, d, \mathfrak{r}x)$, $(\mathfrak{l}x, \mathfrak{r}x, x)$, the sequence $(\mathfrak{l}x)$, $(\mathfrak{l}x, \mathfrak{r}x)$, $(\mathfrak{l}x, \mathfrak{r}x, x)$ satisfies Definition 2.15.

We proceed by induction on p. Suppose that accessing x_p in \mathfrak{R}_p requires p+2 variables. We claim that accessing x_{p+1} in \mathfrak{R}_{p+1} requires p+3 variables.

First, we claim that p+3 variables are necessary to access x_{p+1} in \mathfrak{R}_{p+1} . Suppose that $\mathbf{x}_0, \ldots, \mathbf{x}_n$ witnesses the accessing of x_{p+1} as in Definition 2.15. Then $x = x_{p+1}$ is the last entry of \mathbf{x}_n , while $\mathfrak{l}x$ and $\mathfrak{r}x$ are other entries in \mathbf{x}_n (for otherwise, we cannot access x). Thus $\mathfrak{l}x$ and $\mathfrak{r}x$ occur in \mathbf{x}_{n-1} .

Now, observe that \mathfrak{R}_{p+1} consists of two copies of \mathfrak{R}_p , disjoint except that they share d, with $\mathfrak{l}x$ and $\mathfrak{r}x$ as the roots of the two copies of \mathfrak{R}_p , and with the additional vertex

 $x = x_{p+1}$ on top (look at Figure 2 below). Denote the (left) copy containing $\mathfrak{l}x$ by \mathfrak{M}_p , and the (right) copy containing $\mathfrak{r}x$ by $\mathfrak{r}\mathfrak{M}_p$. Thus each tuple \mathbf{x}_m , m < n, consists of elements from $|\mathfrak{l}\mathfrak{M}_p|$ and elements from $|\mathfrak{r}\mathfrak{M}_p|$. Let $\mathbf{x}_m^{\mathfrak{l}}$ be the tuple from \mathbf{x}_m of elements from $|\mathfrak{l}\mathfrak{M}_p|$, and let $\mathbf{x}_m^{\mathfrak{r}}$ be the tuple from \mathbf{x}_m of elements from $|\mathfrak{r}\mathfrak{R}_p|$.



Figure 2

Note that without loss of generality, we can assume that if $\mathbf{x}_m^{\mathfrak{l}}$ has entries, and m' > m, then $\mathbf{x}_{m'}^{\mathfrak{l}}$ also has entries. Otherwise, we could delete all entries of $|\mathfrak{lR}_p|$ from each $\mathbf{x}_{m''}^{\mathfrak{l}}$, m'' < m', and we would still have a sequence of tuples accessing x. Thus there is an $m_{\mathfrak{l}}$ such that $\mathbf{x}_m^{\mathfrak{l}}$ has entries iff $m \ge m_{\mathfrak{l}}$; similarly, there is an $m_{\mathfrak{r}}$ such that $\mathbf{x}_m^{\mathfrak{r}}$ has entries iff $m \ge m_{\mathfrak{r}}$.

By the Induction Hypothesis, as lx is accessed by the sequence accessing x, there exists $m_l \ge m_l$ such that the tuple $\mathbf{x}_{m_l}^{\mathfrak{l}}$ has at least p+2 entries. Similarly, there exists $m_r \ge m_{\mathfrak{r}}$ such that the tuple $\mathbf{x}_{m_r}^{\mathfrak{r}}$ has at least p+2 entries. Without loss of generality, suppose that $m_l \le m_r$: then $m_l \le m_r$, so \mathbf{x}_{m_r} has at least one from $l\mathfrak{R}_p$. Then \mathbf{x}_{m_r} must have at

least (p + 2) + 1 = p + 3 entries.

Thus accessing x in \mathfrak{R}_{p+1} requires at least p+3 variables.

We conclude by claiming that p + 3 variables suffice.

Let $\mathbf{x}_0, \ldots, \mathbf{x}_{m_t}$ be a sequence of tuples of $|\mathfrak{lR}_p|$ accessing \mathfrak{lx} (with p+2 variables), and let $\mathbf{y}_0, \ldots, \mathbf{y}_{m_t}$ be a sequence of tuples of $|\mathfrak{rR}_p|$ accessing \mathfrak{rx} (again, with p+2 variables). Letting "^" mean concatenation of tuples, the sequence

$$\mathbf{x}_0,\ldots,\mathbf{x}_{m_1},(\mathfrak{l}x)\,\hat{\mathbf{y}}_0,\ldots,(\mathfrak{l}x)\,\hat{\mathbf{y}}_{m_{\mathfrak{r}}},(\mathfrak{l}x,\mathfrak{r}x,x)$$

accesses x within p + 3 variables.

In Subsection 3.4, we will see that uniform connectivity is what we want. We conclude with a definition we will need later.

Definition 2.16 Fix an integer p > 0 and a guard relation \mathfrak{R} . Given $y \in |\mathfrak{R}|$, the puniform norm of y is the least n such that there is a sequence of n p-tuples satisfying Definition 2.14 for accessing y. The p-uniform radius is the maximum p-uniform norm of any element of \mathfrak{R} .

3 Game Programs

In order to simplify proofs, we will use a variant of the pebble game calculus of [Mc95a], which is a sort of generalized Datalog. In this section, we will describe the games and then look at the connection between the games on the one hand and FO_{*} and FO_{*} + pos LFP on the other.

In this section, we will be using Peircean (as in Charles Sanders Peirce) games of the sort described in [HiS97]: given a structure \mathfrak{M} and a property \mathcal{P} , one player is asserting that \mathfrak{M} satisfies property \mathcal{P} and the other is denying it. In this Section, we will look these Peircean games and their connections to guarded least fixed point logic.

3.1 Playing the Game

The game is played between two players — call them Eloise and Abelard — on a board (a joint structure \mathfrak{M}), where the pieces are an assortment of pebbles. The idea is that

Eloise is trying to prove that \mathfrak{M} has a certain property (e.g., it is a connected graph, or a linear partial order, or some such), in the face of Abelard's challenges.

At any moment, the game is in a particular state s (representing some kind of assertion), with pebbles on a tuple of vertices \mathbf{x} . Eloise claims that the notion associated with the state s is true of \mathfrak{M} at the tuple \mathbf{x} , while Abelard claims that it is false. For each state s, there is a *rule* which determines how many pebbles are on the board (possibly zero) and what is to be done: if someone is to move, the rule determines whose turn it is, how that player may move, or if someone is to win, the rule determines the criteria for victory. The collection of these finitely many rules is the game program.

Game programming is a variant of Datalog programming (see a theoretical text like [AbiHV95] or [U88, 89] for a description of Datalog). A game program is a collection of rules

$$(s;\mathbf{x}):-\cdots$$

where \mathbf{x} is a tuple of variables (for positions of the pebbles) and '...' describes what is to be done if the game is in state s. The rules are 'junctive', 'quantitative', or 'terminal'. As in Datalog, we will say that the term on the left hand side of ':- ' is the *head* of the rule while the expression on the right hand side of ':- ' is the *body*. And we will use a bookkeeping convention from Datalog: when the game moves from state to state, within a rule, the values follow the variables; but when the game goes to the next rule, the values follow the order of the arguments, not the variable names.

Let's work within a fixed joint schema (σ, ρ) .

Initial state. We will presume that the game begins in a special state START, with zero variables. Thus the question will be whether Eloise has a winning strategy from the game position (START;).

Junctive rules. The state s might be 'disjunctive' or 'conjunctive'. First, we look at disjunctive rules. Let \mathbf{x}_1 and \mathbf{x}_2 list variables from $\mathbf{x} = x_1, \ldots, x_e$. Then

$$(s;\mathbf{x}):=(s_1;\mathbf{x}_1)\vee(s_2;\mathbf{x}_2)$$

means that if the game is in state s, and the pebbles p_1, \ldots, p_e are on (the vertices represented by the variables) x_1, \ldots, x_e resp., then Eloise decides whether to go to state

 s_1 , with pebbles $p_1, \ldots, p_{e'}$ on $x_{1,1}, \ldots, x_{1,e'}$ (where $\mathbf{x}_1 = x_{1,1}, \ldots, x_{1,e'}$ consists of variables from \mathbf{x}), or to state s_2 , with pebbles $p_1, \ldots, p_{e''}$ on $x_{2,1}, \ldots, x_{2,e''}$ (where $\mathbf{x}_2 = x_{2,1}, \ldots, x_{2,e''}$ consists of variables from \mathbf{x}). Here, for Eloise to claim that s is TRUE at \mathbf{x} , she has to claim that either s_1 is TRUE at \mathbf{x}_1 or that s_2 is TRUE at \mathbf{x}_2 . Since she only has to defend one or the other, she is permitted to choose which junct to defend: from state $(s; \mathbf{x})$, Eloise chooses whether to continue the game from $(s_1; \mathbf{x}_1)$ or from $(s_2; \mathbf{x}_2)$.

Notice that there may be some rearranging of pebbles. For example, for the rule $(s; x, y) := (s_1; x) \lor (s_2; y, x)$, if the game is in state s with pebble p_1 on the first argument and p_2 on the second, then if Eloise chooses to go to state s_2 , the pebbles must be switched (if Eloise chose to go to state s_1 , pebble p_1 would stay put and pebble p_2 would be removed). This is actually analogous what happens in a computer: the variables are like the variable-names of a higher language, while the pebbles are like the registers of the machine. From now on, we will rearrange pebbles without comment.

Similarly, the conjunctive rule

$$(s;\mathbf{x}):=(s_1;\mathbf{x}_1)\wedge(s_2;\mathbf{x}_2)$$

means that Abelard decides whether to go into state s_1 or state s_2 . Here Eloise claims that *both* juncts are TRUE, so as she would be expected to be able to defend either, Abelard can choose which junct to challenge.

Quantitative rules. The state s might be 'existential' or 'universal'. Let \mathbf{x}_1 and \mathbf{x}_2 consist of variables chosen from \mathbf{x} , only \mathbf{x}_1 may also list guard constants. Let P be a guard relation (of ρ). Then an *existential* rule is of the form:

$$(s;\mathbf{x}) := (\exists y: P(\mathbf{x}_1; y))(s'; \mathbf{x}_2, y),$$

which means that if the game is in state s with pebbles on the vertices \mathbf{x} , then Eloise is to choose a vertex y such that $P(\mathbf{x}; y)$. And \mathbf{x}_2 consists of vertices from \mathbf{x} . Then rearranging pebbles so now the tuple \mathbf{x}_2, y is pebbled, the game continues from state s', i.e., from position $(s'; \mathbf{x}_2, y)$.

Now suppose that Eloise claimed that for every y accessible from \mathbf{x}_1 , s_2 would be TRUE at \mathbf{x}_2, y : we would now permit Abelard to choose which y he cared to challenge.

The result is the *universal* rule

$$(s;\mathbf{x}):=(\forall y:P(\mathbf{x}_1;y))(s';\mathbf{x}_2,y).$$

We call these states (and their rules) existential and universal respectively.

Notice that a player may be called on to make a quantitative move, and yet there may be no legal moves: Player Q is to choose y such that $P(\mathbf{x}'; y)$, and yet, $\forall y \neg P(\mathbf{x}'; y)$. What then?

- For (s; x) :- (∃y: P(x₁; y))(s'; x₂, y), Eloise claims that a certain y exists, so if she cannot move from (s; x), no such y exists, so she should lose.
- For (s; x) :- (∀y: P(x₁; y))(s'; x₂, y), Abelard claims that all y with P(x₁; y) lead to winning positions, which is vacuously true if ∀y¬P(x₁; y), so if no such y exists, he should win.

This motivates the following asymmetric convention.

Convention 3.1 For the rule $(s; \mathbf{u}) := (Qv: P(\mathbf{u}_1; v))(s'; \mathbf{u}_2, v)$, from $(s; \mathbf{x})$, if $\forall y \neg P(\mathbf{x}; y)$, then Abelard wins.

Terminal rules. Finally, the state might be 'terminal': this is when Eloise is asserting that an atomic formula (or its negation) is TRUE at a tuple \mathbf{x}' , and all that remains is to check.

Let \mathbf{x}' consist of variables from \mathbf{x} . Here the rule could be of the form

$$(s;\mathbf{x}):=R(\mathbf{x}'),$$

where R is a database relation from σ and \mathbf{x}' has the appropriate number of arguments: this means that if the state is s, and \mathbf{x} is a given tuple of vertices, then Eloise wins iff $R(\mathbf{x}')$; otherwise, Abelard wins. Or the rule could be of the form

$$(s; \mathbf{x}) := \neg R(\mathbf{x}'),$$

which means that Eloise wins iff $\neg R(\mathbf{x}')$, with Abelard winning otherwise. Notice that these terminal rules are the only rules in which the database relations appear explicitly, as opposed to the guard relations, which were explicitly available in the quantitative rules. A terminal rule could also be of the form

$$(s;\mathbf{x}):-x'=c,$$

where x' is a variable from **x** and c is a database constant, or of the form

$$(s;\mathbf{x}):-x'\neq c,$$

where x' is from **x** and c is a database constant. In the first case, Eloise wins iff the variable x' has the same value as the constant c, and in the second case, Eloise wins iff the variable x' has a different value than the variable c. Again, notice that unlike the guard constants, which were explicit in the quantification rules, the database constants only appear at the end of the game.

Definition 3.1 The programs as defined above are the game programs.

Incidentally, a game position $(s; \mathbf{x})$ will be called *junctive* if s is a junctive state, *quantitative* if s is a quantitative state, and *terminal* if s is a terminal state.

Thus the players play the game until one or the other wins. But notice that we have not built in any guarantee that a terminal position will be reached: indeed, we will find that some games go on forever. We will want a convention motivated by a notion similar to the "negation as failure" of [Cl78].

Convention 3.2 If a game goes on forever, then Abelard wins.

The rationale for this is that Eloise has the burden of establishing that the property holds on the given structure: if she never does this, she loses.

Note that Conventions 3.1 and asymmetry, especially the latter, destroy the symmetry between Eloise and Abelard. This will have technically unpleasant consequences later on.

3.2 Winning the Game

Now that we have a notion of playing the game by some rules, let's take a brief look at how the game is won or lost.

Recall that Eloise should be able to win the game iff the structure in question satisfies the given property (represented by the game program). So first, let's get a little shorthand. **Definition 3.2** Given a game program Φ and a structure \mathfrak{M} , let $\mathcal{G}(\Phi, \mathfrak{M})$ be the game played on \mathfrak{M} using the game program Φ .

What does it mean to say that Eloise "wins" or is "able to win"? The standard notion is to use "strategies". A *strategy* is a function

game positions \rightarrow moves

telling a player how to move. (We will not go into the nuts and bolts of strategies in this paper: for that sort of thing, see [Ko85].) If Eloise has a strategy that can defeat Abelard no matter how Abelard plays — i.e., a strategy that can defeat any of Abelard's strategies — we say that Eloise has a *winning strategy*.

Definition 3.3 The game played on the joint structure $(\mathfrak{D}, \mathfrak{R})$ using the program Φ will be denoted $\mathcal{G}(\Phi, (\mathfrak{D}, \mathfrak{R}))$. We say that a player wins $\mathcal{G}(\Phi, (\mathfrak{D}, \mathfrak{R}))$ if that player has a winning strategy for that game, i.e., a strategy that will defeat any strategy employed by her or his opponent.

Now for a little hand-waving. It is a consequence of the Gale-Stewart Theorem ([GaS53] — see, e.g., [Mo80]) that either Eloise or Abelard has a winning strategy.

We will want an important measure: from a given position $(s; \mathbf{x})$, if both players play optimally, how long can the game last? Since Abelard wins if the game goes on forever, we ask instead: how many moves before Eloise wins (if Abelard has a winning strategy, the answer is ∞). In order to develop this measure, we need a fact.

Proposition 3.1 Let $(\mathfrak{D}, \mathfrak{R})$ be a joint structure and let Φ be a game program. Suppose that in $\mathcal{G}(\Phi, (\mathfrak{D}, \mathfrak{R}))$, every universal quantitative position admits only finitely many options that Abelard can choose from. Suppose that for each n, there exists a strategy Z_n for Abelard such that if Abelard uses Z_n , then (starting from the initial position (START;)), Eloise has no strategy that defeats Z_n within n moves. Then Abelard wins $\mathcal{G}(\Phi, (\mathfrak{D}, \mathfrak{R}))$.

Proof. We describe a winning strategy for Abelard. We maintain a set of *active* strategies which Abelard can use at a given position: a strategy will be active if all the moves made

thus far made by Abelard were made consistent with the strategy. Notice that if Z_n is active after m moves, where m < n, then Abelard can play (namely as dictated by Z_n) so that Eloise can't win within n - m moves.

We will start with a set of strategies $ST_0 = \{Z_1, Z_2, Z_3, Z_4, \ldots\}$. If ST_0 is finite, then there exists $Z \in ST_0$ such $Z = Z_n$ for arbitrarily large n, and thus Z can never be defeated by Eloise, and thus is a winning strategy for Abelard. So suppose that ST_0 is infinite. During the game, at the kth move, ST_k will be a set of strategies for Abelard, all of them consistent with the play (i.e., with Abelard's play) thus far.

Start at (START;) with the set of active strategies is $ST_0 = \{Z_1, \ldots\}$. Let ST_k be the set of active strategies at the position for the (k + 1)st move, whether it is Eloise's or Abelard's turn to move. Suppose that ST_k is infinite:

- If it is Eloise's turn to move, then as no move by Eloise can be inconsistent with an active strategy for Abelard, $ST_k = ST_{k+1}$, and ST_{k+1} is infinite.
- We will prove that for any k, if it is Abelard's turn to move, then as ST_k is infinite, Abelard can move so that ST_{k+1} is infinite.

It will follow that for each k, and each N, there will exist n > N such that $Z_n \in ST_k$.

Suppose that the game is at the position $(s; \mathbf{x})$, in which it is Abelard's turn to move, and he has already had k moves. Suppose that ST_k is infinite.

If s is conjunctive, then Abelard is to move according to

$$(s;\mathbf{x}):=(s_1;\mathbf{x}_1)\wedge(s_2;\mathbf{x}_2).$$

So for some $i \in [2]$, there exist infinitely many $Z_n \in ST_k$ such that from $(s; \mathbf{x})$, Z_n chooses to go to junct s_i : Abelard chooses that s_i . Let ST_{k+1} be the set of all $Z_n \in ST_k$ dictating that Abelard move to the *i*th junct, and note that ST_{k+1} is infinite.

If s is universal, then Abelard is to move according to

$$(s; \mathbf{x}) := (\forall y: P(\mathbf{x}'; y))(s'; \mathbf{x}'', y).$$

There are only finitely many y such that $P(\mathbf{x}'; y)$, so for at least one of these y, there are infinitely many $Z_n \in ST_k$ that has Z_n choose that y from $(s; \mathbf{x})$: Abelard chooses that y. Let ST_{k+1} be the set of all $Z_n \in ST_k$ dictating that Abelard choose y, and note that ST_{k+1} is infinite.

If Abelard plays so that ST_k is always infinite, then Eloise can never make ST_k finite, so she never can win within a fixed number of moves, hence certainly never within 0 moves, so she never wins, so Abelard wins by default.

We needed the assumption that the guard relations permit quantification over only finitely many elements. Note that if we had a guard structure like the following, then while Abelard has a strategy Z_n preventing Eloise's victory for n moves, Eloise will eventually win the game.

Example 3.1 Let $\mathfrak{R} = \langle M; R; d \rangle$, where M and R are to be constructed as follows (see Figure 3):

- For each positive integer z, let $M_z = \{a_{z,1}, \ldots, a_{z,z}\}.$
- Let $M = \bigcup_{z=1}^{\infty} M_z$.
- Let $R(a,b) \equiv (a = d \land \bigvee_{z=1}^{\infty} b = a_{z,1}) \lor (\bigvee_{z,i: i \in [z-1]} (a = a_{z,i} \land b = a_{z,i+1})).$

Let Φ consist of Abelard starting from d, and going down R-arcs until he cannot move any more: once he cannot move, he loses. For any n, Abelard can choose to go down a path of length at least n, but no matter which path he chooses, he will eventually reach the end of it and lose.



Figure 3

We return to the problem of how long a game can last before Eloise wins.

Definition 3.4 Fix a joint structure \mathfrak{M} . Suppose that the game $\mathcal{G}(\Phi, \mathfrak{M})$ is being played, and that s is a state from Φ and that \mathbf{x} is a tuple from $|\mathfrak{M}|$. If Eloise has a strategy to win from the position $(s; \mathbf{x})$ within n moves, but no faster, write $|\mathbf{x}|_s = n$. If Eloise does not have a strategy to win from the position $(s; \mathbf{x})$, write $|\mathbf{x}|_s = \infty$.

Notice the similarity to Definition 2.10; in Subsection 4.2, we will see that this similarity is not coincidental.

Definition 3.5 A game program Φ captures a property φ iff for every joint structure \mathfrak{M} , $\mathfrak{M} \models \varphi$ iff Eloise wins the game $\mathcal{G}(\Phi, \mathfrak{M})$. Thus if \mathcal{L} is a logic and \mathcal{P} is a collection of game programs, we say that all \mathcal{L} -expressible queries are \mathcal{P} -expressible if every \mathcal{L} -expressible property is captured by some game program in \mathcal{P} ; we define the converse similarly.

3.3 The Topology of Games

There are two kinds of games: those that can go on indefinitely, and those that end within a fixed number of moves. In this subsection, we find that the difference lies in the "topology" of the game programs – or more precisely, in the topology of the digraphs of the game programs.

Let's start with an example of a game that ends in three moves.

Example 3.2 Consider a database $\mathfrak{M} = \langle M; \operatorname{Arc} \rangle$, where Arc is binary, and a guard structure $\mathfrak{R} = \langle M; P; d \rangle$, where P is binary. Let the database relation be Arc, the database constant be c, the guard relation be P, and the guard constant be d. Consider the sentence

$$(\forall x: P(d; x))(\forall y: P(x; y))(\exists z: P(d; z))(y = z).$$

This sentence says that if there is a P-path from d to x, then P(d, x). Thus if \mathfrak{R} was a connected guard structure, this sentence would say that for every vertex (except perhaps d itself), there is a P-arc from d to x. Notice that this sentence says nothing about the database relation Arc.

Compare the above sentence with the following game program. (We will typically have descriptive names for the states.)

$$(\text{START};) := (\forall x: P(d; x))(\text{ADJACENT}; x)$$

$$(\text{ADJACENT}; x) := (\forall y: P(x; y))(\text{NEXT}; y)$$

$$(\text{NEXT}; y) := (\exists z: P(d; z))(\text{EQUALS}; y, z)$$

$$(\text{EQUALS}; y, z) := y = z$$

Here, Eloise wants to prove d is adjacent to every other vertex (assuming that the guard structure is connected). Then Eloise can win the above game (no matter how Abelard plays) iff the sentence in Example 3.2 is true. We say that the above game *captures* the sentence.

Notice that in the above game, one can never revisit a state. Compare this to the following program. Here we ask a dual question of the database Arc that we asked of the guard relation P: is it true that for *some* vertex x, there exists an Arc from x to d?

Example 3.3 The guard structure is connected, with one guard constant d and one guard relation P. Now we want to search the entire database, even if the guard structure is of

large (if finite) radius.

$$(START;) := (\forall x: x = d)(ASK; x)$$

$$(ASK; x) := (REACH; x) \lor (CONTINUE; x)$$

$$(REACH; x) := (\exists y: d = y)(ARC; x, y)$$

$$(CONTINUE; x) := (\exists y: P(x; y))(ASK; y)$$

$$(ARC; x, y) := Arc(x, y)$$

In this game, the states ASK and CONTINUE can be visited any number of times: we will call these states recursive.

So here we have two kinds of free games: those that have recursive states and those that don't. Let's formalize this notion.

Definition 3.6 Let Φ be a game program. Let " $s \vdash t$ " mean that in Φ , the state t appears in the body of s's rule. Let \vdash^* be the transitive closure of \vdash , and let " $s \vdash^+ t$ " mean that in Φ , for some $u, s \vdash u$ and $u \vdash^* t$.

Call a state s recursive if $s \vdash^+ s$. Call a game recursive if it has recursive states.

If a game's program has no recursive states, the game will not last very long. On the other hand, if there are recursive states, then it is possible for the game to go on forever: recall that if the game never ends, then Abelard wins.

Example 3.4 Here is a game program for graph reachability. The database is a graph $\mathfrak{G} = \langle V; \operatorname{Edge}; a, b \rangle$, where V is the set of vertices and Edge is the edge relation, and $a, b \in V$. The query is: there is a path along edges from a to b. The (2-connected) guard relation is a digraph $\langle V; P; d \rangle$, where P is a binary relation on V and $d \in V$. The naive (unguarded) vertex-by-vertex algorithm can be represented by the following program from [Mc95a].

The suggested program for reachability is the following. First, here is the program for *unguarded games*, as in [Mc95a]: all Eloise has to do is start at a and proceed until she

reaches b:

$$\begin{array}{rcl} (\mathrm{START};) & :- & (\mathrm{REACH}; a, b) \\ (\mathrm{REACH}; x, y) & :- & (\mathrm{EQ}; x, y) \lor (\mathrm{STEP}; x, y) \\ (\mathrm{STEP}; x, y) & :- & \exists z (\mathrm{CHECK}; x, y, z) \\ (\mathrm{CHECK}; x, y, z) & :- & (\mathrm{EDGE}; x, z) \land (\mathrm{REACH}; z, y) \\ (\mathrm{EQ}; x, y) & :- & x = y \\ (\mathrm{EDGE}; x, y) & :- & \mathrm{Edge}(x, y). \end{array}$$

See Figure 4 below for a picture (it may help to represent these games as flowcharts) (notice that for pictoral reasons, variables are handled differently in flowcharts, including "reset" to help human observers keep track of variable values).



To convert this "unguarded" game program into a "guarded" game program, we need

to have Eloise navigate through the guard structure. One way to do this is:

$$\begin{array}{rcl} (\mathrm{START};) &:- & (\exists x: x = d)(\mathrm{TO}\text{-}a; x) \\ (\mathrm{TO}\text{-}a; x) &:- & (\mathrm{IS}\text{-}a; x) \lor (\mathrm{TOWARDS}\text{-}a; x) \\ (\mathrm{TOWARDS}\text{-}a; x) &:- & (\exists w: P(x,w))(\mathrm{TO}\text{-}a; w) \\ (\mathrm{IS}\text{-}a; x) &:- & (\mathrm{EQ}\text{-}a; x) \land (\mathrm{AND}\text{-}b; x) \\ (\mathrm{EQ}\text{-}a; x) &:- & x = a \\ (\mathrm{AND}\text{-}b; x) &:- & (\exists y: y = d)(\mathrm{TO}\text{-}b; x, y) \\ (\mathrm{TO}\text{-}b; x, y) &:- & (\mathrm{IS}\text{-}b; x, y) \lor (\mathrm{TOWARDS}\text{-}b; x, y) \\ (\mathrm{TO}\text{-}b; x, y) &:- & (\exists w: P(x, w))(\mathrm{TO}\text{-}b; x, w) \\ (\mathrm{IS}\text{-}b; x, y) &:- & (\mathrm{EQ}\text{-}b; y) \lor (\mathrm{REACH}; x, y) \\ (\mathrm{EQ}\text{-}b; x) &:- & x = b \\ (\mathrm{REACH}; x, y) &:- & (\mathrm{EQ}\text{-}b; x, y) \lor (\mathrm{STEP}\text{;} x, y) \\ (\mathrm{STEP}\text{;} x, y) &:- & (\exists z: z = d)(\mathrm{NEXT}\text{;} x, y, z) \\ (\mathrm{NEXT}; x, y, z) &:- & (\exists w: P(z, w))(\mathrm{NEXT}; x, y, w) \\ (\mathrm{SEARCH}; x, y, z) &:- & (\mathrm{EDGE}\text{;} x, z) \land (\mathrm{REACH}\text{;} z, y) \\ (\mathrm{EQ}\text{-}x, y) &:- & x = y \\ (\mathrm{EDGE}\text{;} x, y) &:- & \mathrm{Edge}(x, y). \end{array}$$

See Figure 5 below for a picture of this program. Notice that this program has three subdigraphs of recursive states: unlike the unguarded program, a single existential quantification may involve as much work as a recursion of existential quantifications. Notice that in a sense, the guarded program is a refinement of the unguarded one: we will formalize a notion of refinement in Subsubsection 4.3.



Figure 5

3.4 Uniform Connectivity and Games

We are now ready to explain, in a precise sense, why Definition 2.14 is the right definition for accessing an entire database.

To search an entire database, we need something like Example 3.3. Here's the idea. Starting at the guard constants \mathbf{d} , repeatedly accessing additional vertices, one should be

able to reach any vertex. We want to formalize this notion within a game context. In order to do this, we need a notion from combinatorial game theory.

Definition 3.7 Let Φ be a program over a joint schema (σ, ρ) , and let \mathfrak{M} be a joint (σ, ρ) -structure. A run of $\mathcal{G}(\Phi, \mathfrak{M})$ is a sequence of game positions

$$(START;) = (s_0;), (s_1; \mathbf{x}_1), (s_2; \mathbf{x}_2), (s_3; \mathbf{x}_3), \dots,$$

such that:

- for each n, s_n is a state whose number of variables equals the length of the tuple x_n of elements from M, and
- for each $n, s_n \vdash s_{n+1}$, and from $(s_n; \mathbf{x}_n)$, a legal application of the rule for s_n produces $(s_{n+1}; \mathbf{x}_{n+1})$, and
- if the sequence terminates, it terminates in a position $(s; \mathbf{x})$, where s is terminal.

We need a notion from [I81] (see [Mc95b]):

Definition 3.8 Let Φ be a game program. The Number of Variables of Φ is the maximum number of variables of any state of Φ .

Notice that in a run of Φ on any structure, the tuples are all of a length no greater than the number of variables of Φ .

First, we observe the obvious.

Proposition 3.2 Fix a positive integer p and a joint schema (σ, ρ) . Let \mathfrak{M} be a joint (σ, ρ) -structure, and let Φ be a game program of no more than p variables and of schema (σ, ρ) . Suppose that for every $x \in |\mathfrak{M}|$, there is a run of $\mathcal{G}(\Phi, \mathfrak{M})$ such that x occurs in the run. Then \mathfrak{M} is p-uniformly connected.

Thus, if \mathcal{M} is a set of joint structures of a common schema, and if Φ is a game program in that schema such that for every $\mathfrak{M} \in \mathcal{M}$ and every $x \in |\mathfrak{M}|$, x occurs in some run of $\mathcal{G}(\Phi, \mathfrak{M})$, then \mathcal{M} is uniformly connected. And the converse is also true.

Theorem 3.1 Let \mathcal{M} be a class of structures of a common joint schema. Suppose that \mathcal{M} is uniformly connected. Then there exists a game program Φ of that schema such that for every $\mathfrak{M} \in \mathcal{M}$ and every $x \in \mathfrak{M}$, there exists a run of $\mathcal{G}(\Phi, \mathfrak{M})$ such that x occurs in that run.

Proof. Fix a positive integer p, and suppose that \mathcal{M} is p-uniformly connected. We will describe the construction of a complicated variant of the program in Example 3.3. Suppose, for simplicity, that there is but one ((k + 1)-ary, $k \leq p)$, guard relation P and one guard constant d.

The idea is as follows. Then we will construct a game program Φ such that on any $\mathfrak{M} \in \mathcal{M}$ and any $x \in \mathfrak{M}$, if $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$ is a sequence of tuples (each of at most p components) witnessing the accessing of x (so that the sequence satisfies the criteria of Definition 2.14, and x occurs in \mathbf{x}_n), then this is a subsequence of a sequence of tuples in a run of Φ .

Here we go. First, Eloise needs to build up a (p + k - 1)-tuple of copies of d.

$$(\text{START};) :- (\forall u: u = d)(\text{Start}; u)$$

(Start; u) :- (NEXT; u, ..., u) \((NEXT; u, ..., u).

The second line is to get in the tuple of p + k - 1 ds.

Then comes the heart of the program:

$$(\text{NEXT}; \mathbf{u}, \mathbf{v}) := (\exists y : P(\mathbf{u}; y))(\text{ASK}_1; \mathbf{v}, y),$$

where **u** is a k-tuple and **v** is a (p-1)-tuple. Note that ASK₁ has p arguments.

The idea is this: in the *i*th time that the game goes into state ASK₁, the game position is (ASK₁; \mathbf{x}_i). For each *i*, let \mathbf{x}''_i be the tuple of the first *k* components of \mathbf{x}_i used to access $x_{i,p}$, and let \mathbf{x}'_i be the first (p-1) components of \mathbf{x}_i , and let $y_i = x_{i,p}$.

Now comes the tedious part: Eloise wants to select \mathbf{x}_{i+1}'' from \mathbf{x}_i to get a k-tuple of elements in the right order to access $x_{i+1,p}$. She also wants to select \mathbf{x}_{i+1}' from \mathbf{x}_i . to keep the run in lockstep with the connecting sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots$. Thus for each of the k

arguments of \mathbf{x}_{i+1}'' , she chooses one of the (up to) p + 1 elements of \mathbf{x}_i or (don't forget!) *d*. Then for each of the p - 1 arguments of \mathbf{x}_{i+1}' , she chooses one of the (up to) p + 1elements of \mathbf{x}_i or *d*. Together, this can be done by choosing one of the $(p+1)^{k+p-1}$ maps from [k+p-1] (giving the argument positions of \mathbf{x}_{i+1}') to [p+1] (giving the elements of \mathbf{x}_i, d). We can set up rules for this as follows.

Let $\tau_1, \tau_2, ..., \tau_{(p+1)^{k+p-1}}$ be the set of all maps $[k + p - 1] \rightarrow [p+1]$. For each $i < (p+1)^{k+p-1} - 2$, let

$$(ASK_i; \mathbf{w}, y) := (ASK_{i+1}; \mathbf{w}, y) \lor (NEXT; z_{\tau_i(1)}, \dots, z_{\tau_i(k+p-1)}),$$

where, for each $j \in [k + p - 1]$,

$$z_j = \begin{cases} w_j & \text{if } \tau(j) \in [p], \\ d & \text{if } \tau(j) = p+1, \end{cases}$$

and let

$$(\text{ASK}_{(p+1)^{k+p-1}-1}; \mathbf{w}, y) := (\text{NEXT}; z_{\tau_{(p+1)^{k+p-1}-1}(1)}, \dots, z_{\tau_{(p+1)^{k+p-1}-1}(k+p-1)}) \\ \vee (\text{NEXT}; z_{\tau_{(p+1)^{k+p-1}(1)}}, \dots, z_{\tau_{(p+1)^{k+p-1}(k+p-1)}}).$$

The resulting program is Φ . Notice that as there are no terminal states (!), for any \mathfrak{M} , no run of $\mathcal{G}(\Phi, \mathfrak{M})$ terminates.

We now verify that for any $\mathfrak{M} \in \mathcal{M}$ and any $x \in |\mathfrak{M}|$, there is a run of $\mathcal{G}(\Phi, \mathfrak{M})$ in which x occurs. As \mathcal{M} is *p*-uniformly connected, for any $\mathfrak{M} \in \mathcal{M}$, and any $x \in |\mathfrak{M}|$, x occurs as the last element in some *p*-tuple \mathbf{x}_n of a sequence $\mathbf{x}_0, \ldots, \mathbf{x}_n$ satisfying Definition 2.14.

Here is the run that reaches x. For each i, let $\mathbf{w}_i = v'_{i,1}, \ldots, v'_{i,2p}$.

- Start at (START;) at the empty tuple. If x₁' is the (p − 1)-tuple of copies of d, and x''₁ is the k-tuple of copies of d, then Eloise quickly reaches (NEXT; x''₁, x'₁), and then (ASK₁; x₁), where x₁ is the concatenation of x'₁ and y₁.
- After the *i*th move, i < n, at $(ASK_1; \mathbf{x}_i)$, Eloise goes through the disjunctions to choose one that selects \mathbf{x}_{i+1}'' and \mathbf{x}_{i+1}' from \mathbf{x}_i , and the next game position is $(NEXT; \mathbf{x}_{i+1}'', \mathbf{x}_{i+1}')$, followed by $(ASK_1; \mathbf{x}_{i+1}', y_{i+1})$, and \mathbf{x}_{i+1} is the concatenation of \mathbf{x}_{i+1}' and y_{i+1} .

• Repeat until the position is $(ASK_1; \mathbf{x}_n)$, in which x occurs.

Since we will use this sort of system of rules again, lets cook up a formalism. For simplicity, we have a definition for when there is one guard constant d and one ((k + 1)-ary) guard relation P.

We want an abbreviation for: from $(s; \mathbf{x})$, Player \mathcal{Q} chooses y by selecting \mathbf{x}' and \mathbf{x}'' from d, \mathbf{x} , and then the game goes into a state $(s'; \mathbf{x}'', y)$. Suppose that s is an n-variable state, while s' is an m-variable state. Fix $M = (n + 1)^{m+k-1}$.

Definition 3.9 The traversal block for Player Q from s to s' is the sequence of M formulas as follows. First, let

- For each $i \in [M-2]$, τ_i is a map from [k+m-1] to [n+1], and
- For each $i \in [M-2]$, $\mathbf{x}_{\tau_i} = (x_{\tau_i(1)}, \dots, x_{\tau_i(k+m)})$, where $x_{n+1} = d$, and

• Given
$$\mathbf{x}' = (x'_1, \dots, x'_{k+m})$$
, let $\mathbf{x}'' = (x'_1, \dots, x'_k)$ and $\mathbf{x}''' = (x'_{k+1}, \dots, x'_{k+m})$.

Then let

$$(s; \mathbf{x}) :- (s_1; \mathbf{x}) \lor (s_2; \mathbf{x}_{\tau_1})$$

and for each $i \in [M-2]$,

$$(s_i; \mathbf{x}) := (s_{i+1}; \mathbf{x}) \lor (s''; \mathbf{x}_{\tau_{i+1}})$$

culminating with

$$(s_{M-2};\mathbf{x}) := (s'';\mathbf{x}_{\tau_{M-1}}) \vee (s'';\mathbf{x}_{\tau_M}).$$

Then let

$$(s''; \mathbf{x}'') := (Qy: P(\mathbf{x}''; y))(s'; \mathbf{x}''', y)$$

4 Using Game Programs

In this section, we will find that some logics correspond with natural classes of game programs. we will look at non-recursive games and find that non-recursive game programs corresponds to (guarded) first order logic. That will be straightforward. The rest of the section will be devoted to recursive games and (guarded) least fixed point logic.

4.1 Non-Recursive Games and First Order Logic

We first look at games with no recursive states. It turns out that these correspond to the guarded First Order queries.

Theorem 4.1 Let \mathcal{T} be a guard system. The queries expressible in FO_{*} $\upharpoonright \mathcal{T}$ are precisely the queries captured by non-recursive free games using guards from \mathcal{T} .

(Compare with the guard-free version in [HiK83].) We will need a notion of "subformula depth" of a FO formula.

Definition 4.1 The subformula depth (sfdepth) of a formula is computed as follows. If the formula is atomic or the negation of atomic, it is of subformula depth 0. And $sfdepth(\varphi * \psi) = \max\{sfdepth(\varphi)+1, sfdepth(\psi)+1\}\ if * is a junction, while <math>sfdepth((Qy: P(\mathbf{x}, y))\varphi(\mathbf{x}', y)) = sfdepth(\varphi) + 1.$

One warning: sfdepth is a different notion from the more popular "quantifier depth."

Proof of Theorem 4.1. First, suppose that θ is in FO_{*} $|\mathcal{T}|$. By Proposition 2.1, we can assume that all the negations have been pushed down to the atomic level. Decompose it into a game, as follows. We will construct a game program, whose states are precisely the subformulas of θ (counting repetitious occurences of the subformulas in θ), as follows. If $\varphi(\mathbf{v}) \equiv \psi(\mathbf{v}_1) * \delta(\mathbf{v}_2)$ is a subformula, where \mathbf{v}_1 and \mathbf{v}_2 are strings of variables from \mathbf{v} , and where * is a junction, then φ 's rule will be

$$(\varphi; \mathbf{v}) := (\psi; \mathbf{v}_1) * (\delta; \mathbf{v}_2).$$

If $\varphi(\mathbf{v}) \equiv (Qw; P(\mathbf{v}_1, w))\psi(\mathbf{v}_2)$, where \mathbf{v}_1 and \mathbf{v}_2 are strings of arguments from \mathbf{v} , then φ 's rule will be

$$(\varphi; \mathbf{v}) := (Qy: P(\mathbf{v}_1, y))(\psi; \mathbf{v}_2, y)$$

(where again \mathbf{v}_1 and \mathbf{v}_2 are appropriately chosen). And if $\varphi(\mathbf{v}) \equiv R(\mathbf{v}')$, where \mathbf{v}' is a string of arguments from \mathbf{v} , then φ 's rule is $(\varphi; \mathbf{v}) := -R(\mathbf{v}')$; while if $\varphi(\mathbf{v}) \equiv \neg R(\mathbf{v}')$ then its rule will be $(\varphi; \mathbf{v}) := -\neg R(\mathbf{v}')$. Finally, if $\varphi(\mathbf{v}) \equiv v' = c$, then its rule is $(\varphi; \mathbf{v}) := -v' = c$; if $\varphi(\mathbf{v}) \equiv v' \neq c$, then its rule is $(\varphi; \mathbf{v}) := -v' \neq c$

We claim that for any subformula φ and any \mathbf{x} , $\mathfrak{M} \models \varphi(\mathbf{x})$ iff Eloise wins from $(\varphi; \mathbf{x})$. We proceed by induction on the subformula depth sfdepth. Suppose that for any FO_{*} formula ψ of subformula depth less than r, Eloise wins from $(\psi; \mathbf{x})$ on \mathfrak{M} iff $\mathfrak{M} \models \psi(\mathbf{x})$. Suppose that sfdepth $(\varphi) = r$. We claim that Eloise wins from $(\varphi; \mathbf{x})$ on \mathfrak{M} iff $\mathfrak{M} \models \varphi(\mathbf{x})$. There are three cases.

Case \vdash : r = 0 and φ is atomic or the negation of atomic. Then Eloise wins at once iff $\mathfrak{M} \models \varphi(\mathbf{x}).$

Case *: φ is junctive: $\varphi \equiv \psi * \delta$ where max{sfdepth(ψ), sfdepth(δ)} = r - 1. There are two subcases: * is \lor or \land ; consider the subcase * is \lor . Then Eloise wins from (φ ; **x**) iff either she wins from (ψ ; **x**') or from (δ ; **x**''), where **x**' and **x**'' are defined appropriately from **x**. By the induction hypothesis, Eloise thus wins from (φ ; **x**) iff $\mathfrak{M} \models \psi(\mathbf{x}')$ or $\mathfrak{M} \models \delta(\mathbf{x}'')$. Thus Eloise thus wins from (φ ; **x**) iff $\mathfrak{M} \models \theta(\mathbf{x})$. The argument for * being \land is similar.

Case $Q: \varphi$ is quantitative: $\varphi \equiv (Qy: P(\mathbf{x}'; y)\psi(\mathbf{x}'', y))$, where $\mathrm{sfdepth}(\varphi) = r-1$. There are two subcases: Q is \exists or \forall ; consider the subcase Q is \forall . Then Eloise wins from $(\varphi; \mathbf{x})$ iff for any y such that $P(\mathbf{x}'; y)$, Eloise will go on to win from $(\psi; \mathbf{x}'', y)$. By the induction hypothesis, this is equivalent to: Eloise wins from $(\varphi; \mathbf{x})$ iff for any y such that $P(\mathbf{x}'; y)$, $\mathfrak{M} \models \psi(\mathbf{x}'', y)$, which holds iff $\mathfrak{M} \models (Qy: P(\mathbf{x}'; y))\psi(\mathbf{x}'', y)$.

The converse simply goes backwards: given a game, construct the subformulas, one per rule, reversing the construction of the previous paragraph. Again, by an easy induction, $\varphi(\mathbf{x})$ is true iff Player \mathcal{E} wins from $(\varphi; \mathbf{x})$.

4.2 Recursive Games and LFP Logic

Now, let's look at guarded Least Fixed Point logic. We will show that the guarded LFPexpressible queries are precisely those captured by recursive guarded game programs. The following proof is essentially a rearrangement of the proof in [Mc95a].

First, let's revisit the notion of stages in Definitions 2.10 and 3.4.

Say that an operative system $\varphi = \varphi_0, \ldots, \varphi_{\nu}$ is of subformula depth 1 if all φ_i are of subformula depth at most 1. We first observe that there is a correspondence between game programs and operative systems of depth 1: they are essentially variants of each other.

Definition 4.2 Let φ be a positive operative system of formulas of sfdepth 1, and let Φ be a game program. Then φ and Φ are associates if there is a one-to-one correspondence between formulas of φ and rules of Φ as follows. (We will use the states of Φ to index the formulas of φ .)

- 1. If * is a junction, and the formula $\varphi_s(\mathbf{v}) \equiv S_{s_1}(\mathbf{v}_1) * S_{s_2}(\mathbf{v}_2)$ is associated with the rule $(s; \mathbf{v}) := (s_1; \mathbf{v}_1) * (s_2; \mathbf{v}_2)$, then φ_{s_1} is associated with the rule for s_1 and φ_{s_2} is associated with the rule for s_2 .
- 2. If Q is a quantification, and $\varphi_s(\mathbf{v}) \equiv (Qw: P(\mathbf{v}_1, y))S_{s'}(\mathbf{v}_2, y)$ is associated with $(s; \mathbf{v}) :- (Qw: P(\mathbf{v}_1, y))(s'; \mathbf{v}_2, w)$, then $\varphi_{s'}$ is associated with the rule for s'.
- 3. If R is a relation symbol, then $\varphi_s(\mathbf{v}) \equiv R(\mathbf{v}')$ is associated with $(s; \mathbf{v}) := -R(\mathbf{v}')$ and $\varphi_s(\mathbf{v}) \equiv \neg R(\mathbf{v}')$ is associated with $(s; \mathbf{v}) := -\gamma R(\mathbf{v}')$. This is still true if R is =.

The following is elementary but crucial.

Lemma 4.1 If φ is associated with Φ , then on any joint database \mathfrak{M} , and each state s and each tuple \mathbf{x} from \mathfrak{M} , the stage of the induction equals the length of the rest of the game, i.e., $|\mathbf{x}|_s = |\mathbf{x}|_{\varphi_s}$.

Proof. This proof is by an induction on the stages. We will use the states of the game to index the formulas, and we will work on a fixed joint structure \mathfrak{M} . For any state s, $|\mathbf{x}|_s = 0$ iff the game is over iff $|\mathbf{x}|_{\varphi_s} = 0$ (iff s is terminal iff φ_s is atomic or the negation of an atomic). Now suppose that this Lemma was true of all tuples \mathbf{u} of stage $|\mathbf{u}|_s < n$, and suppose that $|\mathbf{x}|_s = n$. There are the usual four cases.

Suppose that s is disjunctive, i.e., that $\varphi_s \equiv \varphi_{s_1} \vee \varphi_{s_2}$, so that Eloise chooses to go to $(s_1; \mathbf{x}_1)$ or $(s_2; \mathbf{x}_2)$ from (s; x). As $|\mathbf{x}|_s = n$, Eloise could choose s_1 or s_2 and win within n-1 moves: min $\{|\mathbf{x}_1|_{s_1}, |\mathbf{x}_2|_{s_2}\} = n-1$. By induction, $|\mathbf{x}_1|_{\varphi_{s_1}} = |\mathbf{x}_1|_{s_1}$ and $|\mathbf{x}_2|_{\varphi_{s_2}} = |\mathbf{x}_2|_{s_2}$, and hence $|\mathbf{x}|_{\varphi_s} = |\mathbf{x}|_s = n$. Conjunction is similar.

Suppose that φ_s is universal: $\varphi_s(\mathbf{x}, \mathbf{S}) \equiv (\forall y: P(\mathbf{x}', y))S_{s'}(\mathbf{x}'', y)$, so that the sth rule is $(s; \mathbf{x}) := (\forall y: P(\mathbf{x}', y))(S'; \mathbf{x}'', y)$. Then for any tuple $\mathbf{a}, \mathfrak{M} \models \varphi_s^n(\mathbf{a})$ iff $\mathfrak{M} \models$

 $(\forall y: P(\mathbf{a}', y))\varphi_{S'}^{n-1}(\mathbf{a}'', y)$, i.e., for every y such that $P(\mathbf{a}', y)$, $\varphi_{S'}^{n-1}(\mathbf{a}'', y)$ is true. By the inductive hypothesis, this is true iff for every y satisfying $\mathfrak{M} \models P(\mathbf{a}', y)$, Eloise wins from $(s'; \mathbf{a}'', y)$ within n-1 moves, which is true iff Eloise wins $(s; \mathbf{a})$ within n moves, so $|\mathbf{x}|_{\varphi_s} = n = |\mathbf{x}|_s$. The argument for existential φ_i is similar.

Thus we can refer to the stage unambiguously as $|\mathbf{x}|_s$, where s is the state of the game, or index of the formula — provided that the operative system is of subformula depth 1. From this we can associate the formula φ_0 with the state START and get:

Corollary 4.1 If φ is associated with Φ (with the states of Φ indexing the formulas of φ), then on any joint database \mathfrak{M} , Eloise wins $\mathcal{G}(\Phi, \mathfrak{M})$ iff $\mathfrak{M} \models \varphi_0^{\infty}$.

Hence all game expressible queries are FO_{\star} + pos LFP expressible. The converse is also true. Almost.

Theorem 4.2 Restrict attention to uniformly connected guard systems. All free game programs can be captured by FO_{\star} + pos LFP and vice versa.

The theorem very similar to Theorem 2.1 of [Mc95a] (which is in fact the main theorem of [HaK84]). By Lemma 4.1 and Corollary 4.1, it suffices to prove:

Lemma 4.2 Every FO_{\star} + pos LFP expressible query can be expressed as a fixed point of a positive operative system with guarded quantification and of subformula depth 1.

Proof. To prove Lemma 4.2, it suffices to prove the following. Let $\varphi = \varphi_0, \ldots, \varphi_{\nu}$ be a positive operative system of formulas in which:

- for each k, i, φ_k is S_i -positive, and
- for each k, the only negations in the formula φ_k are negations of atomic subformulas (which we can require by Proposition 2.1 and the **S**-positivity of the formulas φ_0, \ldots).

Let $\varphi_{k,0}, \ldots, \varphi_{k,\eta_k}$ be the subformulas of φ_k , where $\varphi_{k,0} = \varphi_k$. We will construct a positive operative system

$$\psi_{0,0},\ldots,\psi_{0,\eta_0},\ldots,\psi_{k,j},\ldots,\psi_{
u,\eta_
u}$$

of formulas of subformula depth 1 and guarded quantification, such that for each k, $\varphi_k^{\infty} = \psi_{k,0}^{\infty}$.

Define the formulas $\psi_{k,h}$ as follows:

*. If
$$\varphi_{k,h}(\mathbf{x}) \equiv \varphi_{k,i}(\mathbf{x}') * \varphi_{k,j}(\mathbf{x}'')$$
, let $\psi_{k,h}(\mathbf{x}) \equiv T_{k,i}(\mathbf{x}') * T_{k,j}(\mathbf{x}'')$.
Q. If $\varphi_{k,h}(\mathbf{x}) \equiv (Qy; P(\mathbf{x}', y))\varphi_{k,i}(\mathbf{x}'', y)$, let $\psi_{k,h}(\mathbf{x}) \equiv (Qy; P(\mathbf{x}', y))T_{k,i}(\mathbf{x}'', y)$.
 \vdash . If $\varphi_{k,h}(\mathbf{x})$ is $R(\mathbf{x}')$ or $\neg R(\mathbf{x}')$, let $\psi_{k,h} = \varphi_{k,h}$. If $\varphi_{k,h}(\mathbf{x}) \equiv S_l(\mathbf{x}')$, let $\psi_{k,h}(\mathbf{x}) \equiv T_{l,0}(\mathbf{x}')$.

The result is a positive operative system ψ , associated via Lemma 4.1 with a game program Ψ .

Let $r = \max\{\text{sfdepth}(\varphi_i): i = 0, ..., \nu\}$. We claim that by induction on n and on the subformulas $\varphi_{k,h}$ that for all k, \mathbf{x} ,

(4.1)
$$\varphi_k^{n+1}(\mathbf{x}) \implies \varphi_{k,0}(\mathbf{x},\varphi^n) \implies \psi_{k,0}^{rn+r}(\mathbf{x}) \implies \varphi_k^{rn+r}(\mathbf{x}),$$

where $\varphi^m = \varphi^m_0, \ldots, \varphi^m_{\nu}$. This will imply that for each k, \mathbf{x} ,

$$\varphi_k^{\infty}(\mathbf{x}) \implies \varphi_{k,0}(\mathbf{x},\varphi^{\infty}) \implies \psi_{k,0}(\mathbf{x},\psi^{\infty}) \implies \psi_{k,0}^{\infty}(\mathbf{x}) \implies \varphi_k^{\infty}(\mathbf{x}),$$

and hence $\varphi_k^{\infty} = \psi_{k,0}^{\infty}$ for each k, and Lemma 4.2 follows.

We will actually prove that for each k, h, \mathbf{x}, n ,

$$\varphi_{k,h}(\mathbf{x},\varphi^n) \implies \psi_{k,h}^{rn+\mathrm{sfdepth}(\varphi_{k,h})}(\mathbf{x}) \implies \varphi_{k,h}(\mathbf{x},\varphi^{rn+\mathrm{sfdepth}(\varphi_{k,h})}),$$

from which Formula 4.1 follows by monotonicity. We have the usual cases.

 \vdash : sfdepth($\varphi_{k,h}$) = 0. If $\varphi_{k,h}(\mathbf{x})$ is $R(\mathbf{x}')$ or $\neg R(\mathbf{x}')$, then

$$\varphi_{k,h}(\emptyset,\ldots,\emptyset) \implies \psi^0_{k,h}(\mathbf{x}) \implies \varphi_{k,h}(\mathbf{x},\varphi^0)$$

as all three are the same atomic (or negated atomic) formula. And if $\varphi_{k,h}(\mathbf{x}) \equiv S_{l,0}(\mathbf{x}')$,

then by induction on n,

$$\begin{split} \varphi_{k,h}(\mathbf{x},\varphi^n) &\implies \varphi_{l,0}^{n+1}(\mathbf{x}') \\ &\implies \varphi_{l,0}(\mathbf{x}',\varphi^n) \\ &\implies \psi_{l,0}(\mathbf{x}',\psi^{rn}) \quad \text{by induction} \\ &\implies \psi_{k,h}^{rn+1}(\mathbf{x}') \\ &\implies \psi_{k,h}(\mathbf{x}',\psi^{rn}) \\ &\implies \varphi_{k,h}^{rn+1}(\mathbf{x}') \\ &\implies \varphi_{k,h}(\mathbf{x},\varphi^{rn}) \\ &\implies \varphi_{k,h}(\mathbf{x},\varphi^{rn}) \\ &\implies \varphi_{k,h}(\mathbf{x},\varphi^{rn+sdepth(\varphi_{k,h})}). \end{split}$$

*. Suppose that $\varphi_{k,h}$ is a conjunction. If $\varphi_{k,h}(\mathbf{x}, \mathbf{S}) \equiv \varphi_{k,i}(\mathbf{x}', \mathbf{S}) \wedge \varphi_{k,j}(\mathbf{x}'', \mathbf{S})$, then by induction on subformulas,

$$\begin{aligned} \varphi_{k,h}(\mathbf{x},\varphi^{n}) &\implies \varphi_{k,i}(\mathbf{x}',\varphi^{n}) \land \varphi_{k,j}(\mathbf{x}'',\varphi^{n}) \\ &\implies \psi_{k,i}^{rn+sfdepth(\varphi_{k,i})}(\mathbf{x}') \land \psi_{k,j}^{rn+sfdepth(\varphi_{k,h})-1}(\mathbf{x}'') \\ &\implies \psi_{k,i}^{rn+sfdepth(\varphi_{k,h})-1}(\mathbf{x}') \land \psi_{k,j}^{rn+sfdepth(\varphi_{k,h})-1}(\mathbf{x}'') \\ &\implies \varphi_{k,i}(\mathbf{x}',\varphi^{rn+sfdepth(\varphi_{k,h})-1}) \land \varphi_{k,j}(\mathbf{x}'',\varphi^{rn+sfdepth(\varphi_{k,h})-1}) \\ &\implies \varphi_{k,h}(\mathbf{x},\varphi^{rn+sfdepth(\varphi_{k,h})-1}) \\ &\implies \varphi_{k,h}(\mathbf{x},\varphi^{rn+sfdepth(\varphi_{k,h})}). \end{aligned}$$

Disjunction is similar.

Q. Suppose that $\varphi_{k,h}$ is an existential quantification. If

$$\varphi_{k,h}(\mathbf{x},\mathbf{S}) \equiv (\exists : P(\mathbf{x}';y))\varphi_{k,i}(\mathbf{x}'',y),$$

then by induction on subformulas again,

$$\begin{split} \varphi_{k,h}(\mathbf{x},\varphi^{n}) &\implies (\exists y: P(\mathbf{x}';y))\varphi_{k,i}(\mathbf{x}'',y,\varphi^{n}) \\ &\implies (\exists y: P(\mathbf{x}';y))\psi_{k,i}^{rn+sfdepth(\varphi_{k,i})}(\mathbf{x}'',y) \\ &\implies (\exists y: P(\mathbf{x}';y))\psi_{k,i}^{rn+sfdepth(\varphi_{k,h})-1}(\mathbf{x}'',y) \\ &\implies (\exists y: P(\mathbf{x}';y))\varphi_{k,i}(\mathbf{x}'',y,\varphi^{rn+sfdepth(\varphi_{k,h})-1}) \\ &\implies \varphi_{k,h}(\mathbf{x}'',y,\varphi^{rn+sfdepth(\varphi_{k,h})-1}) \\ &\implies \varphi_{k,h}(\mathbf{x}'',y,\varphi^{rn+sfdepth(\varphi_{k,h})}). \end{split}$$

Universal quantification is similar.

4.3 Least fixed point logic

Recall that we defined guarded and unguarded least fixed point logic in Definition 2.12, and we promised to compare their expressive power. In this subsection, we will compare their expressive power (and time complexity). We will find that (assuming uniform connectivity) they have the same expressive power, but that their computations differ in time complexity. Our time complexity measure will be the *stages of an induction*: recall the stages and closure ordinals from Definitions 2.10, 2.11, and 3.4. We will look at the effect of guarding quantifications on this measure. We will find that the closure ordinal of an FO_{*} + pos LFP induction on a structure \mathfrak{M} is on the order of the *p*-uniform radius times the closure ordinal of the "associated" FO + pos LFP induction.

Remember (Remark 2.2) that there is no *technical* difference between a database relation or constant and a guard relation or constant, so that in setting up an *unguarded* positive operative system of formulas, we can use "database" and "guard" relations and constants indiscriminately. And remember (Proposition 2.2) that guard relations and constants are FO_{*}-expressible, and thus the "guard" relations and constants are both (FO + pos LFP)-expressible and (FO_{*}+ pos LFP)-expressible. With these technicalities out of the way, we proceed to a result involving many more technicalities.

Theorem 4.3 Fix an integer p > 0. Let σ be a database schema and let ρ be a guard schema. Let \mathcal{M} be a uniformly connected class of joint structures — each of finite p-

uniform radius — of joint schema (σ, ρ) . Then FO_{*}+ pos LFP and FO + pos LFP have the same expressive power on \mathcal{M} .

Clearly, all (FO_{*} + pos LFP)-expressible queries on \mathcal{M} are (FO + pos LFP)-expressible. So we want to prove that given a positive operative system φ of formulas with unguarded quantifications, we can find a positive operative system ψ computing the same query (or queries). This proof is game-theoretic, and we will actually prove that there is a game program (with guarded quantification moves) that captures the queries generated by φ .

This is ... merely ... a matter of taking an unguarded positive operative system φ and developing an equivalent guarded game program Φ , i.e., such that for all $\mathfrak{M} \in \mathcal{M}$, all \mathbf{x} from $|\mathfrak{M}|$, and each $j, \mathfrak{M} \models \varphi_i^{\infty}(\mathbf{x})$ iff Eloise wins $\mathcal{G}(\Phi, \mathfrak{M})$ from $(s_j; \mathbf{x})$.

It will be convenient to restrict our attention to FO + pos LFP queries defined from systems of formulas of sfdepth 1.

Proposition 4.1 Every FO + pos LFP expressible query can be expressed as the least fixed point of a positive operative system of formulas of subformula depth 1.

The proof is the guardless version of Lemma 4.2, and we omit it.

So we presume that our unguarded positive operative system consists of formulas of subformula depth 1.

Replacing formulas by rules as in Definition 4.2 (1) and (3) — junctions and literals — is straightforward and we leave them to the reader.

The main problem is Definition 4.2 (2): simulating unguarded quantifications with guarded ones. We will proceed in two subsubsections:

- We construct (sub)games that represent computations (with guarded quantifications) for simulating unguarded existential and universal quantification.
- We use stages of the induction to prove that the unguarded quantifications are successfully simulated, and thus that the guarded game program captures the same query that the FO + pos LFP query did.

In the next subsection, we will see that the assumption that the structures have finite p-uniform radii is necessary by looking at an infinite counterexample.

It is not difficult to see how existential quantification can be simulated, by looking at a game program. Consider the following game. Eloise can start at any tuple of guard constants $\mathbf{x}_0 = \mathbf{d}$, and repeatedly choose *p*-tuples \mathbf{x}_1 , \mathbf{x}_2 , etc., where $\mathbf{x}_j = \mathbf{x}'_j$, *y*, where \mathbf{x}'_j is a (p-1)-tuples from \mathbf{x}_{j-1} and where \mathbf{x}''_j is a tuple of guard constants and entries from \mathbf{x}'_j (which in turn comes from \mathbf{x}_{j-1}), and $P(\mathbf{x}''_j; y)$ for some guard relation *P*. As the guard system is uniformly connected, Eloise can eventually reach any vertex, and thus if there exists a vertex *y* such that Eloise would win $\mathcal{G}(\Theta, ((\mathfrak{A}, \mathfrak{R}), y))$, she will eventually reach it, and go on to win. If no such vertex exists, she will search forever, and thus lose.

Simulating universal quantification is more difficult: it is not obvious how to have Abelard fail to find an element y satisfying $\neg \theta$ in order to justify $\forall y \theta(y)$. (This is the technical problem that arises from Convention 3.2.) We employ the same trick as in Figure 4 of [Mc95a]: we conduct a race. Suppose that an element y such that $\neg \theta(y)$ existed. Then starting from the database constants, Abelard should be able to find it. We do not want to give Abelard the opportunity to stall by wandering around the database, pretending to look for a possibly nonexistent y. So we devise a restriction in which Abelard must always move further and further away from his starting place: if he backtracks, Eloise can challenge his last move and have a chance to prove that Abelard was stalling. (What Eloise will do is challenge Abelard to a race which Eloise can win if Abelard was stalling.)

Notice that this algorithm will *not* work for databases of infinite (uniform) radius, for then Abelard's failure to find, within a *finite* amount of time, an element y such that $\neg \theta(y)$ cannot be taken as evidence that no such y exists.

We turn to simulating an unguarded universal quantification. Here is the idea. Imagine that we are simulating $\forall x \theta(x)$, assuming that there is one guard constant d and one binary guard relation P. Abelard is given the chance to find an x such that $\neg \theta(x)$. Starting from the guard constant $x_0 = d$, Abelard chooses x_1, x_2, \ldots in succession such that $P(x_i; x_{i+1})$ for each i. After each choice x_{i+1} , there is a brief deliberation:

1. Abelard may decide that x_{i+1} is what he wants, and the game continues from his denial of $\theta(x_{i+1})$.

- 2. Eloise may decide that Abelard is stalling, and challenge Abelard to a race from d: he is racing to x_i and she is racing to x_{i+1} . If he doesn't beat her, then he had *not* moved further away from d in his move from x_i to x_{i+1} , so it is fair that Eloise wins. But if he does beat her, then Eloise loses.
- 3. Both Abelard and Eloise decide not to take advantage of options (1) or (2), and Abelard now continues, choosing x_{i+2} such that $P(x_{i+1}; x_{i+2}), \dots$.

That's the idea. The rest of this section is devoted to formalizing this idea.

4.3.1 (Sub)Programs for Simulating Quantification

We first formalize the sort of translation done in Example 3.4. Notice that for logistical reasons, we are "refining" an operative system into a game program.

Definition 4.3 Given an operative system φ of (unguarded) formulas of depth 1, we obtain its guarded refinement Φ by making the following substitutions. Assume p-uniform connectedness. For simplicity, we assume that there are no database constants, one guard constant d, and one (k + 1)-ary guard relation P. (If we had database constants, we could either search for the relevant constant during each quantification cycle, or we could start the program with a search for all relevant database constants. This would add at most O(r) iterations, r being the uniform p-radius of the guard structure.) We replace individual formulas with subsystems of formulas as follows.

Literals. If $\varphi_i(\mathbf{x}, -) \equiv R(\mathbf{x}')$, let $(s_i; \mathbf{x}) := R(\mathbf{x}')$. Similarly, if $\varphi_i(\mathbf{x}, -) \equiv \neg R(\mathbf{x}')$, let $(s_i; \mathbf{x}) := \neg R(\mathbf{x}')$.

Junctions. If $\varphi_i(\mathbf{x}, -) \equiv S_j(\mathbf{x}') * S_k(\mathbf{x}'')$, let $(s_i; \mathbf{x}) :- (s_j; \mathbf{x}') * (s_k; \mathbf{x}'')$.

Existential Quantifications. If $\varphi_i(\mathbf{x}, -) \equiv \exists y S_j(\mathbf{x}', y)$, and letting **d** be a tuple of copies of the guard constant (and letting M_0 be the appropriate number of rearrangements

of tuples), let:

$$(s_i; \mathbf{x}) := (\exists y: d = y)(s_{i,1}; \mathbf{x}, y)$$

$$(s_{i,1}; \mathbf{x}, y) := (s_{i,1,1}; \mathbf{x}, y \dots, y) \lor (s_{i,1,1}; \mathbf{x}, y \dots, y)$$
is an appropriate traversal for Eloise (for tuples \mathbf{z}) from
$$(s_{i,1,1}; \mathbf{x}, \mathbf{z}) \text{ to } (s_{i,2}; \mathbf{x}, \mathbf{z}')$$

$$(s_{i,2}; \mathbf{x}, \mathbf{z}) := (\exists y: P(\mathbf{z}'; y))(s_{i,3}; \mathbf{x}, \mathbf{z}'', y)$$

$$(s_{i,3}; \mathbf{x}, \mathbf{z}, y) := (s_{i,1,1}; \mathbf{x}, \mathbf{z}', y) \lor (s_j; \mathbf{x}', y)$$

where \mathbf{z}' and \mathbf{z}'' are appropriate tuples of guard constants and arguments from \mathbf{z} , and \mathbf{x}' is an appropriate tuple from \mathbf{x} . This search will take up to $2 + (M_0 + 2r)r$ moves.

Universal Quantifications. If $\varphi_i(\mathbf{x}) \equiv \forall y S_j(\mathbf{x}', y)$, we set up a subsystem for Abelard's search and, since Eloise might grow impatient, for conducting the race. The subsystem of formulas is described below.

First, here is some nomenclature for these formulas:

- We use the *p*-tuples z⁻ and z⁺, and w⁻ and w⁺ to denote the position(s) of the pebbles for Eloise and Abelard in the race thus far, with respective goals z and w.
- Let **d** be the tuple of guard constants.
- Let M_1, M_2, M_3 be the appropriate numbers of rearrangements of tuples.
- Let z^{-'}, z^{-''}, z^{+'}, z^{+''}, etc., be the appropriate tuples of guard constants and arguments from z⁻, z⁺, etc., respectively.

Then a subsystem for conducting the universal quantification, i.e., Abelard's search, could be the following subsystems, which we break into several small pieces for clarity.

Before giving the precise subsystem, let's outline the idea. For simplicity, and without loss of much generality, suppose that the one guard relation P is binary, and that there is one guard constant d. Let r be the p-uniform radius. Abelard starts at the guard constant d; let $y_0 = d$. He then successively chooses y_1, y_2, \ldots (going through the traversal blocks) such that for each i, $P(y_i; y_{i+1})$. If he ever reaches y_k such that $\neg \theta(y_k)$, he announces the fact to poor Eloise, and goes on to win. If not, then as the structure is finite, he must reach a $k \leq r$ such that $\operatorname{dist}_P(d, y_k) = \operatorname{dist}_P(d, y_{k+1})$, where dist_P measures distance along *P*-arcs. This part of the game takes at most $(M_1 + 3)r$ moves. When this happens, Eloise challenges him to a race.

For brevity, we describe a combinatorial game, suppressing the game states and junctive moves.

They start at (d, y_k, d, y_{k+1}) , and move alternately, Eloise moving first. When it is Eloise's turn to move from (x, y_k, y, y_{k+1}) , she chooses y' such that P(y; y'), and then they are at (x, y_k, y', y_{k+1}) . Then Abelard moves similarly: if it is his turn to move from (x, y_k, y, y_{k+1}) , he chooses x' such that P(x; x'), the position is (x', y_k, y, y_{k+1}) . If $y = y_{n+1}$, Eloise wins. (Note that each of these "moves" are actually successions of moves through traversal blocks.) If not, then either $x' = y_k$ and Abelard wins, or $x' \neq y_k$ and it is now Eloise's turn to move.

Clearly, Eloise wins iff distance_P $(d, y_k) \leq \text{distance}_P(d, y_{k+1})$.

Now let's construct the precise subsystem for universal quantification.

First, Abelard goes out for his search. Note that after each new position \mathbf{w} is chosen (as opposed to his old position \mathbf{z}), Abelard can decide at $s_{i,4}$ that he's done (go to $(s_j; \mathbf{y})$), or if he wants to continue, Eloise can choose at $s_{i,6}$ whether to let him continue (go to $s_{i,2,1}$), or she can challenge him to a race (go to $s_{i,6}$).

$$(s_{i}; \mathbf{x}) := (\forall y: P(d = u))(s_{i,1}; \mathbf{x}', u)$$

$$(s_{i,1}; \mathbf{x}', u) := (s_{i,2,1}; \mathbf{x}', u \dots, u) \lor (s_{i,2,1}; \mathbf{x}', u \dots, u)$$

$$s_{i,2,1}, \dots, s_{i,2,M_1} \quad \text{is an appropriate traversal for Abelard}$$

$$(\text{for tuples } \mathbf{y}) \text{ from } (s_{i,2,1}; \mathbf{x}, \mathbf{z})$$

$$\text{to } (s_{i,3}; \mathbf{x}, \mathbf{z}, \mathbf{w}'),$$

$$(s_{i,3}; \mathbf{x}', \mathbf{z}, \mathbf{w}') := (\forall y: P(\mathbf{w}''; y))(s_{i,4}; \mathbf{x}', \mathbf{z}, \mathbf{w}', y), \quad \& \text{ if } \mathbf{w} = \mathbf{w}', y,$$

$$(s_{i,4}; \mathbf{x}', \mathbf{z}, \mathbf{w})) := (s_{j}; \mathbf{x}', y) \land (s_{i,5}; \mathbf{x}', \mathbf{z}, \mathbf{w})$$

$$(\text{condensing } \mathbf{w}', y \text{ to } \mathbf{w})$$

$$(s_{i,5}; \mathbf{x}', \mathbf{z}, \mathbf{w}) := (s_{i,2,1}; \mathbf{x}', \mathbf{w}) \lor (s_{i,6}; \mathbf{z}, \mathbf{w}).$$

If Eloise challenges Abelard's move from \mathbf{z} to \mathbf{w} , they conduct a race from d: Abelard to \mathbf{z} and Eloise to \mathbf{w} . If Eloise does not challenge Abelard to a race, this search will take up to $2 + (M_1 + 3)r$ moves.

Suppose Eloise challenges. Both players start from d, \ldots, d :

$$(s_{i,6}; (\mathbf{z}, \mathbf{w}) :- (\forall y: P(d = u))(s_{i,7,1}; \mathbf{z}, u, ..., u, \mathbf{w}, u, ..., u)$$

Then Abelard moves first:

$$\begin{aligned} s_{i,7,1}, \dots, s_{i,7,M_2} & \text{is an appropriate traversal for Abelard} \\ & (\text{for tuples } \mathbf{z}^-) \text{ from } (s_{i,7,1}; \mathbf{z}, \mathbf{z}^-, \mathbf{w}, \mathbf{w}^-) \\ & \text{to } (s_{i,8}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^-) \\ & (\forall y: P(\mathbf{z}^{+\prime\prime}; y))(s_{i,9,1}; \mathbf{z}, \mathbf{z}^{+\prime}, y, \mathbf{w}, \mathbf{w}^-). \end{aligned}$$

Letting $\mathbf{z}^+ = \mathbf{z}^{+\prime}, y$, we let Eloise move:

$$\begin{array}{ll} s_{i,9,1},\ldots,s_{i,9,M_3} & \text{is an appropriate traversal for Eloise} \\ & (\text{for tuples } \mathbf{w}^-) \text{ from } (s_{i,9,1};\mathbf{z},\mathbf{z}^+,\mathbf{w},\mathbf{w}^-) \\ & \text{to } (s_{i,10};\mathbf{z},\mathbf{z}^+,\mathbf{w},\mathbf{w}^+) \\ & (s_{i,10};\mathbf{z},\mathbf{z}^+,\mathbf{w},\mathbf{w}^+,-) & :- & (\exists y: P(\mathbf{w}^{+\prime\prime};y))(s_{i,11};\mathbf{z},\mathbf{z}^+,\mathbf{w},\mathbf{w}^{+\prime},y), \end{array}$$

and letting $\mathbf{z}^+ = \mathbf{w}^{+\prime}, y$, we ask: has someone won the race? This means comparing the tuple \mathbf{w}^+ to \mathbf{w} (and if equality holds, Eloise wins) and then, that failing, \mathbf{z} to \mathbf{z}^+ (and if equality holds, Abelard wins). Notice that we have set it up so that if Abelard announces

he wants to compare \mathbf{z} and \mathbf{z}^+ , then Eloise wins iff they are different.

$$(s_{i,11}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+) := (s_{i,12}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+) \lor (s_{i,11,1}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+)$$
and for $l = 1, ..., p - 1$,
$$(s_{i,11,l}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+) := (s_{i,12}; w_l, w_l^+) \lor (s_{i,11,l+1}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+)$$
while:
$$(s_{i,11,p}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+) := (s_{i,12}; w_p, w_p^+) \lor (s_{i,13}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+)$$

$$(s_{i,12}; u, v) := u = v$$

$$(s_{i,13}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+) := (s_{i,7,1}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+) \land (s_{i,14,1}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+)$$
and for $l = 1, ..., p - 2$,
$$(s_{i,14,l}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+) := (s_{i,15}; z_l, z_l^+) \lor (s_{i,14,l+1}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+)$$

$$while:$$

$$(s_{i,14,p-1}; \mathbf{z}, \mathbf{z}^+, \mathbf{w}, \mathbf{w}^+) := (s_{i,15}; z_{p-1}, z_{p-1}^+) \lor (s_{i,15}; z_p, z_p^+)$$

$$(s_{i,15}; u, v) := u \neq v.$$

Let $M = \max\{M_0, M_1 + M_2 + M_3\}$. Notice that if the radius of the guard structure is r, then this race takes at most $p + 4 + (M_2 + M_3 + 2)r$ moves. And whether or not Eloise wins, if the radius of the guard structure is r, then getting to $(s_j(\mathbf{x}', y), \text{ or to the end of the game, takes at most } p + 6 + (M_1 + M_2 + M_3 + 5)r \le (M + 5)r + p + 6$ moves.

Remark 4.1 If the guard relation is not binary, then assuming p-uniform connectedness, the race ends if Eloise reaches her triple before Abelard reaches his, and checking that the appropriate tuple is reached takes an additional p moves.

Remark 4.2 If an operative system, or game program, is of $n \ge p$ variables, and has one (k + 1)-ary guard relation, and there is one guard constant, then $M_0, M_1, M_2, M_3 \le (n + 1)^{n+k}$. Call $\mu = (n + 1)^{n+k}$ the system or program's traversal number.

4.3.2 The Simulation Works

Now let's check this proposed system of refinements and see if it does the job.

Lemma 4.3 Let φ be a positive operative system of unguarded formulas of depth 1, and traversal number μ . Let Φ be the guarded refinement of φ , so that for each formula φ_i , s_i is the corresponding state in the program Φ . Fix a joint structure. Then for each *i*, and each **x**, if *r* is the radius of the guard system, then

(4.2)
$$|\mathbf{x}|_{\varphi_i} \le |\mathbf{x}|_{s_i} \le (3\mu+5)r|\mathbf{x}|_{\varphi_i} + r + 2p + 6.$$

(The last "+r" is in case there are any database constants, a "+p" for checking at the end of a race as in Remark 4.1, and the last "+p+6" for the end of a challenge race in the universal subroutine.) Thus $|\mathbf{x}|_{\varphi_i} < \infty$ iff $|\mathbf{x}|_{s_i} < \infty$.

Proof. We prove the two inequalities of Formula 4.2 separately by induction on the stages.

First, we prove by induction on n that for each i and \mathbf{x} ,

$$|\mathbf{x}|_{\varphi_i} \ge n \implies |\mathbf{x}|_{s_i} \ge n.$$

This is true if n = 0. Suppose that this is true for all m < n, and suppose that $|\mathbf{x}|_{\varphi_i} \ge n$. We have the usual cases.

*. If φ_i is, say, conjunctive $(\varphi_i \equiv \varphi_j \land \varphi_k)$, then:

$$\begin{aligned} |\mathbf{x}|_{\varphi_i} \ge n &\implies |\mathbf{x}'|_{\varphi_j} \ge n-1 \text{ or } |\mathbf{x}''|_{\varphi_k} \ge n-1 \\ &\implies |\mathbf{x}'|_{s_j} \ge n-1 \text{ or } |\mathbf{x}''|_{s_k} \ge n-1 \quad \text{ (by induction)} \\ &\implies |\mathbf{x}|_{s_i} \ge n. \end{aligned}$$

Disjunction is similar.

Q. If φ_i is, say, existential $(\varphi_i \equiv \exists y \varphi_j)$, then:

$$\begin{aligned} |\mathbf{x}|_{\varphi_i} \ge n &\implies \text{ for all } y, |\mathbf{x}', y|_{\varphi_j} \ge n-1 \\ &\implies \text{ for all } y, |\mathbf{x}', y|_{s_j} \ge n-1 \qquad \text{(by induction)} \\ &\implies |\mathbf{x}|_{s_j} \ge n. \end{aligned}$$

Universal quantification is similar.

This concludes the proof of the left inequality of Formula 4.2.

Second, we prove by induction on n that for each i and \mathbf{x} ,

$$|\mathbf{x}|_{\varphi_i} = n \implies |\mathbf{x}|_{s_i} \le (3\mu + 5)rn + r + 2p + 6.$$

This is true if n = 0. Suppose that this is true for all m < n, and suppose that $|\mathbf{x}|_{\varphi_i} = n$. Again, we have the usual cases.

*. If φ_i is, say, disjunctive $(\varphi_i \equiv \varphi_j \lor \varphi_k)$, then:

$$\begin{aligned} |\mathbf{x}|_{\varphi_i} &= n \implies |\mathbf{x}'|_{\varphi_j} \le n - 1 \text{ or } |\mathbf{x}''|_{\varphi_k} \le n - 1 \\ \implies |\mathbf{x}'|_{s_j} \le (3\mu + 5)(n - 1)r + r + 2p + 6 \\ &\text{ or } |\mathbf{x}''|_{s_k} \le (3\mu + 5)(n - 1)r + r + 2p + 6 \\ & \text{ (by induction)} \\ \implies |\mathbf{x}|_{s_i} \le (3\mu + 5)(n - 1)r + r + 2p + 7 \le (3\mu + 5)nr + r + 2p + 6. \end{aligned}$$

Conjunction is similar.

Q. If φ_i is, say, universal $(\varphi_i \equiv \forall y \varphi_j)$, then:

$$\begin{aligned} |\mathbf{x}|_{\varphi_{i}} &= n \implies \text{ for all } y, |\mathbf{x}', y|_{\varphi_{j}} \leq n-1 \\ \implies \text{ for all } y, |\mathbf{x}', y|_{s_{j}} \leq (3\mu+5)(n-1)r + r + 2p + 6 \quad (\text{by induction}) \\ \implies |\mathbf{x}|_{s_{j}} \leq \max\{(3\mu+5)(n-1)r + (3\mu+5)r + r + 2p + 6, \\ (3\mu+5)r + p + 6\} \\ \implies |\mathbf{x}|_{s_{j}} \leq (3\mu+5)nr + r + 2p + 6. \end{aligned}$$

The last inequality holds because it takes at most $(3\mu + 5)r$ moves to search through the structure and/or conduct a race as in the algorithm of Definition 4.3, and then either a last run of p+6 moves, or the remaining $(3\mu + 5)(n-1)r + (3\mu + 5)r + r + 2p + 6$ moves — but not both. Existential quantification is similar, if simpler, and the search takes only $2 + (\mu + 2)r$ moves.

This concludes the proof of the right inequality of Formula 4.2. \blacksquare

And now for the punch line:

Proof of Theorem 4.3. By Lemma 4.2, all FO + pos LFP expressible queries are fixed points of operative systems of subformula depth 1. Any operative system φ of subformula

depth 1 has a guarded refinement Φ , and by Lemma 4.3, for each φ_i of φ , there is a s_i of Φ such that on any joint structure \mathfrak{M} , and any \mathbf{x} from $|\mathfrak{M}|, |\mathbf{x}|_{\varphi_i} < \infty$ iff $|\mathbf{x}|_{s_i} < \infty$. Thus, for φ_0 being the formula corresponding to $s_0 = \text{START}, \mathfrak{M} \models \varphi_0^{\infty}$ iff Eloise wins $\mathcal{G}(\Phi, \mathfrak{M})$.

Finally, since FO + pos LFP is closed under negation on finite structures ([I86]), so is FO_{\star} + pos LFP.

4.4 Closure Under Negation

We conclude this section with the comment that things are different on infinite structures. Let FO + LFP be the boolean closure of FO + pos LFP, and let $FO_* + LFP$ be the boolean closure of $FO_* + pos LFP$.

By [I86], FO + pos LFP has the same expressive power as FO + LFP on finite databases. Thus on finite databases with connected guard systems, FO_{*} + pos LFP has the same expressive power as FO_{*} + LFP. The situation is entirely different for infinite databases, in which it is possible FO + pos LFP is not closed under negation (see [Mo74]). In addition, Theorem 4.3 is false for infinite structures. Suppose that you had a joint structure, whose domain was the nonnegative integers (\mathbb{N}) and whose guard relations were $\operatorname{succ}(x, y) \equiv y = x + 1$ and $\operatorname{pred}(x, y) \equiv \operatorname{succ}(y, x)$, and whose guard constant was 0. We get the joint structure $\mathfrak{N} = (\langle \mathbb{N}, 0 \rangle, \langle \mathbb{N}, \operatorname{pred}, \operatorname{succ}, 0 \rangle).$

Theorem 4.4 There exists a FO + pos LFP expressible relation that is not FO_{\star} + pos LFP expressible on \mathfrak{N} .

To prove this, we will need a lemma. Let ω be the least transfinite ordinal.

Lemma 4.4 For each operative system φ of **S**-positive formulas with guarded quantifications, $\sup |\mathbf{x}|_{\varphi}^{\mathfrak{N}} \leq \omega$, where the supremum is not achieved.

Proof of Lemma 4.4. Suppose otherwise: for some s and some \mathbf{x} , $|(s; \mathbf{x})|^{\mathfrak{N}} = \omega$. Then either s is a junctive state, in which whoever is to play has two choices to choose from, or s is a quantitative state, in which case the player who is to play has to choose a predecessor

or successor of one of the finitely many elements of \mathbf{x} (or of 0) to continue from. Either way, the player who is to play has finitely many choices to choose from: list them as $(s_1; \mathbf{x}_1), \ldots, (s_k; \mathbf{x}_k)$. If the player is Eloise, at least one of these is of a stage $n < \omega$, in which case $|(s; \mathbf{x})|^{\mathfrak{N}} \leq n + 1$, contradicting $|(s; \mathbf{x})|^{\mathfrak{N}} = \omega$. If the player is Abelard, then all of the finitely many options are of stages less than ω , and hence their maximum is a number $m < \omega$, forcing $|(s; \mathbf{x})|^{\mathfrak{N}} = m + 1$, again contradicting $|(s; \mathbf{x})|^{\mathfrak{N}} = \omega$. Getting a contradiction either way, we conclude that $|(s; \mathbf{x})|^{\mathfrak{N}} < \omega$ for all s, \mathbf{x} .

Proof of Theorem 4.4. We will use some facts about FO + pos LFP from [Mo74]. First of all, note that all the FO_{*} + pos LFP expressible relations on \mathfrak{N} are FO + pos LFP expressible. There are FO + pos LFP inductions whose closure ordinals are greater than ω , e.g., letting S_0 and S_2 range over the "0-ary" relations TRUE and FALSE,

$$\begin{split} \varphi_0(S_0, S_1, S_2) &\equiv S_2, \\ \varphi_1(x, S_0, S_1, S_2) &\equiv x = 0 \lor \exists y (\texttt{pred}(x, y) \land S_1(y)), \\ \varphi_2(S_0, S_1, S_2) &\equiv \forall y S_1(y). \end{split}$$

whose closure ordinal is $\omega + 1$. In [Mo74], a relation is called *hyperelementary* if both it and its complement are in FO + pos LFP. The Closure Theorem, [Mo74, Thm. 2B.4], says that if ψ is an operative system of positive formulas, and if its closure ordinal on \mathfrak{N} is not maximal on \mathfrak{N} , then ψ^{∞} is hyperelementary. Thus all FO_{*} + pos LFP relations on \mathfrak{N} are hyperelementary. But there is a relation — the universal relation for FO + pos LFP relations on \mathfrak{N} (see [Mo74]) — that is not hyperelementary but still FO + pos LFP expressible.

In fact, the industrious reader can confirm that:

Proposition 4.2 On \mathfrak{N} , the FO_{*} + pos LFP expressible relations are precisely the classically semirecursive relations.

5 Excelsior

In this paper, we saw what happened to Least Fixed Point logic when we use guarded quantification. It turns out that guarded Least Fixed Point logic behaves very similarly:

the Stage Comparison Theorem still holds, two popular measures of descriptive complexity seem to behave similarly, and so on. In other words, what increases is the complexity. We have to worry about things like connectivity of the guard structures, and about safety. But the underlying results are the same. This may seem dull, but the nice thing is that we can develop theorems about what can be done in FO + LFP, and then we know that they can be done in FO_* + LFP as well.

We will take advantage of this in a sequel. In [Mc*], we generalize a conjecture of [ChH82]: over any class of structures admitting unbounded inductions of arbitrarily high "dimension,", i.e., there exist FO + pos LFP queries requiring second order recursion variables of arbitrarily high arity. This conjecture has been proven for FO + pos LFP on the class of all finite structures in [Gro96]; we will prove it for FO_{*}+ pos LFP on all classes of joint structures in which the guard system is sufficiently "sparse."

Let us close with a more basic question. Let's first take another look at the topology of games, from Subsection 3.3. Recall that a game flowchart has several strongly connected subdigraphs, which we could call *subroutines*:

Definition 5.1 Given a game program Φ , a subroutine is a maximal set of states Γ from Φ such that for any $s, s' \in \Gamma$, $s \vdash^+ s'$.

In simulating unguarded existential quantification, we constructed a subroutine in which all quantifications were existential (following [Mc95a], we can call this an *existential subroutine*). However, thanks to the asymmetry induced by Convention 3.2, the game program code simulating unguarded universal quantification included a race, which forms (part of) a subroutine in which both guarded existential and guarded universal quantification occurred: following [Mc95a], such a subroutine could be called *alternating*.

In [Ko91], it was proven that there exist FO + pos LFP expressible queries which could not be computed by game programs (with *unguarded* quantification) lacking alternating subroutines. This leads to a number of papers on the fine structure of alternation (or lack thereof), such as [D87], [B1G87], [Gra92], [GraM96], [Mc95a], etc. And it leaves us with the question: in order to simulate unguarded universal quantification, was alternation necessary? **Conjecture 5.1** There exists a first order sentence in the language of graph theory that cannot be represented by a game program with guarded quantifications and no alternating subroutines.

In fact, we suspect that one such first order sentence is:

$$\forall x \forall y \forall z [\neg \operatorname{Edge}(x, y) \lor \neg \operatorname{Edge}(x, z) \lor \neg \operatorname{Edge}(y, z)].$$

References

- [AbaLW89] M. Abadi, L. Lamport & P. Wolper, Realizable and unrealizable specifications of reactive systems, in: G. Ausiello, et al, eds., Proc. 16th ICALP (Stresa, Italy) (Springer Lect. Notes C.S. 372, 1989), 1–17.
- [AbiHV95] S. Abiteboul, R. Hull & V. Vianu, Foundations of Databases (Addison-Wesley, 1995).
- [Ac75] P. Aczel, Quantifiers, games and inductive definitions, Proc. 3rd Scandinavian Logic Symp. (North-Holland, 1975), 1–14.
- [AhU79] A. Aho & J. Ullman, Universality of data retrieval languages, Proc. 6th ACM
 Symp. Principles of Programming Languages (1979), 110-117.
- [AnBN96] H. Andréka, J. van Benthem & I. Németi, Modal Languages and Bounded Fragments of Predicate Logic, ILLC Research Repport and Technical Notes Series (1996).
- [BacW98] R.-J. Back & J. von Wright, Refinement Calculus: A Systematics Introduction (Springer, 1998).
- [Bar77] J. Barwise, On Moschovakis Closure Ordinals, J. Sym. Logic 42 (1977), 292–296.
- [BarM96] J. Barwise & L. Moss, Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena (CSLI, 1996).

- [Bi92] K. Binmore, Fun and Games: A Text on Game Theory (Heath, 1992).
- [BIG87] A. Blass & Y. Gurevich, Existential fixed-point logic, in: E. Börger, ed., Computation Theory and Logic (Lecture Notes in C.S. 270, Springer, 1987), 20-36.
- [CaFI92] J.-Y. Cai, M. Fürer & N. Immerman, An optimal lower bound on the number of variables for graph identification, Combinatorica 12:4 (1992), 389-410.
- [ChH82] A. Chandra & D. Harel, Structure and complexity of relational queries J.
 Comp. Sys. Sci. 25 (1982), 99–128.
- [C178] K. Clark, Negation as Failure, in: H. Gallaire & J. Minke, eds., Logic and Databases, (Plenum Pr., 1978), 293–322.
- [Com83] K. Compton, Some useful preservation theorems, J. Sym. Logic 48:2 (1983), 427–440.
- [D87] E. Dahlhaus, Skolem normal forms concerning the least fixed point, in: E.
 Börger, ed., Computation Theory and Logic (Lecture Notes in C.S. 270, Springer, 1987), 101–106.
- [EbF95] H.-D. Ebbinghaus & J. Flum, **Finite Model Theory** (Springer-Verlag, 1995).
- [Eh61] A. Ehrenfeucht, An application of games to the completeness problem for formalized theories, Fund. Math. 49 (1961), 129–141.
- [F74] R. Fagin, Generalized first-order spectra and polynomial time recognizable sets, in: R. Karp, ed., Complexity of Computations (SIAM-AMS Proc. 7, 1974), 43-73.
- [FHMV95] R. Fagin, J. Halpern, Y. Moses, M. Vardi, Reasoning about Knowledge (MIT Pr., 1995).
- [GaS53] D. Gale & F. M. Stewart, Infinite games with perfect information, Ann. Math.
 Stud. 28 (1953), 245–266.

- [Gra92] E. Grädel, On transitive closure logic, in: Proc. 5th Workshop on Computer Science Logic (CSL'91) (Lect. Notes in C. S. 626, Springer, 1992) 149–163.
- [GraM96] E. Grädel & G. McColm, Hierarchies in transitive closure logic, stratified Datalog and infinitary logic, Ann. Pure & Appl. Logic 77 (1996) 169–199.
- [GraW99] E. Grädel & I. Wakiewicz, Guarded Fixed Point Logic, Proc. 14th IEEE Symp. on Logic in Computer Science (1999), 45–54.
- [Gro96] M. Grohe, Arity hierarchies, Ann. Pure and Applied Logic 82 (1996), 103–163.
- [Gu97] Y. Gurevich, personal communication, 1997.
- [GuS86] Y. Gurevich & S. Shelah, Fixed-point extensions of first order logic, Ann.Pure Appl. Log. 32 (1986), 265-280.
- [HaK84] D. Harel & D. Kozen, A programming language for the inductive sets, and applications, Information and Control 63 (1984), 118-139.
- [He94] W. H. Hesselink, Nondeterminism and recursion via games and stacks, Theoretical Computer Science 124 (1994), 273–295.
- [Hil82] R. Hilpenen, On C. S. Peirce's theory of the proposition: Peirce as a precursor of game-theoretic semantics **The Monist 62** (1982), 182–189.
- [Hin72] J. Hintikka, Language Games and Information (Clarendon, 1972).
- [HiK83] J. Hintikka & J. Kulas, The Game of Language: Studies in Game-Theoretical Semantics and its Applications (D. Reidel, 1983).
- [HiS97] J. Hintikka & G. Sandu, Game-Theoretic Semantics, in: J. van Benthem & A. ter Meulen, eds., Handbook of Logic & Language (MIT Pr. & North-Holland, 1997), 361–410.

- [I81] N. Immerman, Number of Quantifiers is Better than Number of Tape Cells, J.
 Computer and System Sciences 22 (1981), 384–406.
- [I82] N. Immerman, Upper and lower bounds for first order expressibility, J. Computer and System Sciences 25 (1982), 76–98.
- [I86] N. Immerman, Relational Queries Computable in Polynomial Time, Inform.
 & Control 68 (1986), 86–104.
- [I87] N. Immerman, Languages that capture complexity classes, SIAM J. Computing 16 (1987), 760–778.
- [I99] N. Immerman, **Descriptive Complexity** (Springer-Verlag, 1999).
- [Ka91] P. Kanellakis, *Elements of relational database theory*, in: J. van Leeuwen, ed.,
 Handbook of Theoretical Computer Science (Elsevier, 1991), 1074–1156.
- [Ko85] P. Kolaites, Game Quantification, in: J. Barwise & S. Feferman, eds., Model-Theoretic Logics (Springer-Verlag, 1985), 365–421.
- [Ko91] P. Kolaitis, The Expressive Power of Stratified Logic Programs, Information & Comp. 90 (1991), 50–66.
- [Mc89] G. McColm, Some restrictions on simple fixed points of the integers, J. Sym. Log. 54:4 (1989), 1324–1345.
- [Mc90a] G. McColm, Parametrization over inductive relations of a bounded number of variables, Ann. Pure and Applied Logic 48 (1990), 103–134.
- [Mc90b] G. McColm, When is Arithmetic Possible?, Ann. Pure and Applied Logic 50 (1990), 29–51.
- [Mc95a] G. McColm, Pebble Games and Subroutines in Least Fixed Point Logic, Information & Comp. 122:2 (1995), 201–220.
- [Mc95b] G. McColm, Dimension Versus Number of Variables, and Connectivity, too, Math. Log. Quart. 41 (1995), 111–134.

- [Mc*] G. McColm, The Dimension of Guarded LFP Queries, in preparation.
- [Mi99] R. Milner, Communicating and Mobile Systems: the π -Calculus (Cambridge U. Pr., 1999).
- [Mo72] Y. Moschovakis, The Game Quantifier, **Proc. AMS 31:1** (1972), 245-250.
- [Mo74] Y. Moschovakis, Elementary induction on abstract structures, (North-Holland, 1974).
- [Mo80] Y. Moschovakis, **Descriptive Set Theory** (North-Holland, 1980).
- [Mo83] Y. Moschovakis, Abstract recursion as a foundation for the theory of algorithms, in: M. M. Richter, et al, Computation and proof theory, Lect. Notes in Math. 1104 (Springer-Verlag, Berlin, 1983), 289-364.
- [Mo91] Y. Moschovakis, A model of concurrency with fair merge and full recursion, Information & Computation 93 (1991), 114–171.
- [Ot97] M. Otto, Bounded variable logics and counting: a study in finite models (Springer-Verlag, Lect. Notes Logic 9, 1997).
- [Pa85] R. Parikh, The Logic of Games and its Applications, Ann. Disc. Math. 24 (1985), 111–140.
- [Pl81] G. Plotkin, A Structural Approach to Operational Semantics ((DAIMI FN 19, Computer Science Dept., Aarhus U., 1981).
- [Ro67] H. Rogers, Jr., Theory of Recursive Functions and Effective Computability (McGraw-Hill, 1967).
- [U88, 89] J. D. Ullman, Principles of Database Systems and Knowledge Base Systems I & II (Computer Science Pr., 1988, 1989).
- [V82] M. Vardi, Complexity of relational database systems, Proc. 14th ACM Symposium on the Theory Of Computing (1982), 137–146.