Finite Model Theory and Descriptive Complexity

Erich Grädel

This chapter deals with the relationship between logical definability and computational complexity on finite structures. Particular emphasis is given to *game-based evaluation algorithms* for various logical formalisms and to *logics capturing complexity classes*.

In addition to the most common logical systems such as first-order and second-order logic (and their fragments), this survey focuses on algorithmic questions and complexity results related to fixed-point logics (including fixed-point extensions of first-order logic, the modal μ-calculus, the database query language Datalog, and fixed-point logics with counting).

Finally, it is discussed how the general approach and the methodology of finite model theory can be extended to suitable domains of infinite structures. As an example, some results relating *metafinite model theory* to complexity theory are presented.

3.1 Definability and Complexity

One of the central issues in finite model theory is the relationship between logical definability and computational complexity. We want to understand how the expressive power of a logical system – such as first-order or second-order logic, least fixed-point logic, or a logic-based database query language such as Datalog – is related to its algorithmic properties. Conversely, we want to relate natural levels of computational complexity to the defining power of logical languages, i.e., we want *logics that capture complexity classes*.[1]

The aspects of finite model theory that are related to computational complexity are also referred to as *descriptive complexity theory*. While computational complexity theory is concerned with the computational resources such as time, space, or the amount of hardware that are necessary to decide a property, descriptive complexity theory asks for the logical resources that

[1] For a potential application of such results, see Exercise 3.5.32.

are necessary to define it. In this chapter we shall give a survey of descriptive complexity theory. We shall assume that the reader is familiar with fundamental notions of logic and complexity theory. Specifically we assume familiarity with first-order logic and with deterministic and non-deterministic complexity classes. See the appendix to this chapter for a brief survey on alternating complexity classes.

In Sect. 3.1, we discuss some basic issues concerning the relationship between logic and complexity, we introduce model-checking games, and we determine in a detailed way the complexity of first-order model checking.

In Sect. 3.2, we make precise the notion of a logic capturing a complexity class. As our first capturing result, we prove Fagin's Theorem, which says that existential second-order logic captures NP. In a limited scenario, namely for the domain of ordered structures, we then derive capturing results for a number of other complexity classes, including PTIME and LOGSPACE, by use of fragments of second-order logic (such as second-order Horn logic) and by extensions of first-order logic (such as transitive closure logics).

Section 3.3 is devoted to fixed-point logics. These are probably the most important logics for finite model theory and also play an important role in many other fields of logic in computer science. We shall discuss many variants of fixed point logics, including least, inflationary and partial fixed point logic, the modal μ-calculus, and the database query language Datalog. We shall explain model checking issues, capturing results for PTIME and PSPACE, and also discuss structural issues for these logics.

In Sect. 3.4 we introduce logics with counting. One of the limitations of common logics on finite structures is an inability to count. By adding to first-order logic and, in particular, to fixed-point logic an explicit counting mechanism, one obtains powerful logics that come quite close to capturing PTIME.

Section 3.5 is devoted to capturing results on certain specific domains of unordered structures, via a technique called canonization. While the general problem of whether there exists a logic capturing PTIME on all finite structures is still open (and it is widely conjectured that no such logic exists), canonization permits us to find interesting domains of structures where fixed-point logic or fixed-point logic with counting can express all of PTIME.

Finally, in Sect. 3.6 we discuss the extension of the general approach and methods of finite model theory to suitable domains of infinite structures, i.e., the generalization of finite model theory to an *algorithmic model theory*. We discuss several domains of infinite structures for which this approach makes sense, and then treat, as an example, the domain of *metafinite structures*, for which capturing results have been studied in some detail.

3.1.1 Complexity Issues in Logic

One of the central issues in the relationship between complexity theory and logic is the *algorithmic complexity of the common reasoning tasks for a logic.*

There are numerous such tasks, but most of them can be easily reduced to two (at least for logics with reasonable closure properties), namely *satisfiability testing* and *model checking*. The **satisfiability problem** for a logic L on a domain $\mathcal{D}$ of structures takes formulae $\psi \in L$ as inputs, and the question to be answered is whether there exists in $\mathcal{D}$ a model for ψ. Although satisfiability problems are of fundamental importance in many areas of logic and its applications, they do not really play a crucial role in finite model theory. Nevertheless, they are considered occasionally and, moreover, some of the central results of finite model theory have interesting connections with satisfiability problems. We shall point out some such relations later.

On the other hand, model-checking problems occupy a central place in finite model theory. For a logic L and a domain $\mathcal{D}$ of (finite) structures, the **model-checking problem** asks, given a structure $\mathfrak{A} \in \mathcal{D}$ and a formula $\psi \in L$, whether it is the case that $\mathfrak{A} \models \psi$. A closely related problem is **formula evaluation** (or query evaluation): given a structure $\mathfrak{A}$ and a formula $\psi(\overline{x})$ (with free variables $\overline{x}$), the problem is to compute the relation defined by ψ on $\mathfrak{A}$, i.e. the set $\psi^{\mathfrak{A}} := \{\overline{a} : \mathfrak{A} \models \psi(\overline{a})\}$. Obviously, the evaluation problem for a formula with k free variables on a structure with n elements reduces to n^k model-checking problems.

Note that a model-checking problem has two inputs: a structure and a formula. We can measure the complexity in terms of both inputs, and this is what is commonly refered to as the **combined complexity** of the model-checking problem (for L and $\mathcal{D}$). However, in many cases, one of the two inputs is fixed, and we measure the complexity only in terms of the other. If we fix the structure $\mathfrak{A}$, then the model-checking problem for L on this structure amounts to deciding $\mathrm{Th}_L(\mathfrak{A}) := \{\psi \in L : \mathfrak{A} \models \psi\}$, the ***L*-theory** of $\mathfrak{A}$. The complexity of this problem is called the **expression complexity** of the model-checking problem (for L on $\mathfrak{A}$). For first-order logic (FO) and for monadic second-order logic (MSO) in particular, such problems have a long tradition in logic and numerous applications in many fields. Of even greater importance for finite model theory are model-checking problems for a fixed formula ψ, which amounts to deciding the **model class** of ψ inside $\mathcal{D}$, $\mathrm{Mod}_{\mathcal{D}}(\psi) := \{\mathfrak{A} \in \mathcal{D} : \mathfrak{A} \models \psi\}$. Its complexity is the **structure complexity** or **data complexity** of the model-checking problem (for ψ on $\mathcal{D}$).

Besides the algorithmic analysis of logic problems, there is another aspect of logic and complexity that has become even more important for finite model theory, and which is really the central programme of descriptive complexity theory. The goal here is to characterize complexity from the point of view of logic (or, more precisely, model theory)[2] by providing, for each important complexity level, logical systems whose expressive power (on finite structures, or on a particular domain of finite structures) coincides precisely with that

[2] There also exist other logical approaches to complexity, based for instance on proof theory. Connections to the finite model theory approach exist, but the flavour is quite different.

complexity level. For a detailed definition, see Sect. 3.2. We shall see that there have been important successes in this programme, but that there also remain difficult problems that are still open.

3.1.2 Model Checking for First-Order Logic

We shall now discuss the problem of evaluating first-order formulae on finite structures using a game-based approach. Model-checking problems, for almost any logic, can be cast as strategy problems for appropriate model-checking games (also called Hintikka games).[3] With any formula $\psi(\overline{x})$, any structure $\mathfrak{A}$ (of the same vocabulary as ψ), and any tuple $\overline{a}$ of elements of $\mathfrak{A}$, we associate a **model-checking game** $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$. It is played by two players, **Verifier** and **Falsifier**. Verifier (sometimes also called Player 0, or $\exists$, or Eloise) tries to prove that $\mathfrak{A} \models \psi(\overline{a})$, whereas Falsifier (also called Player 1, or $\forall$, or Abelard) tries to establish that the formula is false. For first-order logic, the evaluation games are very simple, in the sense that winning conditions are *positional*, and that the games are *well-founded*, i.e. all possible plays are finite (regardless of whether the input structure is finite or infinite). For more powerful logics, notably fixed-point logics, model checking-games may have infinite plays and more complicated winning conditions (see Sect. 3.3.4).

The Game $\mathcal{G}(\mathcal{A}, \psi(\overline{a}))$

Let $\mathfrak{A}$ be a finite structure and let $\psi(\overline{x})$ be a relational first-order formula, which we assume to be in negation normal form, i.e. built up from atoms and negated atoms by means of the propositional connectives $\wedge, \vee$ and the quantifiers $\exists, \forall$. Obviously, any first-order formula can be converted in linear time into an equivalent one in negation normal form. The model-checking game $\mathcal{G}(\mathcal{A}, \psi(\overline{a}))$ has positions (φ, ρ) such that φ is a subformula of ψ, and $\rho : \mathrm{free}(\varphi) \to A$ is an assignment from the free variables of φ to elements of $\mathfrak{A}$. To simplify the notation we usually write $\varphi(\overline{b})$ for a position (φ, ρ) where ρ assigns the tuple $\overline{b}$ to the free variables of φ. The initial position of the game is the formula $\psi(\overline{a})$.

Verifier (Player 0) moves from positions associated with disjunctions and with formulae starting with an existential quantifier. From a position $\varphi \vee \vartheta$, she moves to either φ or ϑ. From a position $\exists y \varphi(\overline{b}, y)$, Verifier can move to any position $\varphi(\overline{b}, c)$, where $c \in A$. Dually, Falsifier (Player 1) makes corresponding moves from conjunctions and universal quantifications. At atoms or negated atoms, i.e. positions $\varphi(\overline{b})$ of the form $b = b'$, $b \neq b'$, $R\overline{b}$, or $\neg R\overline{b}$, the game is over. Verifier has won the play if $\mathfrak{A} \models \varphi(\overline{b})$; otherwise, Falsifier has won.

Model-checking games are a way of defining the semantics of a logic. The equivalence to the standard definition can be proved by a simple induction.

[3] These games should not be confounded with the games used for model comparison (Ehrenfeucht–Fraïssé games) that describe the power of a logic for distinguishing between two structures.

Proposition 3.1.1. *Verifier has a winning strategy for the game $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$ if, and only if, $\mathfrak{A} \models \psi(\overline{a})$.*

This suggests a game-based approach to model checking: given $\mathfrak{A}$ and ψ, construct the game $\mathcal{G}(\mathfrak{A}, \psi)$ and decide whether Verifier has a winning strategy from the initial position. Let us therefore look a little closer at strategy problems for games.

3.1.3 The Strategy Problem for Finite Games

Abstractly, we can describe a two-player game with positional winning conditions by a directed game graph $\mathcal{G} = (V, V_0, V_1, E)$, with a partioning $V = V_0 \cup V_1$ of the nodes into positions where Player 0 moves and positions where Player 1 moves. The possible moves are described by the edge relation $E \subseteq V \times V$. We call w a successor of v if $(v, w) \in E$, and we denote the set of all successors of v by vE. To decribe the winning conditions, we adopt the convention that Player σ loses at positions $v \in V_\sigma$ where no moves are possible. (Alternatively, one could explicitly include in the game description the sets S_0, S_1 of winning terminal positions for each player.)

A **play** of $\mathcal{G}$ is a path $v_0, v_1, \ldots$ formed by the two players starting from a given position v_0. Whenever the current position v_n belongs to V_σ, Player σ chooses a move to a successor $v_{n+1} \in v_nE$; if no move is available, then Player σ has lost the play. If this never occurs, the play goes on infinitely and the winner has to be established by a winning condition on infinite plays. For the moment, let us say that infinite plays are won by neither of the players.[4]

A **strategy** for a player is a function defining a move for each situation in a play where she has to move. Of particular interest are positional strategies, which do not depend on the history of the play, but only on the current position. Hence, a **positional strategy** for Player σ in $\mathcal{G}$ is a (partial) function $f : V_\sigma \to V$ which indicates a choice $(v, f(v)) \in E$ for positions $v \in V_\sigma$. A play $v_0, v_1, \ldots$ is **consistent** with a positional strategy f for Player σ if $v_{n+1} = f(v_n)$ for all $v_n \in V_\sigma$. A strategy for a player is **winning** from position v_0 if she wins every play starting from v_0 that is consistent with that strategy. We say that a strategy is winning on a set W if it is winning from each position in W. The **winning region** W_σ for Player σ is the set of positions from which she has a winning strategy.

A game is **well-founded** if all its plays are finite. Note that a model-checking game $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$ for a first-order formula ψ has a finite game graph if, and only if, $\mathfrak{A}$ is finite, but it is well-founded in all cases. In general, however, games with finite game graphs need not be well-founded.

A game is **determined** if, from each position, one of the players has a winning strategy, i.e. if $W_0 \cup W_1 = V$. Well-founded games are always

[4] We shall later introduce games with more interesting winning conditions for infinite plays.

determined, and so are large classes of more general games (such as games in the Borel hierarchy; see [82, 96]).

We denote by Game the strategy problem for games with finite game graphs and positional winning conditions, i.e.

$$\text{Game} = \{(\mathcal{G}, v) : \text{Player 0 has a winning strategy in } \mathcal{G} \text{ from position } v\}.$$

It is obvious that the Game problem can be solved in polynomial time. Denote by W_σ^n the set of positions from which Player σ has a strategy to win the game in at most n moves. Then $W_\sigma^0 = \{v \in V_{1-\sigma} : vE = \emptyset\}$ is the set of winning terminal positions for Player σ, and we can compute the sets W_σ^n inductively by using

$$W_\sigma^{n+1} := \{v \in V_0 : vE \cap W_\sigma^n \neq \emptyset\} \cup \{v \in V_1 : vE \subseteq W_\sigma^n\}$$

until $W_\sigma^{n+1} = W_\sigma^n$.

To see that Game can actually be solved in *linear time*, a little more work is necessary. The following algorithm is a variant of depth-first search, and computes the entire winning sets for both players in time $O(|V| + |E|)$.

Theorem 3.1.2. *Winning regions of finite games can be computed in linear time.*

Proof. We present an algorithm that computes, for each position, which player, if any, has a winning strategy for the game starting at that position. During the computation three arrays are used:

- win[v] contains either 0 or 1, indicating which player wins, or $\bot$ if we do not know yet, or if none of the players has a winning strategy from v;
- $P[v]$ contains the predecessors of v; and
- $n[v]$ is the number of those successors for which win[v] = $\bot$.

A linear-time algorithm for the Game problem

Input: A game $\mathcal{G} = (V, V_0, V_1, E)$

```
forall  $v \in V$ do                    (* 1: initialization *)
    win[$v$] := $\bot$
    $P[v] := \emptyset$
    $n[v] := 0$
enddo

forall  $(u, v) \in E$ do               (* 2: calculate $P$ and $n$ *)
    $P[v] := P[v] \cup \{u\}$
    $n[u] := n[u] + 1$
enddo
```

```
forall $v \in V_0$                     (* 3: calculate win *)
    if $n[v] = 0$ then  Propagate($v$, 1)
forall $v \in V_1$
    if $n[v] = 0$ then  Propagate($v$, 0)
return win end

procedure Propagate($v$, $\sigma$)
    if win[$v$] $\neq \perp$ then  return
    win[$v$] := $\sigma$                    (* 4: mark $v$ as winning for Player $\sigma$ *)
    forall $u \in P[v]$ do              (* 5: propagate change to predecessors *)
        $n[u] := n[u] - 1$
        if $u \in V_\sigma$ or $n[u] = 0$ then  Propagate($u$, $\sigma$)
    enddo
end
```

The heart of this algorithm is the procedure Propagate(v, σ) which is called any time we have found that Player σ has a winning strategy from position v. Propagate(v, σ) records this fact and investigates whether we are now able to determine the winning player for any of the predecessors of v. This is done by applying the following rules:

- If the predecessor u belongs to Player σ, then this player has a winning strategy from u by moving to position v.
- If the predecessor u belongs to the opponent of Player σ, if win$[u]$ is undefined, and if the winning player has already been determined for all successors w of u, then win$[w] = \sigma$ for all of those successors, and hence Player σ wins from u regardless of the choice of her opponent.

Since parts 4 and 5 of the algorithm are reached only once for each position v, the inner part of the loop in part 5 is executed at most $\sum_v |P[v]| = |E|$ times. Therefore the running time of the algorithm is $O(|V| + |E|)$.

The correctness of the value assigned to win$[v]$ is proved by a straightforward induction on the number of moves in which the corresponding player can ensure that she wins. Note that the positions satisfying $n[v] = 0$ in part 3 are exactly those without outgoing edges even if $n[v]$ is modified by Propagate. □

Game is known to be a PTIME-complete problem (see [57]). This remains the case for **strictly alternating games**, where $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$. Indeed, any game can be transformed into an equivalent strictly alternating one by introducing for each move $(u, v) \in V_\sigma \times V_\sigma$ a new node $e \in V_{1-\sigma}$ and by replacing the move (u, v) by two moves (u, e) and (e, u).

The Game problem (sometimes also called the problem of *alternating reachability*) is a general combinatorial problem that reappears in different guises in many areas. To illustrate this by an example, we shall now show that the satisfiability problem for propositional Horn formulae is essentially the same problem as Game.

Satisfiability for Horn Formulae

It is well known that SAT-HORN, the satisfiability problem for propositional Horn formulae, is

- PTIME-complete [57], and
- solvable in linear time [36, 68].

Using the GAME problem, we can obtain very simple proofs for both results. Indeed, GAME and SAT-HORN are equivalent under log–lin reductions, i.e. reductions that are computable in linear time and logarithmic space. The reductions are so simple that we can say that GAME and SAT-HORN are really the same problem.

Theorem 3.1.3. SAT-HORN *is log–lin equivalent to* GAME.

Proof. GAME $\leq_{\text{log-lin}}$ SAT-HORN. Given a finite game graph $\mathcal{G} = (V, V_0, V_1, E)$, we can construct in time $O(|V| + |E|)$ a propositional Horn formula $\psi_{\mathcal{G}}$ consisting of the clauses $u \leftarrow v$ for all edges $(u, v) \in E$ with $u \in V_0$, and the clauses $u \leftarrow v_1 \wedge \cdots \wedge v_m$ for all nodes $u \in V_1$, where $uE = \{v_1, \ldots, v_m\}$. The minimal model of $\psi_{\mathcal{G}}$ is precisely the winning set W_0 for Player 0. Hence $v \in W_0$ if the Horn formula $\psi_{\mathcal{G}} \wedge (0 \leftarrow v)$ is unsatisfiable.

SAT-HORN $\leq_{\text{log-lin}}$ GAME: Given a Horn formula $\psi(X_1, \ldots, X_n) = \bigwedge_{i \in I} C_i$ with propositional variables $X_1, \ldots, X_n$ and Horn clauses C_i of the form $H_i \leftarrow X_{i_1} \wedge \cdots X_{i_m}$ (where the head of the clause, H_i, is either a propositional variable or the constant 0), we define a game $\mathcal{G}_\psi$ as follows. The positions of Player 0 are the initial position 0 and the propositional variables $X_1, \ldots, X_n$, and the positions of Player 1 are the clauses of ψ. Player 0 can move from a position X to any clause C_i with head X, and Player 1 can move from a clause C_i to any variable occurring in the body of C_i. Formally, $\mathcal{G}_\psi = (V, E)$, $V = V_0 \cup V_1$ with $V_0 = \{0\} \cup \{X_1, \ldots, X_n\}$, $V_1 = \{C_i : i \in I\}$, and

$$E = \{(X, C) \in V_0 \times V_1 : X = \text{head}(C)\} \cup \{(C, X) \in V_1 \times V_0 : X \in \text{body}(C)\}.$$

Player 0 has a winning strategy for $\mathcal{G}_\psi$ from position X if, and only if, $\psi \models X$. In particular, ψ is unsatisfiable if, and only if, Player 0 wins from position 0. □

3.1.4 Complexity of First-Order Model Checking

Roughly, the size of the model-checking game $\mathcal{G}(\mathfrak{A}, \psi)$ is the number of different instantiations of the subformulae of ψ with elements from $\mathfrak{A}$. It is in many cases not efficient to construct the full model-checking game explicitly and then solve the strategy problem, since many positions of the game will not really be needed.

To measure the size of games, and the resulting time and space bounds for the complexity of model checking as precisely as possible, we use, besides the formula length $|\psi|$, the following parameters. The **closure** $\mathrm{cl}(\psi)$ is the set of all subformulae of ψ. Obviously, $|\mathrm{cl}(\psi)| \le |\psi|$, and in some cases $|\mathrm{cl}(\psi)|$ can be much smaller than $|\psi|$. The **quantifier rank** $\mathrm{qr}(\psi)$ is the maximal nesting depth of quantifiers in ψ, and the **width** of ψ is the maximal number of free variables in subformulae, i.e.

$$\mathrm{width}(\psi) = \max\{|\mathrm{free}(\varphi)| : \varphi \in \mathrm{cl}(\psi)\}.$$

Instead of considering the width, one can also rewrite formulae with as few variables as possible.

Lemma 3.1.4. *A first-order formula ψ has width k if, and only if, it is equivalent, via a renaming of bound variables, to a first-order formula with at most k distinct variable symbols.*

Bounded-variable fragments of logics have received a lot of attention in finite model theory. However, here we state the results in terms of formula width rather than number of variables to avoid the necessity to economize on the number of variables. Given the close connection between games and alternating algorithms, it is not surprising that the good estimates for the complexity of model-checking games are often in terms of alternating complexity classes. We now describe an alternating model-checking algorithm for first-order logic that can be viewed as an on-the-fly construction of the model-checking game while playing it.

Theorem 3.1.5. *There is an alternating model-checking algorithm that, given a finite structure $\mathfrak{A}$ and a first-order sentence ψ, decides whether $\mathfrak{A} \models \psi$ in time $O(|\psi| + \mathrm{qr}(\psi)\log|A|)$ and space $O(\log|\psi| + \mathrm{width}(\psi)\log|A|)$ (assuming that atomic statements are evaluated in constant time).*

Proof. We present a recursive alternating procedure **ModelCheck**$(\mathfrak{A}, \rho, \psi)$ that, given a finite structure $\mathfrak{A}$, a first-order formula ψ that may contain free variables, and an assignment $\rho : \mathrm{free}(\psi) \to A$, decides whether $\mathfrak{A} \models \psi[\rho]$.

```
  **ModelCheck**$(\mathfrak{A}, \rho, \psi)$

  **Input:** a first-order formula $\psi$ in negation normal form
          a finite structure $\mathfrak{A}$ (with universe $A$),
          an assignment $\rho : \mathrm{free}(\psi) \to A$
  **if** $\psi$ is an atom or negated atom **then**
          **if** $\mathfrak{A} \models \psi[\rho]$ **accept else reject**
  **if** $\psi = \eta \vee \vartheta$ **then do**
          **guess** $\varphi \in \{\eta, \vartheta\}$, and let $\rho' := \rho|_{\mathrm{free}(\varphi)}$
          **ModelCheck**$(\mathfrak{A}, \rho', \varphi)$
  **if** $\psi = \eta \wedge \vartheta$ **then do**
```

```
        universally choose $\varphi \in \{\eta, \vartheta\}$, and let $\rho' := \rho\,|_{\mathrm{free}(\varphi)}$
        ModelCheck($\mathfrak{A}, \rho', \varphi$)
if $\psi = \exists x\varphi$ then do
        guess an element $a$ of $\mathfrak{A}$
        ModelCheck($\mathfrak{A}, \rho[x \mapsto a], \varphi$)
if $\psi = \forall x\varphi$ then do
        universally choose an element $a$ of $\mathfrak{A}$
        ModelCheck($\mathfrak{A}, \rho[x \mapsto a], \varphi$)
```

A straightforward induction shows that the procedure is correct. The time needed by the procedure is the depth of the syntax tree of ψ plus the time needed to produce the variable assignments. On each computation path, at most $\mathrm{qr}(\psi)$ elements of $\mathfrak{A}$ have to be chosen, and each element needs $\log|A|$ bits. Hence the time complexity is $O(|\psi| + \mathrm{qr}(\psi)\log|A|)$. During the evaluation, the algorithm needs to maintain a pointer to the current position in ψ and to store the current assignment, which needs $\mathrm{free}(\varphi)\log|A|$ bits for the current subformula φ. Hence the space needed by the algorithm is $O(\log|\psi| + \mathrm{width}(\psi)\log|A|)$. □

Theorem 3.1.6. *The model-checking problem for first-order logic is* PSPACE*-complete. For any fixed $k \geq 2$, the model-checking problem for first-order formulae of width at most k is* PTIME*-complete.*

Proof. Membership of these complexity classes follows immediately from Theorem 3.1.5 via the facts that alternating polynomial time coincides with polynomial space and alternating logarithmic space coincides with polynomial time.

Completeness follows by straightforward reductions from known complete problems. QBF, the evaluation problem for quantified Boolean formulae, is PSPACE-complete. It reduces to first-order model checking on the fixed structure (A, P) with $A = \{0, 1\}$ and $P = \{1\}$. Given a quantified Boolean formula ψ without free propositional variables,we can translate it into a first-order sentence ψ as follows: replace every quantification $\exists X_i$ or $\forall X_i$ over a propositional variable X_i by a corresponding first-order quantification $\exists x_i$ or $\forall x_i$ and replace atomic propositions X_i by atoms Px_i. Obviously, ψ evaluates to *true* if, and only if, $(A, P) \models \varphi'$. This proves that the expression complexity and the combined complexity of first-order model checking is PSPACE-complete.

To see that the model-checking problem for first-order formulae of width 2 is PTIME-complete, we reduce to it the Game problem for strictly alternating games, with Player 0 moving first. Given a strictly alternating game graph $\mathcal{G} = (V, V_0, V_1, E)$, we construct formulae $\psi_i(x)$ of width 2, expressing the fact that Player 0 has a winning strategy from $x \in V_0$ in n rounds. Let

$$\begin{aligned}\psi_1(x) &:= \exists y(Exy \land \forall z \neg Eyz)\\ \psi_{i+1}(x) &:= \exists y(Exy \land \forall z(Eyz \rightarrow \psi_i(z)).\end{aligned}$$

Obviously, ψ_n has width 2, and $\mathcal{G} \models \psi_n(v)$ if, and only if, Player 0 can win from position v in at most n rounds. Now, if Player 0 has a winning strategy, then she also has one for winning in at most n rounds, where $n = |V|$, since otherwise the game will be caught in a loop. Hence any instance $\mathcal{G}, v$ of the GAME problem (for strictly alternating games), with $v \in V_0$, can be reduced to the instance $\mathcal{G}, \psi_n(v)$ of the model-checking problem for first-order formulae of width 2. □

Remark. The argument for PTIME-completeness applies also in fact to *propositional modal logic* (ML) [55]. Instead of the formulae $\psi_n(x)$ constructed above, we take the modal formulae

$$\varphi_1 := \Diamond\Box \mathit{false}, \qquad \varphi_{n+1} := \Diamond\Box\varphi_n.$$

Corollary 3.1.7. *The model-checking problem for* ML *is* PTIME-*complete.*

If we consider a fixed formula ψ, Theorem 3.1.5 tells us that the data complexity of first-order logic is much lower than the expression or combined complexity.

Corollary 3.1.8. *Let ψ be a first-order sentence. Then*

$$\{\mathfrak{A} : \mathfrak{A} \ \ \mathit{finite}, \mathfrak{A} \models \psi\} \in \text{ALOGTIME}.$$

In particular, the evaluation problem for any fixed first-order sentence can be computed deterministically in logarithmic space.

3.1.5 Encoding Finite Structures by Words

Complexity theory, at least in its current form, is based on classical computational models, most notably Turing machines, that take as inputs words over a fixed finite alphabet. If we want to measure the complexity of problems on finite structures in terms of these notions, we have to represent structures by words so that they can be used as inputs for, say, Turing machines. This may seem a trivial issue, and for purely algorithmic questions (say for determining the cost of a model-checking algorithm) it indeed often is. However, the programme of finite model theory is to link complexity with logical definability in a deeper way, and for this purpose the represention of structures by words needs careful consideration. It is also at the source of some major unresolved problems that we shall discuss later.

At least implicitly, an encoding of a finite structure by a word requires that we select *an ordered representation of the structure.* To see this, consider the common encoding of a graph $\mathcal{G} = (V, E)$ by its adjacency matrix. Once we have fixed an enumeration of V, say $V = \{v_0, \dots, v_{n-1}\}$, we can represent the graph by the word $w_0 \cdots w_{n^2-1}$, where $w_{in+j} = 1$ if $(v_i, v_j) \in E$ and $w_{in+j} = 0$ otherwise, i.e. row after row of the adjacency matrix. However, this encoding is not canonic. There are $n!$ possibilities of enumerating V, so there may be up to $n!$ different encodings of the same graph by binary strings. But if the graphs come along with a linear order, we do have a canonic way of enumerating the elements and therefore a canonic encoding. Let us now discuss encodings of arbitrary finite structures (of finite vocabulary) by words.

Definition 3.1.9. For any vocabulary τ, we write $\mathrm{Fin}(\tau)$ for the class of finite τ-structures and $\mathrm{Ord}(\tau)$ for the class of all structures $(\mathfrak{A}, <)$, where $\mathfrak{A} \in \mathrm{Fin}(\tau)$ and $<$ is a linear order on A (the universe of $\mathfrak{A}$).

For any structure $(\mathfrak{A}, <) \in \mathrm{Ord}(\tau)$ of cardinality n and for any k, we can identify A^k with the set $\{0, \dots, n^k - 1\}$, by associating each k-tuple with its rank in the lexicographical ordering induced by $<$ on A^k. Ordered structures can be encoded as binary strings in many natural ways. The particular choice of an encoding is not important. We only need the following conditions to be satisfied.

Definition 3.1.10. An encoding $code : \mathrm{Ord}(\tau) \to \Sigma^*$ (over any finite alphabet Σ) is **good** if it identifies isomorphic structures, if its values are polynomially bounded, if it is first-order definable, and if it allows to compute efficiently the values of atomic statements. Formally, this means that the following conditions are satisfied:

(i) $code(\mathfrak{A}, <) = code(\mathfrak{B}, <)$ if and only if $(\mathfrak{A}, <) \cong (\mathfrak{B}, <)$.
(ii) $|code(\mathfrak{A}, <)| \le p(|A|)$ for some polynomial p.
(iii) For all $k \in \mathbb{N}$ and all symbols $\sigma \in \Sigma$, there exists a first-order formula $\beta_\sigma(x_1, \dots, x_k)$ of vocabulary $\tau \cup \{<\}$ such that, for all structures $(\mathfrak{A}, <) \in \mathrm{Ord}(\tau)$ and all $\overline{a} \in A^k$, the following equivalence holds:

$$(\mathfrak{A}, <) \models \beta_\sigma(\overline{a}) \text{ iff the } \overline{a}\text{-th symbol of } code(\mathfrak{A}, <) \text{ is } \sigma.$$

(iv) Given $code(\mathfrak{A}, <)$, a relation symbol R of τ, and (a representation of) a tuple $\overline{a}$, one can efficiently decide whether $\mathfrak{A} \models R\overline{a}$.

The precise meaning of 'efficiently' in clause (iv) depends on the context (e.g. the problem that is studied, the machine model considered, and the level of abstraction at which one is studying a given problem). For the analysis of algorithms, one often assumes that atomic statements are evaluated in constant (or even unit) time on a Random Access Machine (RAM). A minimal requirement is that atoms can be evaluated in linear time and logarithmic space.

A convenient encoding is given as follows. Let $<$ be a linear order on A and let $\mathfrak{A} = (A, R_1, \dots, R_t)$ be a τ-structure of cardinality n. Let ℓ be the maximal arity of $R_1, \dots, R_t$. With each relation R of arity j, we associate a string $\chi(R) = w_0 \cdots w_{n^j-1} 0^{n^\ell - n^j} \in \{0,1\}^{n^\ell}$, where $w_i = 1$ if the ith tuple of A^j belongs to R, and $w_i = 0$ otherwise. Now, we set $code(\mathfrak{A}, <) = 1^n 0^{n^\ell - n} \chi(R_1) \cdots \chi(R_t)$.

Exercise 3.1.11. Prove that this encoding is good. In fact, this encoding lends itself to a very simple logical description in the following sense: if, besides (or instead of) the linear ordering $<$, the corresponding successor relation S and the constants $0, e$ for the first and last elements with respect to $<$ are available, then the encoding is definable by *quantifier-free* formulae $\beta_\sigma(\overline{x})$.

We can fix any good encoding function and understand ordered structures to be represented by their encodings. With an unordered structure $\mathfrak{A}$, we associate the *set* of all encodings $code(\mathfrak{A}, <)$, where $<$ is a linear order on A. So, when we say that an algorithm M decides a class $\mathcal{K}$ of τ-structures, we actually mean that M decides the set of encodings of structures in $\mathcal{K}$, i.e. the language

$$code(\mathcal{K}) := \{code(\mathfrak{A}, <) : \mathfrak{A} \in \mathcal{K} \text{ and } < \text{ is a linear order on } A\}.$$

It thus makes sense to ask whether such a $\mathcal{K}$ belongs to a complexity class, such as P or NP. In particular, we can ask how complicated it is to decide the class of models of a logical sentence.

Word Structures

We have seen how classes of structures are encoded by languages. On the other hand, any language $L \subseteq \Gamma^*$ can also be considered as a class of structures over the vocabulary $\{<\} \cup \{P_a : a \in \Gamma\}$. Indeed, a word $w = w_0 \dots w_{m-1} \in \Gamma^*$ is described by the structure $\mathfrak{B}(w)$ with universe $\{0, \dots, m-1\}$, with the usual interpretation of $<$ and where $P_a = \{i : w_i = a\}$.

Isomorphism Invariance

We have seen that encoding an unordered structure involves selecting an ordering on the universe. In general, different orderings produce different encodings. However, we want to consider properties of structures, not of their encodings, An algorithm that decides whether a structure has a certain property gets encodings $code(\mathfrak{A}, <)$ as inputs and should produce the same answer (yes or no) for all encodings of the same structure. That is, the outcome of the algorithm should not depend on the particular ordered representation of the structure, but only on its isomorphism type. In other words the algorithm should be isomorphism-invariant. For most of the algorithms considered here isomorphism invariance is obvious, but in general it is an undecidable property.

Exercise 3.1.12. A first-order sentence ψ of vocabulary $\tau \cup \{<\}$ is *order-invariant* on a class $\mathcal{K}$ of τ-structures if its truth on any structure in $\mathcal{K}$ does not depend on the choice of the linear ordering $<$. That is, for any $\mathfrak{A} \in \mathcal{K}$ and any pair $<, <'$ of linear orderings on $\mathfrak{A}$ we have that $(\mathfrak{A}, <) \models \psi \iff (\mathfrak{A}, <') \models \psi$. Prove that it is undecidable whether a given first-order formula is order-invariant on finite structures. *Hint:* use Trakhtenbrot's Theorem. A first-order sentence ψ, in which $<$ and Q do not occur, has a finite model with at least two elements if, and only if, $\psi \rightarrow \forall x \exists y(x < y \vee Qx)$ is not order-invariant.

3.2 Capturing Complexity Classes

We have already mentioned that the research programme of descriptive complexity theory links complexity with logic in a deeper way than a complexity analysis of model-checking algorithms can do. We are looking for results saying that, on a certain domain $\mathcal{D}$ of structures, a logic L (such as first-order logic, least fixed-point logic, or a fragment of second-order logic) *captures* a complexity class *Comp*. This means that (1) for every fixed sentence $\psi \in L$, the data complexity of evaluating ψ on structures from $\mathcal{D}$ is a problem in the complexity class *Comp*, and (2) every property of structures in $\mathcal{D}$ that can be decided with complexity *Comp* is definable in the logic L.

Two important examples of such results are Fagin's Theorem, which says that existential second-order logic captures NP on the class of all finite structures, and the Immerman–Vardi Theorem, which says that least fixed-point logic captures PTIME on the class of all ordered finite structures. On *ordered* finite structures, logical characterizations of this kind are known for all major complexity classes. On the other hand, it is not known, and it is one of the major open problems in the area, whether PTIME can be captured by any logic if no ordering is present.

In Sect. 3.2.1, we prove Fagin's Theorem and relate it it to the spectrum problem, which is a classical problem in mathematical logic. In Sect. 3.2.2, we make precise the notion of a logic capturing a complexity class on a domain of finite structures. We then show in Sect. 3.2.3 that on ordered structures, second-order Horn logic captures polynomial time. In Sects. 3.2.4 and 3.2.5, we discuss logics that capture logarithmic space complexity classes.

3.2.1 Capturing NP: Fagin's Theorem

The **spectrum** of a first-order sentence ψ is the set of cardinalities of its finite models, i.e.

$$\text{spectrum}(\psi) := \{k \in \mathbb{N} : \psi \text{ has a model with } k \text{ elements}\}.$$

As early as 1952, Scholz [93] posed the problem of characterizing the class of spectra, i.e. the subsets $S \subseteq \mathbb{N}$ for which there exists a first-order sentence ψ such that $\text{spectrum}(\psi) = S$. A more specific problem is the **complementation problem for spectra**, posed by Asser [7], who asked whether the complement of each spectrum is also a spectrum.

Note that the spectrum of a first-order sentence ψ of relational vocabulary $\tau = \{R_1, \dots, R_m\}$ can be viewed as the set of finite models of the existential second-order sentence $\exists R_1 \cdots \exists R_m \psi$. Since all relation symbols are quantified, this is a sentence over the empty vocabulary, i.e. its models are just sets. Thus there is a one-to-one correspondence between the spectra of first-order sentences and the classes of finite models of existential second-order sentences *over the empty vocabulary.* If we allow different vocabularies for existential second-order sentences, this naturally leads to the notion of a generalized spectrum [43].

Definition 3.2.1. Existential second-order logic, sometimes denoted by Σ_1^1, is the set of formulae of the form $\exists R_1 \cdots \exists R_m \varphi$, where $m \in \mathbb{N}$, $R_1, \dots, R_m$ are relation symbols of any finite arity, and φ is a first-order formula. A **generalized spectrum** is the class of finite models of a sentence in existential second-order logic.

Example 3.2.2. The class of bipartite graphs is a generalized spectrum. It is defined by the sentence

$$\exists R \forall x \forall y (Exy \rightarrow (Rx \leftrightarrow \neg Ry)).$$

Exercise 3.2.3. Prove that the class of Hamiltonian graphs, the class of k-colourable graphs (for any fixed k), and the class of graphs that admit a perfect matching are generalized spectra. (A perfect matching in an undirected graph $G = (V, E)$ is a set $M \subseteq E$ of edges such that every node belongs to precisely one edge of M.)

Theorem 3.2.4 (Fagin). *Let $\mathcal{K}$ be an isomorphism-closed class of finite structures of some fixed non-empty finite vocabulary. Then $\mathcal{K}$ is in* NP *if and only if $\mathcal{K}$ is definable by an existential second-order sentence, i.e. if and only if $\mathcal{K}$ is a generalized spectrum.*

Proof. First, we show how to decide a generalized spectrum. Let $\psi := \exists R_1 \cdots \exists R_m \varphi$ be an existential second-order sentence. We shall describe a non-deterministic polynomial-time algorithm M which, given an encoding $code(\mathfrak{A}, <)$ of a structure $\mathfrak{A}$, decides whether $\mathfrak{A} \models \psi$. First, M non-deterministically guesses relations $R_1, \dots, R_m$ on A. A relation R_i is determined by a binary string of length n^{r_i}, where r_i is the arity of R_i and $n = |A|$. Then M decides whether $(\mathfrak{A}, R_1, \dots, R_m) \models \varphi$. Since φ is first-order, this can be done in logarithmic space and therefore in polynomial time.

Hence the computation of M consists of guessing a polynomial number of bits, followed by a deterministic polynomial-time computation. Obviously, M decides the class of finite models of ψ.

Conversely, let $\mathcal{K}$ be an isomorphism-closed class of τ-structures and let M be a non-deterministic one-tape Turing machine which, given an input $code(\mathfrak{A}, <)$, decides in polynomial time whether $\mathfrak{A}$ belongs to $\mathcal{K}$. We shall construct an existential second-order sentence φ whose finite models are precisely the structures in $\mathcal{K}$. The construction given here is not quite the standard one. It is optimized so that it can be easily adapted to other situations, in particular for giving a capturing result for PTIME (see Section 3.2.3).

Let $M = (Q, \Sigma, q_0, F^+, F^-, \delta)$, where Q is the set of states, Σ is the alphabet of M, q_0 is the initial state, F^+ and F^- are the set of accepting and rejecting states, and $\delta : (Q \times \Sigma) \to \mathcal{P}(Q \times \Sigma \times \{-1, 0, 1\})$ is the transition function. Without loss of generality, we can assume that all computations of M for an input $code(\mathfrak{A}, <)$ reach an accepting or rejecting state after at most $n^k - 1$ steps (where n is the cardinality of $\mathfrak{A}$).

We represent a computation of M for an input $code(\mathfrak{A}, <)$ by a tuple $\overline{X}$ of relations on A, and we shall construct a first-order sentence ψ_M of vocabulary $\tau \cup \{<\} \cup \{\overline{X}\}$ such that

$$(\mathfrak{A}, < \overline{X}) \models \psi_M \iff \text{the relations } \overline{X} \text{ represent an accepting computation of } M \text{ on } code(\mathfrak{A}, <).$$

To represent the n^k time and space parameters of the computation we identify numbers up to $n^k - 1$ with tuples in A^k. Given a linear order, the associated successor relation and the least and greatest element are of course definable. Note, further, that if a successor relation S and constants $0, e$ for the first and last elements are available, then the induced successor relation $\overline{y} = \overline{x} + 1$ on k-tuples is definable by a quantifier-free formula

$$\bigvee_{i<k} \Big(\bigwedge_{j<i} (x_j = e \wedge y_j = 0) \wedge S x_i y_i \wedge \bigwedge_{j>i} x_j = y_j \Big).$$

Hence, for any fixed integer m, the relation $\overline{y} = \overline{x} + m$ is also expressible.

The description $\overline{X}$ of a computation of M on $code(\mathfrak{A}, <)$ consists of the following relations.

(1) For each state $q \in Q$, the predicate

$$X_q := \{\overline{t} \in A^k : \text{at time } \overline{t},\ M \text{ is in state } q\}.$$

(2) For each symbol $\sigma \in \Sigma$, the predicate

$$Y_\sigma := \{(\overline{t}, \overline{a}) \in A^k \times A^k : \text{at time } \overline{t}, \text{ cell } \overline{a} \text{ contains the symbol } \sigma\}.$$

(3) The head predicate

$$Z := \{(\overline{t}, \overline{a}) \in A^k \times A^k : \text{at time } \overline{t}, \text{ the head of } M \text{ is on position } \overline{a}\}.$$

The sentence ψ_M is the universal closure of the conjunction

$$\text{START} \wedge \text{COMPUTE} \wedge \text{END}.$$

The subformula START enforces the condition that the configuration of M at time $t = 0$ is $C_0(\mathfrak{A}, <)$, the input configuration on $code(\mathfrak{A}, <)$. Recall that a good encoding is represented by first-order formulae $\beta_\sigma(\overline{x})$ (condition (iii) of the definition of good encodings). We set

$$\text{START} := X_{q_0}(\overline{0}) \wedge Z(\overline{0},\overline{0}) \wedge \bigwedge_{\sigma\in\Sigma} \big(\beta_\sigma(\overline{x}) \rightarrow Y_\sigma(\overline{0},\overline{x})\big).$$

The subformula COMPUTE describes the transitions from one configuration to the next. It is the conjunction of the formulae

$$\text{NOCHANGE} := \bigwedge_{\sigma\in\Sigma} \Big(Y_\sigma(\overline{t},\overline{x}) \wedge (\overline{y} \neq \overline{x}) \wedge (\overline{t}' = \overline{t}+1) \wedge Z(\overline{t},\overline{y}) \rightarrow Y_\sigma(\overline{t}',\overline{x})\Big)$$

and

$$\text{CHANGE} := \bigwedge_{\substack{q\in Q\\ \sigma\in\Sigma}} \Big(\text{PRE}[q,\sigma] \rightarrow \bigvee_{(q',\sigma',m)\in\delta(q,\sigma)} \text{POST}[q',\sigma',m]\Big)$$

where

$$\begin{aligned} \text{PRE}[q,\sigma] &:= X_q(\overline{t}) \wedge Z(\overline{t},\overline{x}) \wedge Y_\sigma(\overline{t},\overline{x}) \wedge \overline{t}' = \overline{t}+1 \\ \text{POST}[q',\sigma',m] &:= X_{q'}(\overline{t}') \wedge Y_{\sigma'}(\overline{t}',\overline{x} \wedge \exists\overline{y}(\overline{x}+m = \overline{y} \wedge Z(\overline{t}',\overline{y})). \end{aligned}$$

NOCHANGE expresses the fact that the contents of tape cells that are not currently being scanned do not change from one configuration to the next, whereas CHANGE enforces the changes in the relations X_q, Y_σ, and Z imposed by the transition function.

Finally, we have the formula

$$\text{END} := \bigwedge_{q\in F^-} \neg X_q(\overline{t}),$$

which enforces acceptance by forbidding rejection.

Claim 1. *If M accepts code$(\mathfrak{A},<)$, then $(\mathfrak{A},<) \models (\exists\overline{X})\psi_M$.*

This follows immediately from the construction of ψ_M, since for any accepting computation of M on *code*$(\mathfrak{A},<)$ the intended meaning of $\overline{X}$ satisfies ψ_M.

Claim 2. *If $(\mathfrak{A},<\overline{X}) \models \psi_M$, then M accepts code$(\mathfrak{A},<)$.*

Suppose that $(\mathfrak{A},<\overline{X}) \models \psi_M$. For any M-configuration C with state q, head position p, and tape content $w_0\cdots w_{n^k-1} \in \Sigma^*$, and for any time $j < n^k$, let CONF$[C,j]$ be the conjunction of the atomic statements that hold for C at time j, i.e.

$$\text{CONF}[C,j] := X_q(\overline{j}) \wedge Z(\overline{j},\overline{p}) \wedge \bigwedge_{i=0}^{n^k-1} Y_{w_i}(\overline{j},\overline{i})$$

where $\overline{j},\overline{p}$ and $\overline{i}$ are the tuples in A^k representing the numbers j,p, and i.

(a) Let C_0 be the input configuration of M for input $code(\mathfrak{A}, <)$. Since $(\mathfrak{A}, <, \overline{X}) \models \mathrm{START}$, it follows that

$$(\mathfrak{A}, <, \overline{X}) \models \mathrm{CONF}[C_0, 0].$$

(b) Owing to the subformula COMPUTE of ψ_M, we have, for all non-final configurations C and all $j < n^k - 1$, that

$$\psi_M \wedge \mathrm{CONF}[C, j] \models \bigvee_{C' \in \mathrm{Next}(C)} \mathrm{CONF}[C', j+1],$$

where $\mathrm{Next}(C) = \{C' : C \vdash_M C'\}$ is the set of successor configurations of C. It follows that there exists a computation

$$C_0(\mathfrak{A}, <) = C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_{n^k-1} = C_{\mathrm{end}}$$

of M on $code(\mathfrak{A}, <)$ such that, for all $j < n^k$,

$$(\mathfrak{A}, <, \overline{X}) \models \mathrm{CONF}[C_j, j].$$

(c) Since $(\mathfrak{A}, <, \overline{X}) \models \mathrm{END}$, the configuration C_{end} is not rejecting. Thus, M accepts $code(\mathfrak{A}, <)$.

This proves Claim 2. Clearly, one can axiomatize linear orders in first-order logic. Hence

$$\mathfrak{A} \in \mathcal{K} \quad \text{iff} \quad \mathfrak{A} \models (\exists <)(\exists \overline{X})(\text{“}< \text{ is a linear order”} \wedge \psi_M).$$

This proves that $\mathcal{K}$ is a generalized spectrum. □

Exercise 3.2.5. Prove that every set in NP can be defined by a Σ^1_1-sentence whose first-order part has an $\forall^*\exists^*$-prefix. Furthermore, prove that this cannot be reduced to $\forall^*$. Finally, prove that it can be reduced to $\forall^*$ if

(a) existential second-order quantification over function symbols is allowed, or
(b) if we consider only ordered structures with an explicitly given successor relation and constants 0, e for the first and last elements.

There are several interesting consequences of Fagin's Theorem. First of all, the NP-completeness of SAT (the satisfiability problem for propositional logic) is an easy corollary of Fagin's Theorem.

Theorem 3.2.6 (Cook and Levin). SAT *is* NP-*complete.*

Proof. It is obvious that SAT is an NP-problem. It remains to show that any problem $\mathcal{K}$ in NP can be reduced to SAT. Since, as explained above, words can be viewed as special kinds of finite structures, we can assume that $\mathcal{K} \subseteq \mathrm{Fin}(\tau)$ for some finite vocabulary τ. By Fagin's Theorem, there exists a first-order sentence ψ such that

$$\mathcal{K} = \{\mathfrak{A} \in \mathrm{Fin}(\tau) : \mathfrak{A} \models \exists R_1 \cdots \exists R_m \psi\}.$$

We now present a logspace reduction that associates with every input structure $\mathfrak{A} \in \mathrm{Fin}(\tau)$ a propositional formula $\psi_{\mathfrak{A}}$. Given $\mathfrak{A}$, replace in ψ

- all subformulae $\exists x_i \varphi$ by $\bigvee_{a_i \in A} \varphi[x_i/a_i]$,
- all subformulae $\forall x_i \varphi$ by $\bigwedge_{a_i \in A} \varphi[x_i/a_i]$, and
- all τ-atoms $P\overline{a}$ by their truth values in $\mathfrak{A}$.

Since the τ-atoms can be evaluated efficiently, this translation is computable efficiently. Viewing the atoms $R_i\overline{a}$ as propositional variables, we have obtained a propositional formula $\psi_{\mathfrak{A}}$ such that

$$\mathfrak{A} \in \mathcal{K} \qquad \iff \qquad \mathfrak{A} \models \exists R_1 \cdots \exists R_m \psi \qquad \iff \qquad \psi_{\mathfrak{A}} \in \mathrm{SAT}.$$

□

Fagin's Theorem is readily extended to the higher levels of the polynomial-time hierarchy , and thus to a correspondance between second-order logic and the polynomial-time hierarchy.

Corollary 3.2.7. *Let $\mathcal{K}$ be an isomorphism-closed class of finite structures of some fixed non-empty vocabulary τ. Then code($\mathcal{K}$) is in the polynomial-time hierarchy* PH *if and only if there exists a second-order sentence ψ such that $\mathcal{K}$ is the class of finite models of ψ.*

In the statement of Fagin's Theorem, we required the vocabulary to be non-empty. The case of the empty vocabulary, i.e. spectra, is different, because the natural way of specifying a finite set is to write down its size n in binary, and so the length of the encoding is logarithmic in n, whereas encodings of structures of non-empty vocabularies have polynomial length. The formula constructed in the proof of Fagin's Theorem talks about computations that are polynomial in n, and hence, in the case of spectra, exponential in the length of the input. As a consequence, Fagin's characterization of generalized spectra in terms of NP implies a characterization of spectra in terms of NEXPTIME. This has also been established in a different way in [71].

Corollary 3.2.8 (Jones and Selman). *A set $S \subseteq \mathbb{N}$ is a spectrum if and only if $S \in$* NEXPTIME.

Hence the complementation problem for spectra is really a complexity-theoretic problem: spectra are closed under complementation if, and only if, NEXPTIME = Co-NEXPTIME.

Exercise 3.2.9. Prove that a set $S \subseteq \mathbb{N}$ is in EXPTIME if and only if it is a *categorical spectrum*, i.e. the spectrum of a first-order sentence that has, up to isomorphism, at most one model in any finite cardinality.

3.2.2 Logics That Capture Complexity Classes

Fagin's Theorem gives a precise correspondence between a logic and a complexity class: a property of finite structures is decidable in non-deterministic polynomial time exactly when it is definable in existential second-order logic. The same is true for the correspondence between the polynomial-time hierarchy and SO, as given by Corollary 3.2.7.

Note that the results on the model-checking complexity of first-order logic do *not* give such precise correspondences. We know by Theorem 3.1.5 and Corollary 3.1.8 that whenever a property of finite structures is first-order definable, it is decidable in LOGSPACE and in fact even in ALOGTIME. But we do not have a result giving the converse, and in fact the converse is *false*. There are computationally very simple properties of finite structures that are not first-order definable; one of them is the property of having an even number of elements.

Hence the natural question arises of whether complexity classes other than NP and the polynomial-time hierarchy can also be precisely captured by logics. For most of the popular complexity classes, notably PTIME, we do not know whether this is possible on the domain of all finite structures. But we have a lot of interesting capturing results if we do not consider arbitrary finite structures, but certain specific domains. In particular we have close correspondences between logic and complexity for the domain of *ordered finite* structures.

By a **model class** we always mean a class $\mathcal{K}$ of structures of a fixed vocabulary τ that is closed under isomorphism, i.e. if $\mathfrak{A} \in \mathcal{K}$ and $\mathfrak{A} \cong \mathfrak{B}$, then also $\mathfrak{B} \in \mathcal{K}$. We speak of a **domain** of structures instead, if the vocabulary is not fixed. For a domain $\mathcal{D}$ and vocabulary τ, we write $\mathcal{D}(\tau)$ for the class of τ-structures in $\mathcal{D}$.

Intuitively, a logic L captures a complexity class *Comp* on $\mathcal{D}$ if the L-definable properties of structures in $\mathcal{D}$ are precisely those that are decidable in *Comp*. Here is a more detailed definition.

Definition 3.2.10. Let L be a logic, *Comp* a complexity class, and $\mathcal{D}$ a domain of finite structures. We say that L **captures** *Comp* **on** $\mathcal{D}$ if

(1) For every vocabulary τ and every sentence $\psi \in L(\tau)$, the model-checking problem for ψ on $\mathcal{D}(\tau)$ is in the complexity class *Comp*.
(2) For every model class $\mathcal{K} \subseteq \mathcal{D}(\tau)$ whose membership problem is in *Comp*, there exists a sentence $\psi \in L(\tau)$ such that

$$\mathcal{K} = \{\mathfrak{A} \in \mathcal{D}(\tau) : \mathfrak{A} \models \psi\}.$$

By Fagin's Theorem, the logic Σ_1^1 captures NP on the domain of all finite structures, and by Corollary 3.2.7, second-order logic captures the polynomial-time hierarchy.

We sometimes simply write $L \subseteq$ *Comp* to say that condition (1) of Definition 3.2.10 is satisfied for L and *Comp* on the domain of *all* finite structures.

A classical result, from the 'prehistory' of finite model theory, says that a language is regular (i.e. recognizable by a finite automaton) if, and only if, it is definable in monadic second-order logic (MSO). As words can be viewed as a special domain of structures, this is a capturing result in the sense of Definition 3.2.10.

Theorem 3.2.11 (Büchi, Elgot, and Trakhtenbrot). *On the domain of word structures, monadic second-order logic captures the regular languages.*

There are numerous extensions and ramifications of this theorem, most of them established in the context of automata theory. We refer to [95, 97] for a proof and further results. However, the emphasis of most of the work in finite model theory is on structures more complicated structures than words, and concerns complexity levels higher than the regular languages.

3.2.3 Capturing Polynomial Time on Ordered Structures

In this section, we present a logical characterization of polynomial time on ordered structures, in terms of second-order Horn logic. Other such characterizations will follow in subsequent sections.

Definition 3.2.12. Second-order Horn logic, denoted by SO-HORN, is the set of second-order sentences of the form

$$Q_1 R_1 \cdots Q_m R_m \forall y_1 \cdots \forall y_s \bigwedge_{i=1}^{t} C_i$$

where $Q_i \in \{\exists, \forall\}$, the R_i are relation symbols, and the C_i are *Horn clauses with respect to* $R_1, \ldots, R_m$. More precisely, each C_i is an implication of the form

$$H \leftarrow \beta_1 \wedge \cdots \wedge \beta_m$$

where each β_j is either a positive atom $R_k\overline{z}$, or a first-order formula that does not contain $R_1, \ldots, R_m$. The conjunction $\beta_1 \wedge \cdots \wedge \beta_m$ is called the **body** of the clause. H, the **head** of the clause, is either an atom $R_j\overline{z}$ or the Boolean constant 0 (for *false*).

Thus the first-order parts of the sentences in SO-HORN are universal Horn sentences with respect to the quantified predicates $R_1, \ldots, R_m$, but may use arbitrary first-order information about the 'input predicates' from the underlying vocabulary. Σ_1^1-HORN denotes the existential fragment of SO-HORN, i.e. the set of SO-HORN sentences where all second-order quantifiers are existential.

Example 3.2.13. The problem GEN is a well-known P-complete problem [57, 70]. It may be presented as the set of structures (A, S, f, a) in the vocabulary of one unary predicate S, one binary function f, and a constant a, such that

a is contained in the closure of S under f. Clearly, the complement of GEN is also P-complete. It is defined by the following sentence of Σ^1_1-HORN:

$$\exists R \forall y \forall z \Big((Ry \leftarrow Sy) \wedge (Rfyz \leftarrow Ry \wedge Rz) \wedge (0 \leftarrow Ra) \Big).$$

Example 3.2.14. The circuit value problem (CVP) is also P-complete [57], even when restricted to circuits with a fan-in of 2 over NAND gates. Such a circuit can be considered as a structure (V, E, I^+, I^-, out), where (V, E) is a directed acyclic graph, I^+ and I^- are monadic predicates, and a is a constant. Here Exy means that node x is one of the two input nodes for y; I^+ and I^- contain the input nodes with values 1 and 0, respectively; and *out* stands for the output node.

We shall take for granted that E is a connected, acyclic graph with a fan-in of 2, sources $I^+ \cup I^-$, and sink *out*. The formula $\exists T \exists F \forall x \forall y \forall z \varphi$, where φ is the conjunction of the clauses

$$\begin{aligned}
Tx &\leftarrow I^+x \\
Fx &\leftarrow I^-x \\
Ty &\leftarrow Fx \wedge Exy \\
Fz &\leftarrow Tx \wedge Exz \wedge Ty \wedge Eyz \wedge y \neq z \\
0 &\leftarrow Tx \wedge Fx \\
Tx &\leftarrow x = out
\end{aligned}$$

then states that the circuit (V, E, I^+, I^-, out) evaluates to 1.

Exercise 3.2.15. To justify the definition of SO-HORN, show that the admission of quantifiers over functions, or of first-order prefixes of a more general form, would make the restriction to Horn clauses pointless. Any such extension of SO-HORN has the full power of second-order logic.

Theorem 3.2.16. *Every sentence $\psi \in$ SO-HORN is equivalent to some sentence $\psi' \in \Sigma^1_1$-HORN.*

Proof. It suffices to prove the theorem for formulae of the form

$$\psi := \forall P \exists R_1 \cdots \exists R_m \forall \overline{z} \varphi,$$

where φ is a conjunction of Horn clauses. An arbitrary formula in SO-HORN may then be brought to existential form by successively removing the innermost universal second-order quantifier. We first prove the following claim.

Claim. *A formula $\exists \overline{R} \forall \overline{z} \varphi(P, \overline{R}) \in \Sigma^1_1$-HORN is true for all predicates P (on a given structure $\mathfrak{A}$) if it holds for those predicates P that are false at at most one point.*

Let k be the arity of P. For every k-tuple $\overline{a}$, let $P^{\overline{a}} = A^k - \{\overline{a}\}$, i.e. the predicate that is false at $\overline{a}$ and true at all other points. By assumption, there exist predicates $\overline{R}^{\overline{a}}$ such that

$$(\mathfrak{A}, P^{\overline{a}}, \overline{R}^{\overline{a}}) \models \forall \overline{z}\varphi.$$

Now, take any predicate $P \neq A^k$, and let $R_i := \bigcap_{\overline{a} \notin P} R_i^{\overline{a}}$. We claim that $(\mathfrak{A}, P, \overline{R}) \models \forall \overline{z}\varphi$.

Suppose that this is false; there then exists a relation $P \neq A^k$, a clause C of φ, and an assignment $\rho : \{z_1 \dots, z_s\} \to A$ such that $(\mathfrak{A}, P, \overline{R}) \models \neg C[\rho]$. We now show that there then exists a tuple $\overline{a}$ such that also $(\mathfrak{A}, P^{\overline{a}}, \overline{R}^{\overline{a}}) \models \neg C[\rho]$.

If the head of $C[\rho]$ is $P\overline{u}$, then take $\overline{a} = \overline{u} \notin P$. If the head of $C[\rho]$ is $R_i\overline{u}$, then choose some $\overline{a} \notin P$ such that $\overline{u} \notin R_i^{\overline{a}}$; such an $\overline{a}$ must exist because $\overline{u} \notin R_i$. Finally, if the head is 0, take an arbitrary $\overline{a} \notin P$. The head of $C[\rho]$ is clearly false in $(\mathfrak{A}, P^{\overline{a}}, \overline{R}^{\overline{a}})$. The atom $P\overline{a}$ does not occur in the body of $C[\rho]$, because $\overline{a} \notin P$ and all atoms in the body of $C[\rho]$ are true in $(\mathfrak{A}, P, \overline{R})$; all other atoms of the form $P\overline{v}$ that might occur in the body of the clause remain true for $P^{\overline{a}}$ also. Moreover, every atom $R_i\overline{v}$ in the body remains true if R_i is replaced by $R_i^{\overline{a}}$ (because $R_i \subseteq R_i^{\overline{a}}$). This implies that the clause $(\mathfrak{A}, P^{\overline{a}}, \overline{R}^{\overline{a}}) \models \neg C[\rho]$, and thus

$$(\mathfrak{A}, P^{\overline{a}}, \overline{R}^{\overline{a}}) \models \neg\forall \overline{z}\varphi,$$

which contradicts our assumption.

Thus the claim has been established. This implies that the original formula ψ is equivalent to the conjunction

$$\exists \overline{R} \forall \overline{z} \varphi_0 \wedge \forall \overline{y} (\exists \overline{R}) \forall \overline{z} \varphi_1,$$

where φ_1 and φ_0 are obtained from φ by replacing every atom $P\overline{u}$ by $\overline{u} \neq \overline{y}$ (which is true iff $\overline{u} \in P^{\overline{y}}$), or by $(\overline{u} = \overline{u})$ (which is always true), respectively. It is easy to transform this conjunction into an equivalent formula in Σ_1^1-HORN. $\square$

Theorem 3.2.17. *If $\psi \in$ SO-HORN, then the set of finite models of ψ is in PTIME.*

Proof. We can restrict our attention to sentences $\psi = \exists R_1 \cdots \exists R_m \forall \overline{z} \bigwedge_i C_i$ in Σ_1^1-HORN. Given any finite structure $\mathfrak{A}$ of appropriate vocabulary, we reduce the problem of whether $\mathfrak{A} \models \psi$ to the satisfiability problem for a propositional Horn formula by the same technique as in the proof of Theorem 3.2.6.

Replace the universal quantifiers $\forall z_i$ by conjunctions over the elements $a_i \in A$ and omit the quantifier prefix. Then substitute in the body of each clause the first-order formulae that do not involve $R_1, \dots, R_m$ by their truth values in $\mathfrak{A}$. If there is any clause that is already made false by this partial interpretation (i.e. the head is false and all atoms in the body are true), then reject ψ. Otherwise, omit all clauses that are already made true (i.e. the head is true or a conjunct of the body is false) and delete the conjuncts already interpreted from the remaining clauses. Consider the atoms $R_i\overline{u}$ as propositional variables. The resulting formula is a propositional Horn formula whose length is polynomially bounded in the cardinality of $\mathfrak{A}$ and which is satisfiable if and only if $\mathfrak{A} \models \psi$. The satisfiability problem for propositional Horn formulae can be solved in linear time. $\square$

Theorem 3.2.18 (Grädel). *On ordered structures,* SO-HORN *and* Σ^1_1-HORN *capture* PTIME.

Proof. This follows from an analysis of our proof of Fagin's Theorem. If the Turing machine M happens to be deterministic, then the sentence $\exists \overline{X} \psi_M$ constructed in that proof can easily be transformed to an equivalent sentence in Σ^1_1-HORN.

To see this, recall that ψ_M is the universal closure of START $\wedge$ NOCHANGE $\wedge$ CHANGE $\wedge$ END. The formulae START, NOCHANGE, and END are already in Horn form. The formula CHANGE has the form

$$\bigwedge_{\substack{q \in Q \\ \sigma \in \Sigma}} \Big(\mathrm{PRE}[q,\sigma] \rightarrow \bigvee_{(q',\sigma',m) \in \delta(q,\sigma)} \mathrm{POST}[q',\sigma',m] \Big),$$

where

$$\begin{aligned} \mathrm{PRE}[q,\sigma] &:= X_q(\overline{t}) \wedge Z(\overline{t},\overline{x}) \wedge Y_\sigma(\overline{t},\overline{x}) \wedge \overline{t}' = \overline{t} + 1 \\ \mathrm{POST}[q',\sigma',m] &:= X_{q'}(\overline{t}') \wedge Y_{\sigma'}(\overline{t}',\overline{x}) \wedge \exists \overline{y}(\overline{x} + m = \overline{y} \wedge Z(\overline{t}',\overline{y})). \end{aligned}$$

For a deterministic M, we have for each pair (q,σ) a unique value $\delta(q,\sigma) = (q',\sigma',m)$. In this case, the implication $\mathrm{PRE}[q,\sigma] \rightarrow \mathrm{POST}[q',\sigma',m]$ can be replaced by the conjunction of the Horn clauses

$$\begin{gathered} \mathrm{PRE}[q,\sigma] \rightarrow X_{q'}(\overline{t}') \\ \mathrm{PRE}[q,\sigma] \rightarrow Y_{\sigma'}(\overline{t}',\overline{x}) \\ \mathrm{PRE}[q,\sigma] \wedge \overline{y} = \overline{x} + m \rightarrow Z(\overline{t}',\overline{y}). \end{gathered}$$

$\square$

Exercise 3.2.19. Prove that, contrary to the case of Fagin's Theorem, the assumption that a linear order is explicitly available cannot be eliminated, since linear orderings are not axiomatizable by Horn formulae.

Exercise 3.2.20. In [47], where the results of this section were proved, a weaker variant of SO-HORN was used, in which the body may not contain arbitrary first-order formulae of the input vocabulary, but only atoms and negated input atoms. Prove that the two variants of SO-HORN are equivalent on ordered structures with a successor relation and with constants for the first and last elements, but not on ordered structures without a successor relation. *Hint*: sentences in the weak variant of SO-HORN are preserved under substructures, i.e. if $\mathfrak{A} \models \psi$ and $\mathfrak{B} \subseteq \mathfrak{A}$, then also $\mathfrak{B} \models \psi$.

3.2.4 Capturing Logarithmic Space Complexity

In this section and the next, we describe two approaches to defining logics that capture logarithmic space complexity classes on ordered structures. The first approach is based on restrictions of second-order logic, similarly to the definition of SO-HORN, whereas the second technique adds transitive closure operators to first-order logic.

Definition 3.2.21. Second-order Krom logic, denoted by SO-KROM, is the set of second-order formulae

$$Q_1R_1 \cdots Q_mR_m \forall y_1 \cdots \forall y_s \bigwedge_{i=1}^{t} C_i$$

where every clause C_i is a disjunction of at most two literals of the form $(\neg)R_i\overline{y}$ and of a first-order formula that does not contain $R_1, \ldots, R_m$. Such formulae are Krom (i.e. in 2-CNF) with respect to the quantified predicates. Σ_1^1-KROM is the existential fragment of SO-KROM. The intersection of Σ_1^1-HORN and Σ_1^1-KROM is denoted by Σ_1^1-KROM-HORN.

Example 3.2.22. The reachability problem ('Is there a path in the graph (V, E) from a to b?') is complete for NLOGSPACE via first-order translations. Its complement is expressible by a formula from Σ_1^1-KROM-HORN,

$$\exists T \forall x \forall y \forall z \Big(Txx \wedge (Txz \leftarrow Txy \wedge Eyz) \wedge (0 \leftarrow Tab) \Big).$$

As in the case of SO-HORN, it is also known that every sentence of SO-KROM is equivalent to a sentence of Σ_1^1-KROM (see [47]).

Proposition 3.2.23. *For every sentence $\psi \in$ SO-KROM, the set of finite models of ψ is in* NLOGSPACE.

The proof is analogous to the proof of Theorem 3.2.17. It uses the fact that 2-SAT, the satisfiability problem for propositional Krom formulae, is in NLOGSPACE. On ordered structures, SO-KROM captures NLOGSPACE. We shall indicate the general idea of the proof here. Suppose that M is an $O(\log n)$-space-bounded non-deterministic Turing machine with an input tape carrying a representation $code(\mathfrak{A}, <)$ of an input structure, and one or more separate work tapes. A *reduced configuration* of M reflects the control state of M, the content of the work tapes, and the positions of the heads on the input tape and the work tapes. Thus a configuration is specified by a reduced configuration together with the input. Given that reduced configurations of M for the input $code(\mathfrak{A}, <)$ have a logarithmic length with respect to $|A|$, we can represent them by tuples $\overline{c} = c_1, \ldots, c_r \in A^r$ for fixed r. The *initial* reduced configuration on any input $code(\mathfrak{A}, <)$ is represented by the tuple $\overline{0}$. Assume that M has a single accepting state, say state 1, and let the first component of the reduced configuration describe the state. The condition that $\overline{y}$ represents an *accepting* configuration is then expressed by $\mathrm{ACCEPT}(\overline{y}) := (y_1 = 1)$. Further, it is not difficult (although it is somewhat lengthy) to write down a quantifier-free formula $\mathrm{NEXT}(\overline{x}, \overline{y})$ such that, for every successor structure $(\mathfrak{A}, S, 0, e)$ and every tuple $\overline{c}$ representing a reduced configuration,

$$(\mathfrak{A}, S, 0, e) \models \mathrm{NEXT}(\overline{c}, \overline{d})$$

if, and only if, $\overline{d}$ represents a reduced successor configuration of $\overline{c}$ for the input $(\mathfrak{A}, <)$. Taking the disjunctive normal form $\mathrm{NEXT}(\overline{x}, \overline{y}) = \bigvee_i \mathrm{NEXT}_i(\overline{x}, \overline{y})$, we can express the staement that M does *not* accept the input $code(\mathfrak{A}, <)$ by the sentence

$$\begin{aligned}\psi_M := &\exists R \forall \overline{x} \forall \overline{y} \big(R\overline{0} \wedge \bigwedge_i (R\overline{y} \leftarrow R\overline{x} \wedge \mathrm{NEXT}_i(\overline{x}, \overline{y})) \\ &\wedge (\Box \leftarrow R\overline{y} \wedge \mathrm{ACCEPT}(\overline{y})\big).\end{aligned}$$

This proves that, on ordered structures, the complement of every problem in NLOGSPACE is definable in SO-KROM. Since NLOGSPACE is closed under complements, and since the formula ψ_M is in fact in Σ^1_1-KROM-HORN, we have proved the following result.

Theorem 3.2.24 (Grädel). *On ordered structures, the logics* SO-KROM*,* Σ^1_1-KROM*, and* Σ^1_1-KROM-HORN *capture* NLOGSPACE*.*

Remark. The characterizations of P and NLOGSPACE by second-order Horn and Krom logics can also be reformulated in terms of generalized spectra. The notion of a generalized spectrum can be appropriately modified to the notions of a *generalized Horn spectrum* and a *generalized Krom spectrum.* Let a *model class* be any isomorphism-closed class of structures of some fixed finite signature. Fagin's Theorem and Theorems 3.2.18 and 3.2.24 can then be summarized as follows:

- A model class of finite structures is NP iff it is a generalized spectrum.
- A model class of ordered structures is in P iff it is a generalized Horn spectrum.
- A model class of ordered structures is in NLOGSPACE iff it is a generalized Krom spectrum.

3.2.5 Transitive Closure Logics

One of the limitations of first-order logic is the lack of a mechanism for unbounded iteration or recursion. This has motivated the study of more powerful languages that add recursion in one way or another to first-order logic. A simple but important example of a query that is not first-order expressible is reachability. By adding transitive closure operators to FO, we obtain a natural family of logics with a recursion mechanism.

Definition 3.2.25. Transitive closure logic, denoted by TC, is obtained by augmenting the syntax of first order logic by the following rule for building formulae:

Let $\varphi(\overline{x},\overline{y})$ be a formula with variables $\overline{x}=x_1,\dots,x_k$ and $\overline{y}=y_1,\dots,y_k$, and let $\overline{u}$ and $\overline{v}$ be two k-tuples of terms. Then

$$[\mathbf{tc}_{\overline{x},\overline{y}}\ \varphi(\overline{x},\overline{y})](\overline{u},\overline{v})$$

is a formula which says that the pair $(\overline{u},\overline{v})$ is contained in the transitive closure of the binary relation on k-tuples that is defined by φ. In other words, $\mathfrak{A} \models [\mathbf{tc}_{\overline{x},\overline{y}}\ \varphi(\overline{x},\overline{y})](\overline{a},\overline{b})$ if, and only if, there exist an $n \geq 1$ and tuples $\overline{c}_0,\dots,\overline{c}_n$ in A^k such that $\overline{c}_0=\overline{a}$, $\overline{c}_n=\overline{b}$, and $\mathfrak{A}\models\varphi(\overline{c}_i,\overline{c}_{i+1})$, for all $i<n$.

Of course, it is understood that φ can contain free variables other than $\overline{x}$ and $\overline{y}$; these will also be free in the new formula. Moreover, transitive closure logic is closed under the usual first-order operations. We can thus build Boolean combinations of TC-formulae, we can nest TC-operators, etc.

Example 3.2.26. A directed graph $G=(V,E)$ is acyclic if, and only if, $G \models \forall z[\mathbf{tc}_{x,y}Exy](z,z)$. It is well known that a graph is bipartite (2-colourable) if, and only if, it does not contain a cycle of odd length. This is expressed by the TC-formula $\forall x\forall y([\mathbf{tc}_{x,y}x\neq y\wedge\exists zExz\wedge Ezy](x,y)\rightarrow\neg Eyx)$.

Exercise 3.2.27. Show that, for every $\psi\in$ TC, the set of finite models of ψ is decidable in NLOGSPACE.

The same idea as in the proof of Theorem 3.2.24 shows that, on ordered structures, TC captures NLOGSPACE. The condition that an $O(\log n)$-space-bounded Turing machine M accepts $code(\mathfrak{A},<)$ is expressed by the formula

$$\exists\overline{z}\big(\mathrm{ACCEPT}(\overline{z})\wedge[\mathbf{tc}_{\overline{x},\overline{y}}\ \mathrm{NEXT}(\overline{x},\overline{y})](\overline{0},\overline{z})\big).$$

Theorem 3.2.28 (Immerman). *On ordered structures,* TC *captures* NLOGSPACE.

An interesting variant of TC is **deterministic transitive closure logic**, denoted DTC, which makes definable the transitive closure of any *deterministic* definable relation. The syntax of DTC is analogous to TC, allowing us to build formulae of the form $[\mathbf{dtc}_{\overline{x},\overline{y}}\ \varphi(\overline{x},\overline{y})](\overline{u},\overline{v})$, for any formula $\varphi(\overline{x},\overline{y})$. The semantics can be defined by the equivalence

$$[\mathbf{dtc}_{\overline{x},\overline{y}}\ \varphi(\overline{x},\overline{y})](\overline{u},\overline{v})\equiv[\mathbf{tc}_{\overline{x},\overline{y}}\ \varphi(\overline{x},\overline{y})\wedge\forall\overline{z}(\varphi(\overline{x},\overline{z})\rightarrow\overline{y}=\overline{z})](\overline{u},\overline{v}).$$

It is clear that transitive closures of deterministic relations can be checked by deterministic Turing machines using only logarithmic space. Conversely, acceptance by such machines amounts to deciding a reachability problem ('is there an accepting configuration that is reachable from the input configuration?') with respect to the successor relation $\vdash_M$ on configurations. Of course, for deterministic Turing machines, $\vdash_M$ is deterministic. We already know that on ordered structures, $\vdash_M$ is first-order definable, and hence acceptance can be defined in DTC.

Theorem 3.2.29 (Immerman). *On ordered finite structures* DTC *captures* LOGSPACE.

In particular, separating DTC from TC on ordered finite structures would amount to separating the complexity classes LOGSPACE and NLOGSPACE. However, on the domain of arbitrary finite structures, we can actually separate these logics [51].

Given a graph $G = (V, E)$, let $2G$ be the graph with vertex set $V \times \{0,1\}$ and edges $\langle(u,i),(v,j)\rangle$ for $(u,v) \in E, i,j \in \{0,1\}$. It is easy to see that on the class of all 'double graphs' $2G$, DTC collapses to FO. Take any tuple $\bar{u} = (u_1,i_1),\dots,(u_k,i_k)$ of vertices in a double graph $2G$, and let the closure of $\bar{u}$ be the set $\{u_1,\dots,u_k\} \times \{0,1\}$. Switching the second component of any node is an automorphism of $2G$, and hence no definable deterministic path from $\overline{u}$ can leave the closure of $\overline{u}$. That is, if $2G \models [\mathbf{dtc}_{\overline{x},\overline{y}}\varphi(\overline{x},\overline{y})](\overline{u},\overline{v})$, then each node of $\overline{v}$ belongs to the closure of $\overline{u}$. Therefore DTC-definable paths are of bounded length, and can thus be defined by first-order formulae. On the other hand the usual argument (based on Ehrenfeucht–Fraïssé games) showing that transitive closures are not first-order definable applies also to the class of double graphs. Hence DTC is strictly less powerful than TC on double graphs. In [51] other graph classes are identified on which TC is more expressive than DTC. An interesting example is the class of all hypercubes.

Theorem 3.2.30. *On finite graphs,* DTC $\subsetneq$ TC.

TC is a much richer and more complicated logic than DTC also in other respects. For instance, DTC has a positive normal form: formulae $\neg[\mathbf{dtc}_{\overline{x}\overline{y}}\varphi(\overline{x},\overline{y})](\overline{u},\overline{v})$ can be rewritten using the **dtc** operator only positively. On the other hand, the alternation hierarchy in TC is strict [52].

3.3 Fixed-Point Logics

One of the distinguishing features of finite model theory compared with other branches of logic is the eminent role of various kinds of fixed-point logics. Fixed-point logics extend a basic logical formalism (such as first-order logic, conjunctive queries, or propositional modal logic) by a constructor for forming *fixed points of relational operators*.

What do we mean by a **relational operator**? Note that any formula $\psi(R,\overline{x})$ of vocabulary $\tau \cup \{R\}$ can be viewed as defining, for every τ-structure $\mathfrak{A}$, an update operator $F_\psi : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ on the class of k-ary relations on A, namely

$$F_\psi : R \mapsto \{\overline{a} : (\mathfrak{A}, R) \models \psi(R, \overline{a})\}.$$

A fixed point of F_ψ is a relation R for which $F_\psi(R) = R$. In general, a fixed point of F_ψ need not exist, or there may exist many of them. However, if R happens to occur only positively in ψ, then the operator F_ψ is monotone, and in that case there exists a *least* relation $R \subseteq A^k$ such that $F_\psi(R) = R$. The most influential fixed-point formalisms in logic are concerned with least (and greatest) fixed points, so we shall discuss these first. In finite model theory, a number of other fixed-point logics are important as well, and the structure, expressive power, and algorithmic properties of these logics have been studied intensively. We shall discuss them later.

3.3.1 Some Fixed-Point Theory

There is a well-developed mathematical theory of fixed points of monotone operators on complete lattices. A **complete lattice** is a partial order $(A, \leq)$ such that each set $X \subseteq A$ has a supremum (a least upper bound) and an infimum (a greatest lower bound). Here we are interested mainly in power set lattices $(\mathcal{P}(A^k), \subseteq)$ (where A is the universe of a structure), and later in product lattices $(\mathcal{P}(B_1) \times \cdots \times \mathcal{P}(B_m), \subseteq)$. For simplicity, we shall describe the basic facts of fixed-point theory for lattices $(\mathcal{P}(B), \subseteq)$, where B is an arbitrary (finite or infinite) set.

Definition 3.3.1. Let $F : \mathcal{P}(B) \to \mathcal{P}(B)$ be a function.

(1) $X \subseteq B$ is a **fixed point** of F if $F(X) = X$.
(2) A **least fixed point** or a **greatest fixed point** of F is a fixed point X of F such that $X \subseteq Y$ or $Y \subseteq X$, respectively, for each fixed point Y of F.
(3) F is **monotone**, if $X \subseteq Y \Longrightarrow F(X) \subseteq F(Y)$ for all $X, Y \subseteq B$.

Theorem 3.3.2 (Knaster and Tarski). *Every monotone operator $F : \mathcal{P}(B) \to \mathcal{P}(B)$ has a least fixed point* $\mathbf{lfp}(F)$ *and a greatest fixed point* $\mathbf{gfp}(F)$. *Further, these fixed points may be written in the form*

$$\begin{aligned}\mathbf{lfp}(F) &= \bigcap\{X : F(X) = X\} = \bigcap\{X : F(X) \subseteq X\}\\ \mathbf{gfp}(F) &= \bigcup\{X : F(X) = X\} = \bigcup\{X : F(X) \supseteq X\}.\end{aligned}$$

Proof. Let $S = \{X \subseteq B : F(X) \subseteq X\}$ and $Y = \bigcap S$. We first show that Y is a fixed point of F.

$F(Y) \subseteq Y$. Clearly, $Y \subseteq X$ for all $X \in S$. As F is monotone, it follows that $F(Y) \subseteq F(X) \subseteq X$. Hence $F(Y) \subseteq \bigcap S = Y$.

$Y \subseteq F(Y)$. As $F(Y) \subseteq Y$, we have $F(F(Y)) \subseteq F(Y)$, and hence $F(Y) \in S$. Thus $Y = \bigcap S \subseteq F(Y)$.

By definition, Y is contained in all X such that $F(X) \subseteq X$. In particular Y is contained in all fixed points of F. Hence Y is the least fixed point of F.

The argument for the greatest fixed point is analogous. □

Least fixed points can also be constructed inductively. We call an operator $F : \mathcal{P}(B) \to \mathcal{P}(B)$ **inductive** if the sequence of its **stages** X^α (where α is an ordinal), defined by

$$\begin{aligned} X^0 &:= \emptyset, \\ X^{\alpha+1} &:= F(X^\alpha), \text{ and} \\ X^\lambda &:= \bigcup_{\alpha<\lambda} X^\alpha \text{ for limit ordinals } \lambda, \end{aligned}$$

is increasing, i.e. if $X^\beta \subseteq X^\alpha$ for all $\beta < \alpha$. Obviously, monotone operators are inductive. The sequence of stages of an inductive operator eventually reaches a fixed point, which we denote by X^∞. The least ordinal β for which $X^\beta = X^{\beta+1} = X^\infty$ is called $\mathrm{cl}(F)$, the **closure ordinal** of F.

Lemma 3.3.3. *For every inductive operator $F : \mathcal{P}(B) \to \mathcal{P}(B)$, $|\mathrm{cl}(F)| \leq |B|$.*

Proof. Let $|B|^+$ denote the smallest cardinal greater than $|B|$. Suppose that the claim is false for F. Then for each $\alpha < |B|^+$ there exists an element $x_\alpha \in X^{\alpha+1} - X^\alpha$. The set $\{x_\alpha : \alpha < |B|^+\}$ is a subset of B of cardinality $|B|^+ > |B|$, which is impossible. □

Proposition 3.3.4. *For monotone operators, the inductively constructed fixed point coincides with the least fixed point, i.e. $X^\infty = \mathbf{lfp}(F)$.*

Proof. As X^∞ is a fixed point, $\mathbf{lfp}(X) \subseteq X^\infty$. For the converse, we show by induction that $X^\alpha \subseteq \mathbf{lfp}(F)$ for all α. As $\mathbf{lfp}(F) = \bigcap\{Z : F(Z) \subseteq Z\}$, it suffices to show that X^α is contained in all Z for which $F(Z) \subseteq Z$.

For $\alpha = 0$, this is trivial. By monotonicity and the induction hypothesis, we have $X^{\alpha+1} = F(X^\alpha) \subseteq F(Z) \subseteq Z$. For limit ordinals λ with $X^\alpha \subseteq Z$ for all $\alpha < \lambda$ we also have $X^\lambda = \bigcup_{\alpha<\lambda} \subseteq Z$. □

The greatest fixed point can be constructed by a dual induction, starting with $Y^0 = B$, by setting $Y^{\alpha+1} := F(Y^\alpha)$ and $Y^\lambda = \bigcap_{\alpha<\lambda} Y^\alpha$ for limit ordinals. The *decreasing* sequence of these stages then eventually converges to the greatest fixed point $Y^\infty = \mathbf{gfp}(F)$.

The least and greatest fixed points are dual to each other. For every monotone operator F, the dual operator $F^d : X \mapsto \overline{F(\overline{X})}$ (where $\overline{X}$ denotes the complement of X) is also monotone, and we have that

$$\mathbf{lfp}(F) = \overline{\mathbf{gfp}(F^d)} \text{ and } \mathbf{gfp}(F) = \overline{\mathbf{lfp}(F^d)}.$$

Exercise 3.3.5. Prove this.

Everything said so far holds for operators on arbitrary (finite or infinite) power set lattices. In *finite model theory*, we consider operators $F : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ for finite A only. In this case the inductive constructions will reach the least or greatest fixed point in a polynomial number of steps. As a consequence, these fixed points can be constructed efficiently.

Lemma 3.3.6. *Let $F : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ be a monotone operator on a finite set A. If F is computable in polynomial time (with respect to $|A|$), then so are the fixed points* $\mathbf{lfp}(F)$ *and* $\mathbf{gfp}(F)$.

3.3.2 Least Fixed-Point Logic

LFP is the logic obtained by adding least and greatest fixed points to first-order logic.

Definition 3.3.7. *Least fixed-point logic* (LFP) is defined by adding to the syntax of first-order logic the following *least fixed-point formation rule*: If $\psi(R, \overline{x})$ is a formula of vocabulary $\tau \cup \{R\}$ with only positive occurrences of R, if $\overline{x}$ is a tuple of variables, and if $\overline{t}$ is a tuple of terms (such that the lengths of $\overline{x}$ and $\overline{t}$ match the arity of R), then

$$[\mathbf{lfp} R\overline{x} \,.\, \psi](\overline{t}) \text{ and } [\mathbf{gfp} R\overline{x} \,.\, \psi](\overline{t})$$

are formulae of vocabulary τ. The free first-order variables of these formulae are those in $(\text{free}(\psi) - \{x : x \text{ in } \overline{x}\}) \cup \text{free}(\overline{t})$.

Semantics. For any τ-structure $\mathfrak{A}$ providing interpetations for all free variables in the formula, we have that $\mathfrak{A} \models [\mathbf{lfp} R\overline{x} \,.\, \psi](\overline{t})$ if $\overline{t}^{\mathfrak{A}}$ (the tuple of elements of $\mathfrak{A}$ interpreting $\overline{t}$) is contained in $\mathbf{lfp}(F_\psi)$, where F_ψ is the update operator defined by ψ on $\mathfrak{A}$. Similarly for greatest fixed points.

Example 3.3.8. Here is a fixed-point formula that defines the transitive closure of the binary predicate E:

$$\text{TC}(u, v) := [\mathbf{lfp} Txy \,.\, Exy \lor \exists z(Exz \land Tzy)](u, v).$$

Note that in a formula $[\mathbf{lfp} R\overline{x} \,.\, \varphi](\overline{t})$, there may be free variables in φ additional to those in $\overline{x}$, and these remain free in the fixed-point formula. They are often called **parameters** of the fixed-point formula. For instance, the transitive closure can also be defined by the formula

$$\varphi(u, v) := [\mathbf{lfp} Ty \,.\, Euy \lor \exists x(Tx \land Exy)](v)$$

which has u as a parameter.

Exercise 3.3.9. Show that every LFP-formula is equivalent to one without parameters (at the cost of increasing the arity of the fixed-point variables).

Example 3.3.10. Let $\varphi := \forall y(y < x \to Ry)$ and let $(A, <)$ be a partial order. The formula $[\mathbf{lfp} Rx \,.\, \varphi](x)$ then defines the well-founded part of $<$. The closure ordinal of F_φ on $(A, <)$ is the length of the longest well-founded initial segment of $<$, and $(A, <) \models \forall x[\mathbf{lfp} Rx \,.\, \varphi](x)$ if, and only if, $(A, <)$ is well-founded.

Exercise 3.3.11. Prove that the LFP-sentence

$$\psi := \forall y \exists z Fyz \land \forall y [\mathbf{lfp} Ry \,.\, \forall x (Fxy \rightarrow Rx)](y)$$

is an infinity axiom, i.e. it is satisfiable but does not have a finite model.

Example 3.3.12. The GAME query asks, given a finite game $\mathcal{G} = (V, V_0, V_1, E)$, to compute the set of winning positions for Player 0 (see Section 3.1.3). The GAME query is LFP-definable, by use of $[\mathbf{lfp} Wx \,.\, \varphi](x)$ with

$$\varphi(W, x) := (V_0 x \land \exists y (Exy \land Wy)) \lor (V_1 \land \forall y (Exy \rightarrow Wy)).$$

The GAME query plays an important role for LFP. It can be shown that every LFP-definable property of finite structures can be reduced to GAME by a quantifier-free translation [31]. Hence GAME is complete for LFP via this notion of reduction, and thus a natural candidate if one is trying to separate a weaker logic from LFP.

Exercise 3.3.13. Prove that the problem GEN and the circuit value problem (see Examples 3.2.13 and 3.2.14) are expressible in LFP.

The duality between the least and greatest fixed points implies that for any formula ψ,

$$[\mathbf{gfp} R\overline{x} \,.\, \psi](\overline{t}) \equiv \neg[\mathbf{lfp} R\overline{x} \,.\, \neg\psi[R/\neg R]](\overline{t}),$$

where $\psi[R/\neg R]$ is the formula obtained from ψ by replacing all occurrences of R-atoms by their negations. (As R occurs only positively in ψ, the same is true for $\neg\psi[R/\neg R]$.) Because of this duality, greatest fixed points are often omitted in the definition of LFP. On the other hand, it is sometimes convenient to keep the greatest fixed points, and to use the duality (and de Morgan's laws) to translate LFP-formulae to *negation normal form*, i.e. to push negations all the way to the atoms.

Capturing Polynomial Time

From the fact that first-order operations are polynomial-time computable and from Lemma 3.3.6, we can immediately conclude that every LFP-definable property of finite strucures is computable in polynomial time.

Proposition 3.3.14. *Let ψ be a sentence in* LFP. *It is decidable in polynomial time whether a given finite structure $\mathfrak{A}$ is a model of ψ. In short,* LFP $\subseteq$ PTIME.

Obviously LFP, is a fragment of second-order logic. Indeed, by the Tarski–Knaster Theorem,

$$[\mathbf{lfp} R\overline{x} \,.\, \psi(R, \overline{x})](\overline{y}) \equiv \forall R((\forall \overline{x}(\psi(R, \overline{x}) \rightarrow R\overline{x})) \rightarrow R\overline{y}).$$

We next relate LFP to SO-HORN.

Theorem 3.3.15. *Every formula $\psi \in$ SO-HORN is equivalent to some formula $\psi^* \in$ LFP.*

Proof. By Theorem 3.2.16, we can assume that $\psi = (\exists R_1) \cdots (\exists R_m)\varphi \in \Sigma^1_1$-HORN. By combining the predicates $R_1, \dots, R_m$ into a single predicate R of larger arity and by renaming variables, it is easy to transform ψ into an equivalent formula

$$\psi' := \exists R \forall \overline{x} \forall \overline{y} \bigwedge_i C_i \wedge \bigwedge_j D_j,$$

where the C_i are clauses of the form $R\overline{x} \leftarrow \alpha_i(R, \overline{x}, \overline{y})$ (with exactly the same head $R\overline{x}$ for every i) and the D_j are clauses of the form $0 \leftarrow \beta_j(R, \overline{x}, \overline{y})$. The clauses C_i define, on every structure $\mathfrak{A}$, a monotone operator $F : R \mapsto \{\overline{x} : \bigvee_i \exists \overline{y} \alpha_i(\overline{x}, \overline{y})\}$. Let R^ω be the least fixed point of this operator. Obviously $\mathfrak{A} \models \neg\psi$ if and only if $\mathfrak{A} \models \beta_i(R^\omega, \overline{a}, \overline{b})$ for some i and some tuple $\overline{a}$, $\overline{b}$. But R^ω is defined by the fixed-point formula

$$\alpha^\omega(\overline{x}) := [\mathbf{lfp} R\overline{x} \,.\, \bigvee_i \exists \overline{y} \alpha_i(\overline{x}, \overline{y})](\overline{x}).$$

Hence, for $\beta := \exists \overline{x} \exists \overline{y} \bigvee_j \beta_j(\overline{x}, \overline{y})$, ψ is equivalent to the formula $\psi^* := \neg\beta[R\overline{z}/\alpha^\omega(\overline{z})]$ obtained from $\neg\beta$ by substituting all occurrences of atoms $R\overline{z}$ by $\alpha^\omega(\overline{z})$. Clearly, this formula is in LFP. □

Hence SO-HORN $\leq$ LFP $\leq$ SO. As an immediate consequence of Theorems 3.2.18 and 3.3.15 we obain the Immerman–Vardi Theorem.

Theorem 3.3.16 (Immerman and Vardi). *On ordered structures, least fixed-point logic captures polynomial time.*

However, on unordered structures, SO-HORN is strictly weaker than LFP.

3.3.3 The Modal μ-Calculus

A fragment of LFP that is of fundamental importance in many areas of computer science (e.g. controller synthesis, hardware verification, and knowledge representation) is the modal μ-calculus (L_μ). It is obtained by adding least and greatest fixed points to propositional modal logic (ML). In other words L_μ relates to ML in the same way as LFP relates to FO.

Modal logics such as ML and the μ-calculus are evaluated on transition systems (alias Kripke structures, alias coloured graphs) at a particular node. Given a formula ψ and a transition system G, we write $G, v \models \psi$ to denote that G holds at node v of G. Recall that formulae of ML, for reasoning about **transition systems** $G = (V, (E_a)_{a \in A}, (P_b)_{b \in B})$, are built from atomic propositions P_b by means of the usual propositional connectives and the modal operators $\langle a \rangle$ and $[a]$. That is, if ψ is a formula and $a \in A$ is an action, then we can build the formulae $\langle a \rangle \psi$ and $[a]\psi$, with the following semantics:

$$G, v \models \langle a \rangle \psi \text{ iff } G, w \models \psi \text{ for } \textit{some } w \text{ such that } (v, w) \in E_a,$$
$$G, v \models [a]\psi \text{ iff } G, w \models \psi \text{ for } \textit{all } w \text{ such that } (v, w) \in E_a.$$

If there is only one transition relation, i.e. $A = \{a\}$, then we simply write $\Box$ and $\Diamond$ for $[a]$ and $\langle a \rangle$, respectively.

ML can be viewed as an extension of propositional logic. However, in our context it is more convenient to view it as a simple fragment of first-order logic. A modal formula ψ defines a query on transition systems, associating with G a set of nodes $\psi^G := \{v : G, v \models \psi\}$, and this set can be defined equivalently by a first-order formula $\psi^*(x)$. This translation maps atomic propositions P_b to atoms $P_b x$, it commutes with the Boolean connectives, and it translates the modal operators by use of quantifiers as follows:

$$(\langle a \rangle \psi)^*(x) := \exists y(E_a xy \wedge \psi^*(y))$$
$$([a]\psi)^*(x) := \forall y(E_a xy \rightarrow \psi^*(y)).$$

Note that the resulting formula has width 2 and can thus be written with only two variables. We have proved the following proposition.

Proposition 3.3.17. *For every formula* $\psi \in$ ML*, there exists a first-order formula* $\psi^*(x)$ *of width 2, which is equivalent to* ψ *in the sense that* $G, v \models \psi$ *iff* $G \models \psi^*(v)$.

The *modal fragment* of first-order logic is the image of propositional modal logic under this translation. It has turned out that the modal fragment has interesting algorithmic and model-theoretic properties (see [3] and the references given there).

Definition 3.3.18. The **modal μ-calculus** L_μ extends ML (including propositional variables $X, Y, \ldots$, which can be be viewed as monadic second-order variables) by the following rule for building fixed point formulae: If ψ is a formula in L_μ and X is a propositional variable that only occurs positively in ψ, then $\mu X.\psi$ and $\nu X.\psi$ are also L_μ-formulae.

The semantics of these fixed-point formulae is completely analogous to that for LFP. The formula ψ defines on G (with universe V, and with interpretations for other free second-order variables that ψ may have besides X) the monotone operator $F_\psi : \mathcal{P}(V) \rightarrow \mathcal{P}(V)$ assigning to every set $X \subseteq V$ the set $\psi^G(X) := \{v \in V : (G, X), v \models \psi\}$. Now,

$$G, v \models \mu X.\psi \text{ iff } v \in \mathbf{lfp}(F_\psi)$$
$$G, v \models \nu X.\psi \text{ iff } v \in \mathbf{gfp}(F_\psi).$$

Example 3.3.19. The formula $\mu X.\varphi \vee \langle a \rangle X$ asserts that there exists a path along a-transitions to a node where φ holds.

The formula $\psi := \nu X.\left(\bigvee_{a \in A} \langle a \rangle \textit{true} \wedge \bigwedge_{a \in A} [a]X\right)$ expresses the assertion that the given transition system is deadlock-free. In other words, $G, v \models \psi$

if no path from v in G reaches a dead end (i.e. a node without outgoing transitions).

Finally, the formula $\nu X.\mu Y.\langle a\rangle((\varphi \wedge X) \vee Y)$ says that there exists a path from the current node on which φ holds infinitely often.

Exercise 3.3.20. Prove that the formulae in Example 3.3.19 do indeed express the stated properties.

The translation from ML into FO is readily extended to a translation from L_μ into LFP.

Proposition 3.3.21. *Every formula $\psi \in L_\mu$ is equivalent to a formula $\psi^*(x) \in \text{LFP}$.*

Proof. By induction. A formula of form $\mu X.\varphi$ is translated to $[\mathbf{lfp} Xx\,.\,\varphi^*](x)$, and similarly for greatest fixed points. $\square$

Further the argument proving that LFP can be embedded into SO also shows that L_μ is a fragment of MSO.

Let us turn to algorithmic issues. The complexity of the model-checking problem for L_μ is a major open problem, as far as combined complexity and expression complexity are concerned (see Section 3.3.5). However, the data complexity can be settled easily.

Proposition 3.3.22 (data complexity of L_μ). *Fix any formula $\psi \in L_\mu$. Given a finite transition system G and a node v, it can be decided in polynomial time whether $G, v \models \psi$. Further, there exist $\psi \in L_\mu$ for which the model checking problem is* PTIME*-complete.*

Proof. As L_μ is a fragment of LFP, the first claim is obvious. For the second claim, recall that the GAME problem for strictly alternating games is PTIME-complete (see Section 3.1.2). Player 0 has a winning strategy from position $v \in V_0$ in the game $G = (V, V_0, V_1, E)$ if, and only if, $G, v \models \mu X.\Diamond\Box X$. $\square$

Despite this result, it is not difficult to see that the μ-calculus does not suffice to capture PTIME, even in very restricted scenarios such as word structures. Indeed, as L_μ is a fragment of MSO, it can only define *regular languages*, and of course, not all PTIME-languages are regular. However, we shall see in Section 3.5.3 that there is a multidimensional variant of L_μ that captures the *bisimulation-invariant* fragment of PTIME.

For more information on the μ-calculus, we refer to [5, 21, 56] and the references therein.

3.3.4 Parity Games

For least fixed-point logics, the appropriate evaluation games are *parity games.* These are games of possibly infinite duration where each position is assigned a natural number, called its priority, and the winner of an infinite play is determined according to whether the least priority seen infinitely often during the play is even or odd. It is open whether winning sets and winning strategies for parity games can be computed in polynomial time. The best algorithms known today are polynomial in the size of the game, but exponential with respect to the number of priorities. Practically competitive model-checking algorithms for the modal μ-calculus work by solving the strategy problem for the associated parity game (see e.g. [73]).

Definition 3.3.23. We describe a **parity game** by a labelled graph $\mathcal{G} = (V, V_0, V_1, E, \Omega)$, where (V, V_0, V_1, E) is a game graph as in Section 3.1.2, and $\Omega : V \to \mathbb{N}$ assigns to each position a **priority**. The set V of positions may be finite or infinite, but the number of different priorities must be finite; it is called the **index** of $\mathcal{G}$. Recall that a finite play of a game is lost by the player who gets stuck, i.e. cannot move. The difference to the games of Section 3.1.2 is that we have different winning conditions for infinite plays $v_0 v_1 v_2 \dots$. If the smallest number appearing infinitely often in the sequence $\Omega(v_0)\Omega(v_1)\dots$ of priorities is even, then Player 0 wins the play; otherwise, Player 1 wins.

Recall that a **positional strategy** of Player σ is a partial function $f : V_\sigma \to V$ with $(v, f(v)) \in E$. A strategy f is said to be winning on a set of positions $W \subseteq V$ if any play that starts at a position in W and is consistent with f is winning for Player σ. Further, W_σ, the **winning region** of Player σ, is the set of positions from which Player σ has a winning strategy (which, a priori, need not be positional).

Exercise 3.3.24. (**Combination of positional strategies).** Let f and f' be positional strategies for Player σ that are winning on the sets W and W', respectively. Let $f \lhd f'$ be the positional strategy defined by

$$(f \lhd f')(x) := \begin{cases} f(x) & \text{if } x \in W \\ f'(x) & \text{otherwise.} \end{cases}$$

Prove that $f \lhd f'$ is winning on $W \cup W'$.

The Positional Determinacy Theorem for parity games states that parity games are always determined (i.e., from each position, one of the players has a winning strategy) and in fact, positional strategies always suffice. This was proved independently by Emerson and Jutla [40] and by Mostowski [86]. Earlier, Gurevich and Harrington [62] had proved that Muller games (which are more general than parity games) are determined via finite-memory strategies.

Theorem 3.3.25 (Positional Determinacy). *In any parity game, the set of positions can be partitioned into two sets W_0 and W_1 such that Player 0 has a positional strategy that is winning on W_0 and Player 1 has a positional strategy that is winning on W_1.*

Here, we only prove this theorem for the case of finite game graphs. The presentation is inspired by a similar proof due to Ehrenfeucht and Mycielski [39] for mean payoff games; see also [12]. For the general case, we refer the reader to [102] or [97].

Proof. Let $\mathcal{G} = (V, V_0, V_1, E, \Omega)$ be a parity game with a finite set V of positions. We call a position $v \in V$ *live* if it is non-terminal (i.e. if there is at least one possible move from v). The theorem trivially holds for games with at most one live position. We now proceed by induction over the number of live positions.

For every live position v in $\mathcal{G}$ and for $\sigma = 0, 1$, we define the game $\mathcal{G}[v, \sigma]$, which is the same as $\mathcal{G}$ except that we change v to a terminal position where Player σ wins. (Technically this means that we put v into $V_{1-\sigma}$ and delete all outgoing edges from v.) By the induction hypothesis, the Forgetful Determinacy Theorem holds for $\mathcal{G}[v, \sigma]$, and we write $W_0[v, \sigma]$ and $W_1[v, \sigma]$ for the winning regions of $\mathcal{G}[v, \sigma]$.

It suffices to show that for every live position u in $\mathcal{G}$, one of the players has a positional strategy to win $\mathcal{G}$ from u. By Exercise 3.3.24, these strategies can then be combined into positional strategies that win on the entire winning regions.

Clearly,

$$W_0[v, 1] \subseteq W_0 \text{ and } W_1[v, 0] \subseteq W_1.$$

Moreover, any positional strategy f for Player σ that is winning from position u in the game $\mathcal{G}[v, 1-\sigma]$ is also winning from u in the game $\mathcal{G}$ and avoids v (i.e. no play that starts at u and is consistent with f ever hits position v). Now let

$$A_\sigma := \bigcup_{v \text{ live}} W_\sigma[v, 1-\sigma].$$

We call positions $u \in A_\sigma$ *strong winning positions* for Player σ because, informally speaking, Player σ can win $\mathcal{G}$ from u even if she gives away some live positions to her opponent. Similarly, positions outside $A_0 \cup A_1$ are called *weak positions.* It remains to show that from weak positions also, one of the players has a positional winning strategy. In fact, one of the players wins, with a positional winning strategy, from *all* weak positions.

By the induction hypothesis, if u is not in $A_{1-\sigma}$, then, for *all* live positions v of $\mathcal{G}$, we have that $u \in W_\sigma[v, \sigma]$ and, moreover, Player σ has a positional strategy f_v by which, starting at any position $u \notin A_{1-\sigma}$, she either wins or eventually reaches v.

We distinguish two cases, depending on whether or not there exist strong winning positions that are live (terminal positions are, of course, always strong).

Case (i). Suppose that there exists a live position $v \in A_\sigma$. In this case, Player σ also wins from every weak position u.

We already know that Player σ has a positional strategy f to win $\mathcal{G}$ from v, and a positional strategy f_v by which she either wins $\mathcal{G}$ or reaches v from u. We can easily combine f and f_v into a positional winning strategy f^* to win $\mathcal{G}$ from u: we set $f^*(x) := f(x)$ if f is winning from x, and $f^*(x) := f_v(x)$ otherwise.

Case (ii). Suppose now that all live positions are weak. We claim that in this case, Player 0 wins from all live (i.e. all weak) positions if the minimal priority on $\mathcal{G}$ is even, and Player 1 wins from all live positions if the minimal priority is odd.

Since all live positions are weak, we already know that Player σ has, for every live position y, a positional strategy f_y by which she either wins or reaches y from any live position in $\mathcal{G}$.

Take now a live position v of minimal priority, and put $\sigma = 0$ if $\Omega(v)$ is even, and $\sigma = 1$ if $\Omega(v)$ is odd. In addition, pick any live position $w \neq v$. We combine the positional winning strategies f_v and f_w into a new positional strategy f with

$$f(x) := \begin{cases} f_w(x) & \text{if } x = v \\ f_v(x) & \text{otherwise.} \end{cases}$$

We claim that f is a winning strategy for Player σ from all live positions of $\mathcal{G}$. If a play in $\mathcal{G}$ in which Player 0 moves according to f hits v only finitely often, then this play eventually coincides with a play consistent with f_v, and is therefore won by Player σ. But if the play hits v infinitely often, the minimal priority seen infinitely often is $\Omega(v)$, and hence Player σ wins also in this case. □

Exercise 3.3.26. Let $\mathcal{G}$ be a parity game with winning sets W_0 and W_1. Obviously every positional winning strategy for Player 0 has to remain inside W_0, i.e. $f(V_0 \cap W_0) \subseteq W_0$. However, remaining inside the winning region does not suffice for winning a game! Construct a parity game and a positional strategy f for Player 0 such that all plays consistent with f remain insiside W_0, yet are won by Player 1. *Hint*: a trivial game with two positions suffices.

Exercise 3.3.27. A **future game** is any game on a game graph $\mathcal{G} = (V, V_0, V_1, E)$ where the winning condition does not depend on finite prefixes of plays. This means that whenever $\pi = v_0 v_1 \cdots$ and $\pi' = v'_0 v'_1 \cdots$ are two infinite plays of $\mathcal{G}$ such that for some n and m $v_m v_{m+1} \cdots = v'_n v'_{n+1} \cdots$, then π and π' are won by the same player. Obviously parity games are a special case of future games.

Prove that for every future game $\mathcal{G}$, the winning region of Player 0 is a fixed point (not necessarily the least one) of the operator F_ψ, defined by the formula $\psi(X) := (V_0 \wedge \Diamond X) \vee (V_1 \wedge \Box X)$. Since F_ψ is monotone, the least and greatest fixed points exist, and $\mathbf{lfp}(F_\psi) \subseteq W_0 \subseteq \mathbf{gfp}(F_\psi)$. Find conditions (on parity games) implying that $W_0 = \mathbf{lfp}(F_\psi)$ or that $W_0 = \mathbf{gfp}(F_\psi)$.

Theorem 3.3.28. *It can be decided in* NP $\cap$ Co-NP *whether a given position in a parity game is a winning position for Player 0.*

Proof. A node v in a parity game $\mathcal{G} = (V, V_0, V_1, E, \Omega)$ is a winning position for Player σ if there exists a positional strategy $f : V_\sigma \to V$ which is winning from position v. It therefore suffices to show that the question of whether a given $f : V_\sigma \to V$ is a winning strategy for Player σ from position v can be decided in polynomial time. We prove this for Player 0; the argument for Player 1 is analogous.

Given $\mathcal{G}$ and $f : V_0 \to V$ we obtain a reduced game graph $\mathcal{G}_f = (V, E_f)$ by keeping only the moves that are consistent with f, i.e.

$$E_f = \{(v, w) : (v \in V_\sigma \wedge w = f(v)) \vee (v \in V_{1-\sigma} \wedge (v, w) \in E\}.$$

In this reduced game, only the opponent, Player 1, makes non-trivial moves. We call a cycle in (V, E_f) odd if the smalest priority of its nodes is odd. Clearly, Player 0 wins $\mathcal{G}$ from position v via strategy f if, and only if, in $\mathcal{G}_f$, no odd cycle and no terminal position $w \in V_0$ are reachable from v. Since the reachability problem is solvable in polynomial time, the claim follows. $\square$

In fact, Jurdziński [72] proved that the problem is in UP $\cap$ Co-UP, where UP denotes the class of NP-problems with unique witnesses. The best known deterministic algorithms to compute winning partitions of parity games have running times that are polynomial with respect to the size of the game graph, but exponential with respect to the index of the game [73].

Theorem 3.3.29. *The winning partition of a parity game $\mathcal{G} = (V, V_0, V_1, E, \Omega)$ of index d can be computed in space $O(d \cdot |E|)$ and time*

$$O\left(d \cdot |E| \cdot \left(\frac{|V|}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}\right).$$

The Unfolding of a Parity Game

Let $\mathcal{G} = (V, V_0, V_1, E, \Omega)$ be a parity game. We assume that the minimal priority in the range of Ω is even, and that every node v with minimal priority has a unique successor $s(v)$ (i.e. $vE = \{s(v)\}$). This is no loss of generality. We can always tranform a parity game in such a way that all nodes with non-maximal priority have unique successors (i.e. choices are made only at the least relevant nodes). If the smallest priority in the game is odd, we consider instead the dual game (with the roles of the players switched and priorities decreased by one).

Let T be the set of nodes with minimal priority and let $\mathcal{G}^-$ be the game obtained by deleting from $\mathcal{G}$ all edges $(v, s(v)) \in T \times V$ so that the nodes in T become terminal positions. We define the **unfolding** of $\mathcal{G}$ as a sequence of games $\mathcal{G}^\alpha$ (where α ranges over the ordinals) which all coincide with $\mathcal{G}^-$ up to the winning conditions for the terminal positions $v \in T$. For every α, we define a decomposition $T = T_0^\alpha \cup T_1^\alpha$, where T_σ^α is the set of $v \in T$ in which we declare, for the game $\mathcal{G}^\alpha$, Player σ to be the winner. Further, for every α, we write W_σ^α for the winning set of Player σ in the game $\mathcal{G}^\alpha$. Note that W_σ^α depends of course on the decomposition $T = T_0^\alpha \cup T_1^\alpha$ (this also applies concerning positions outside T). In turn, the decomposition of T for $\alpha + 1$ depends on the winning sets W_σ^α in $\mathcal{G}^\alpha$. We set

$$\begin{aligned} T_0^0 &:= T \\ T_0^{\alpha+1} &:= \{v \in T : s(v) \in W_0^\alpha\} \\ T_0^\lambda &:= \bigcap_{\alpha<\lambda} T_0^\alpha \text{ for limit ordinals } \lambda. \end{aligned}$$

By determinacy, $V = W_0^\alpha \cup W_1^\alpha$ for all α, and with increasing α, the winning sets of Player 0 are decreasing and the winning sets of Player 1 are increasing:

$$\begin{aligned} &W_0^0 \supseteq W_0^1 \supseteq \cdots W_0^\alpha \supseteq W_0^{\alpha+1} \supseteq \cdots \\ &W_1^0 \subseteq W_1^1 \subseteq \cdots W_1^\alpha \subseteq W_1^{\alpha+1} \subseteq \cdots . \end{aligned}$$

Hence there exists an ordinal α (whose cardinality is bounded by the cardinality of V) for which $W_0^\alpha = W_0^{\alpha+1} =: W_0^\infty$ and $W_1^\alpha = W_1^{\alpha+1} =: W_1^\infty$. We claim that these fixed points coincide with the winning sets W_0 and W_1 for the original game $\mathcal{G}$.

Lemma 3.3.30 (Unfolding Lemma). *$W_0 = W_0^\infty$ and $W_1 = W_1^\infty$.*

Proof. It suffices to define a strategy f for Player 0 and a strategy g for Player 1 for the game $\mathcal{G}$, by means of which Player σ wins from all positions $v \in W_\sigma^\infty$.

First, we fix a winning strategy f^α for Player 0 in $\mathcal{G}^\alpha$, with winning set $W_0^\alpha = W_0^\infty$. Note that f^α can be trivially extended to a strategy f for the game $\mathcal{G}$, since the nodes in T have unique successors in $\mathcal{G}$. We claim that f is in fact a winning strategy in $\mathcal{G}$ from all positions $v \in W_0^\alpha$.

To see this, consider any play $v_0v_1v_2\ldots$ in $\mathcal{G}$ from position $v_0 \in W_0^\alpha$ against f. Such a play can never leave W_0^α. If $v_i \in W_0^\alpha \setminus T$, then $v_{i+1} \in W_0^\alpha$ because f is a winning strategy for $\mathcal{G}^\alpha$; and if $v_i \in W_0^\alpha \cap T = W_0^{\alpha+1} \cap T$, then $v_i \in T_0^{\alpha+1}$, which implies, by the definition of $T_0^{\alpha+1}$, that $v_{i+1} = s(v_i) \in W_0^\alpha$. But a play that never leaves W_0^α is necessarily won by Player 0: either it goes only finitely often through positions in T, and then coincides from a certain point onwards with a winning play in $\mathcal{G}^\alpha$, or it goes infinitely often through positions in T, in which case Player 0 wins because the minimal priority that is hit infinitely often is even.

To construct a winning strategy for Player 1 in the game $\mathcal{G}$, we define, for every node $v \in W_1^\infty$, the ordinal

$$\rho(v) := \min\{\beta : v \in W_1^\beta\}.$$

We fix, for every ordinal α, a winning strategy g^α for Player 1 with winning set W_1^α in the game $\mathcal{G}^\alpha$, and set

$$g(v) := g^{\sigma(v)}(v) \text{ for all } v \in V_1 \setminus T$$

and $g(v) := s(v)$ for $v \in V_1 \cap T$.

Consider any play $v_0v_1v_2\dots$ in $\mathcal{G}$ from position $v_0 \in W_1^\infty$ against g. We claim that whenever $v_i \in W_1^\infty$, then

(1) $v_{i+1} \in W_1^\infty$,
(2) $\rho(v_{i+1}) \leq \rho(v_i)$, and
(3) if $v_i \in T$, then $\rho(v_{i+1}) < \rho(v_i)$.

If $v_i \in W_1^\infty \setminus T$ and $\rho(v_i) = \alpha$, then $v_i \in W_1^\alpha$, and therefore (since Player 1 moves locally according to his winning strategy g^α and Player 0 cannot leave winning sets of her opponent) $v_{i+1} \in W_1^\alpha$. But if $v_i \in W_1^\infty \cap T$ and $\rho(v_i) = \alpha$, then $v_i \in T_1^\alpha$, $\alpha = \beta + 1$ is a successor ordinal, and $v_{i+1} = s(v_i) \in W_1^\beta$ (by the definition of T_1^α). Hence $\rho(v_{i+1}) \leq \beta < \rho(v_i)$.

Properties (1), (2), and (3) imply that the play stays inside W_1^∞ and that the values $\rho(v)$ are decreasing. Since there are no infinite strictly descending chains of ordinals, the play eventually remains inside W_1^α, for a fixed α, and outside T (since moves from T would reduce the value of $\sigma(v)$). Hence the play eventually coincides with a play in $\mathcal{G}^\alpha$ in which Player 1 plays according to his winning strategy g^α. Thus, Player 1 wins. $\square$

3.3.5 Model-Checking Games for Least Fixed-Point Logic

For the purpose of defining evaluation games for LFP-formulae and analysing the complexity of model checking, it is convenient to make the following assumptions. First, the fixed-point formulae should not contain parameters (the reason for this will be discussed below). Second, the formula should be in negation normal form, i.e. negations apply to atoms only, and third, it should be **well-named**, i.e. every fixed-point variable is bound only once and the free second-order variables are distinct from the fixed-point variables. We write $D_\psi(T)$ for the unique subformula in ψ of the form $[\mathbf{fp}T\overline{x}\,.\,\varphi(T,\overline{x})]$ (where **fp** means either **lfp** or **gfp**). For technical reasons, we assume, finally, that each fixed-point variable T occurs in $D_\psi(T)$ only inside the scope of a quantifier. This is a common assumption that does not affect the expressive power. We say that T' **depends** on T if T occurs free in $D_\psi(T')$. The transitive closure of this dependency relation is called the **dependency order**, denoted by $\sqsubset_\psi$. The **alternation level** $al_\psi(T)$ of T in ψ is the maximal number of alternations between least and greatest fixed-point variables on the $\sqsubset_\psi$-paths from

T. The **alternation depth** $ad(\psi)$ of a fixed-point formula ψ is the maximal alternation level of its fixed point variables.

Consider now a finite structure $\mathfrak{A}$ and an LFP-formula $\psi(\overline{x})$, which we assume to be well-named, in negation normal form, and without parameters. The model-checking game $\mathcal{G}(\mathcal{A}, \psi(\overline{a}))$ is a parity game. As in the case of first-order logic, the positions of the game are expressions $\varphi(\overline{b})$, i.e. subformulae of ψ that are instantiated by elements of $\mathfrak{A}$. The initial position is $\psi(\overline{a})$. The moves are as in the first-order game, except for the positions associated with fixed-point formulae and with fixed-point atoms. At such positions there is a unique move (by Falsifier, say) to the formula defining the fixed point. For a more formal definition, recall that as ψ is well-named, there is, for any fixed-point variable T in ψ, a unique subformula $[\mathbf{fp}\ T\overline{x} \, . \, \varphi(T,\overline{x})](\overline{y})$. From position $[\mathbf{fp}T\overline{x} \, . \, \varphi(T,\overline{x})](\overline{b})$, Falsifier moves to $\varphi(T,\overline{b})$, and from any fixed point atom $T\overline{c}$, she moves to the position $\varphi(T,\overline{c})$.

Hence the case where we do not have fixed points the game is the usual model-checking game for first-order logic. Next, we consider the case of a formula with only one fixed-point operator, which is an **lfp**. The intuition is that from position $[\mathbf{lfp}\ T\overline{x} \, . \, \varphi(T,\overline{x})](\overline{b})$, Verifier tries to establish that $\overline{b}$ enters T at some stage α of the fixed-point induction that is defined by φ on $\mathfrak{A}$. The game goes to $\varphi(T,\overline{b})$ and from there, as φ is a first-order formula, Verifier can either win the φ-game in a finite number of steps, or force it to a position $T\overline{c}$, where $\overline{c}$ enters the fixed point at some stage $\beta < \alpha$. The game then resumes at position $\varphi(\overline{c})$, associated again with φ. As any descending sequence of ordinals is finite, Verifier will win the game in a finite number of steps. If the formula is not true, then Falsifier can either win in a finite number of steps or force the play to go through infinitely many positions of the form $T\overline{c}$. Hence, these positions should be assigned priority 1 (and all other positions higher priorities) so that such a play will be won by Falsifier. For **gfp**-formulae, the situation is reversed. Verifier wants to force an infinite play, going infinitely often through positions $T\overline{c}$, so **gfp**-atoms are assigned priority 0.

In the general case, we have a formula ψ with nested least and greatest fixed points, and in an infinite play of $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$ one may see different fixed point variables infinitely often. But one of these variables is then the smallest with respect to the dependency order $\sqsubset_\psi$. It can be shown that $\mathfrak{A} \models \psi$ iff this smallest variable is a **gfp**-variable (provided the players play optimally).

Hence, the priority labelling should assign even priorities to **gfp**-atoms and odd priorities to **lfp**-atoms. Further, if $T \sqsubset_\psi T'$ and T, T' are fixed-point variables of different kinds, then T-atoms should get a lower priority than T'-atoms.

As the index of a parity game is the main source of difficulty in computing winning sets, the number of different priorities should be kept as small as possible. We can avoid the factor of 2 appearing in common constructions of this kind by adjusting the definitions of the alternation level and alternation depth, setting $al^*_\psi(T) := al_\psi(T) + 1$ if $al_\psi(T)$ is even or odd and T is an **lfp**-

variable or a **gfp**-variable, respectively. In all other cases, $al^*_\psi(T) = al_\psi(T)$. Finally, let $ad^*(\psi)$ be the maximal value of $ad^*_\psi(T)$ for the fixed-point variables in ψ. The priority labelling Ω on positions of $\mathcal{G}(\mathfrak{A}, \psi)$ is then defined by $\Omega(T\overline{b}) = al^*_\psi(T)$ for fixed-point atoms, and $\Omega(\varphi(\overline{b})) = ad^*(\psi)$ for all other formulae.

This completes the definition of the game $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$. Note that the priority labelling has the properties described above, and that the index of $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$ is at most $ad(\psi) + 1$.

Theorem 3.3.31. *Let $\psi(\overline{x})$ be a well-named and parameter-free* LFP*-formula in negation normal form, and let $\mathfrak{A}$ be a relational structure. $\mathfrak{A} \models \psi(\overline{a})$ if and only if Player 0 has a winning strategy for the parity game $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$.*

Proof. This is proved by induction on ψ. The interesting case concerns fixed-point formulae $\psi(\overline{x}) := [\mathbf{gfp} T\overline{x} \,.\, \varphi(\overline{x})](\overline{x})$.

In the game $\mathcal{G}(\mathcal{A}, \psi(\overline{a}))$, the positions of minimal priority are the fixed-point atoms $T\overline{b}$, which have unique successors $\varphi(\overline{b})$. By the induction hypothesis we know that, for every interpretation T_0 of T, $(\mathfrak{A}, T_0) \models \varphi(\overline{a})$ iff Player 0 has a winning strategy for $\mathcal{G}((\mathfrak{A}, T_0), \varphi(\overline{a}))$. By the unfolding of greatest fixed points, we also know that $\mathfrak{A} \models [\mathbf{gfp} T\overline{x} \,.\, \varphi(\overline{x})](\overline{a})$ if $(\mathfrak{A}, T^\alpha) \models \varphi(\overline{a})$ for all approximations T^α.

By ordinal induction, one can immediately see that the games $\mathcal{G}((\mathfrak{A}, T^\alpha), \varphi(\overline{a}))$ coincide with the unfolding of the game $\mathcal{G} = \mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$ to the games $\mathcal{G}^\alpha$. By the Unfolding Lemma, we conclude that Player 0 wins the game $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$ if, and only if, she wins all games $\mathcal{G}^\alpha$ which is the case if, and only if, $(\mathfrak{A}, T^\alpha) \models \varphi(\overline{a})$ for all α, which is equivalent to $\mathfrak{A} \models \psi(\overline{a})$.

For least fixed-point formulae we proceed by dualization. □

Clearly, the size of the game $\mathcal{G}(\mathfrak{A}, \psi(\overline{a}))$ (and the time complexity of its construction) is bounded by $|\mathrm{cl}(\psi)| \cdot |A|^{\mathrm{width}(\psi)}$. Hence, for LFP-formulae of bounded width, the size of the game is polynomially bounded.

Corollary 3.3.32. *The model-checking problem for* LFP*-formulae of bounded width (and without parameters) is in* NP ∩ Co-NP*, in fact in* UP ∩ Co-UP.

As formulae of the μ-calculus can be viewed as LFP-formulae of width 2, the same bound applies to L_μ. (For a different approach to this problem, which does not mention games explicitly, see [100].) It is a well-known open problem whether the model-checking problem for L_μ can be solved in polynomial time.

Exercise 3.3.33. Prove that if the model-checking problem for L_μ can be solved in polynomial time, then the same is true for (parameter-free) LFP-formulae of width k, for any fixed $k \in \mathbb{N}$. *Hint*: given a finite structure $\mathfrak{A} = (A, R_1, \dots, R_m)$, with relations of R_i of arities $r_i \leq k$, let $G^k(\mathfrak{A})$ be the transition system with universe A^k, unary relations $R^*_i = \{(a_1, \dots, a_k) : (a_1, \dots, a_{r_i}) \in R_i\}$ and $I_{ij} = \{(a_1, \dots, a_k) : a_i = a_j\}$, and binary relations $E_j = \{(\overline{a}, \overline{b}) : a_i = b_i \text{ for } i \neq j\}$ (for $j = 1, \dots, k$) and $E_\sigma = \{(\overline{a}, \overline{b}) : b_i =$

$a_{\sigma(i)}$ for $i = 1, \dots, k\}$ for each substitution $\sigma : \{1, \dots, k\} \to \{1, \dots, k\}$. Translate formulae $\psi \in$ LFP of width k into formulae $\psi^* \in L_\mu$ such that $\mathfrak{A} \models \psi(\overline{a})$ iff $G^k(\mathfrak{A}), \overline{a} \models \psi^*$. (See [55, pp. 110–111] for details.)

By Theorem 3.3.29, we obtain the following deterministic complexity bounds for LFP model checking.

Theorem 3.3.34. *Given a finite structure $\mathfrak{A}$ and a formula $\psi(\overline{a})$ of width k and alternation depth d, it can be decided whether $\mathfrak{A} \models \psi(\overline{a})$ in space $O(d \cdot |\mathrm{cl}(\psi)| \cdot |A|^k)$ and time*

$$O\left(d^2 \cdot \left(\frac{|\mathrm{cl}(\psi)| \cdot |A|^k}{\lfloor (d+1)/2 \rfloor}\right)^{\lfloor (d+3)/2 \rfloor}\right).$$

Corollary 3.3.35. *The model-checking problem for* LFP*-formulae of bounded width and bounded alternation depth is solvable in polynomial time.*

Fixed-Point Formulae with Parameters

We have imposed the condition that the fixed-point formulae do not contain parameters. If parameters are allowed, then, at least with a naive definition of width, Corollary 3.3.32 is no longer true (unless UP = PSPACE). The intuitive reason is that parameters allow us to 'hide' first-order variables in fixed-point variables. Indeed, Dziembowski [37] proved that QBF, the evaluation problem for quantified Boolean formulae, can be reduced to evaluating LFP-formulae with two first-order variables (but an unbounded number of monadic fixed-point variables) on a fixed structure with three elements. Hence the expression complexity of evaluating such formulae is PSPACE-complete. A similar argument works for the case where also the number of fixed-point variables is bounded, but the structure is not fixed (combined complexity rather than expression complexity). We remark that the collection of all unwindings in infinitary logic of LFP-formulae with k variables, including parameters, is not contained in any bounded width fragment of infinitary logic.

LFP-Formulae of Unbounded Width

For LFP-formulae of unbounded width, Theorem 3.3.34 gives only an exponential time bound. In fact, this cannot be improved, even for very simple LFP-formulae [99].

Theorem 3.3.36 (Vardi). *The model-checking problem for* LFP*-formulae (of unbounded width) is* EXPTIME*-complete, even for formulae with only one fixed-point operator, and on a fixed structure with only two elements.*

We defer the hardness proof to Section 3.3.10, where we shall show that the expression complexity is EXPTIME-hard even for Datalog, which is a more restricted formalism than LFP.

3.3.6 Definability of Winning Regions in Parity Games

We have seen that the model-checking problem for the μ-calculus or LFP can be reduced to the problem of computing winning regions in parity games. In fact, there is also a reduction in the reverse direction. We can represent any parity game $\mathcal{G} = (V, V_0, V_1, E, \Omega)$ with a priority function $\Omega : V \to \{0, \dots d-1\}$ by a transition system $(V, E, V_0, V_1, P_0, \dots, P_{d-1})$, where $P_i = \{V : \Omega(v) = i\}$. We can then construct, for every fixed $d \in \mathbb{N}$, a formula Win_d of the μ-calculus that defines the winning region of Player 0 in any parity game with priorities $0, \dots, d-1$. We set

$$\mathrm{Win}_d = \nu X_0 \mu X_1 \nu X_2 \dots \lambda X_{d-1} \bigvee_{j=0}^{d-1} \big((V_0 \wedge P_j \wedge \Diamond X_j) \vee (V_1 \wedge P_j \wedge \Box X_j) \big).$$

In this formula, the fixed-point operators alternate between ν and μ, and hence $\lambda = \nu$ if d is odd, and $\lambda = \mu$ if d is even.

Theorem 3.3.37. *For every $d \in \mathbb{N}$, the formula* Win_d *defines the winning region of Player 0 in parity games with priorities* $0, \dots, d-1$.

Proof. We have to show that, for any parity game $\mathcal{G} = (V, V_0, V_1, P_0, \dots, P_{d-1})$ and every position $v \in V$,

$$\mathcal{G}, v \models \mathrm{Win}_d \iff \text{Player 0 has a winning strategy for } \mathcal{G} \text{ from } v.$$

To see this, let $\mathcal{G}^*$ be the model-checking game for the formula Win_d on $\mathcal{G}, v$ and identify Verifier with Player 0 and Falsifier with Player 1. Hence, Player 0 has a winning strategy for $\mathcal{G}^*$ if, and only if, $\mathcal{G}, v \models \mathrm{Win}_d$.

By the construction of model-checking games, $\mathcal{G}^*$ has positions of the form (φ, u), where $u \in V$ and φ is a subformula of Win_d. The priority of a position (X_i, u) is i, and when φ is not a fixed point variable, the priority of (φ, u) is d.

We claim that the game $\mathcal{G}^*$ is essentially, i.e. up to elimination of stupid moves and contraction of several moves into one, the same as the the original game $\mathcal{G}$. To see this, we compare playing $\mathcal{G}$ from a current position $u \in V_0 \cup P_i$ with playing $\mathcal{G}^*$ from any position (φ_k, u), where φ_k is the subformula of Win_d that starts with νX_k or μX_k.

In $\mathcal{G}$, Player 0 selects at position u a successor $w \in uE$, and the play proceeds from w. In $\mathcal{G}^*$, the play goes from (φ_k, u) through positions $(\varphi_{k+1}, u) \dots, (\varphi_{d-1}, u)$ to (ϑ, u), where

$$\vartheta = \bigvee_{j=0}^{d-1} \big((V_0 \wedge P_j \wedge \Diamond X_j) \vee (V_1 \wedge P_j \wedge \Box X_j) \big).$$

The only reasonable choice for Verifier (Player 0) at this point is to move to the position $(V_0 \wedge P_i \wedge \Diamond X_i, u)$, since with any other move she would lose immediately. But from there, the only reasonable move of Falsifier (Player 1) is to go to position $(\Diamond X_i, u)$, and it is now the turn of Player 0 to select a successor $w \in vE$ and move to position (X_i, w) from which the play proceeds to (φ_i, w).

Thus one move from u to w in $\mathcal{G}$ corresponds to a sequence of moves in $\mathcal{G}^*$ from (φ_k, u) to (φ_i, w), but the only genuine choice is the move from $(\Diamond X_i, u)$ to (X_i, w), i.e. the choice of a successor $w \in uE$. In $\mathcal{G}$, the position u has priority i, and in $\mathcal{G}^*$ the minimal, and hence relevant, priority that is seen in the sequence of moves from (φ_k, u) to (φ_i, w) is that of (X_i, u) which is also i. The situation for positions $u \in V_1 \cap P_i$ is the same, except that the play in $\mathcal{G}^*$ now goes through $(\Box X_i, u)$ and it is Player 1 who selects a successor $w \in uE$ and moves to (X_i, w).

Hence the (reasonable) choices that have to be made by the players in $\mathcal{G}^*$ and the relevant priorities that are seen are the same as in a corresponding play of $\mathcal{G}$. Thus, Player 0 has a winning strategy for $\mathcal{G}$ from v if, and only if, Player 0 has a winning strategy for $\mathcal{G}^*$ from position (φ_0, v). But since $\mathcal{G}^*$ is the model-checking game for Win_d on $\mathcal{G}$, with initial position (φ_0, v), this is the case if, and only if, $\mathcal{G}, v \models \mathrm{Win}_d$. $\Box$

Corollary 3.3.38. *The following three problems are algorithmically equivalent, in the sense that if one of them admits a polynomial-time algorithm, then all of them do.*

(1) Computing winning regions in parity games.
(2) The model-checking problem for LFP*-formulae of width at most k, for any $k \geq 2$.*
(3) The model-checking problem for the modal μ-calculus.

The formulae Win_d also play an important role in the study of the alternation hierarchy of the modal μ-calculus. Clearly, Win_d has alternation depth d and it has been shown that there is no formula in the μ-calculus with alternation depth $< d$ can be equivalent to Win_d. Hence the alternation hierarchy of the μ-calculus is strict [4, 20].

3.3.7 Simultaneous Fixed-Point Inductions

A more general variant of LFP permits simultaneous inductions over several formulae. A simultaneous induction is based on on a system of operators of the form

$$\begin{aligned} F_1 : \mathcal{P}(B_1) \times \cdots \times \mathcal{P}(B_m) &\longrightarrow \mathcal{P}(B_1) \\ &\vdots \\ F_m : \mathcal{P}(B_1) \times \cdots \times \mathcal{P}(B_m) &\longrightarrow \mathcal{P}(B_m), \end{aligned}$$

forming together an operator

$$F = (F_1, \dots, F_m) : \mathcal{P}(B_1) \times \cdots \times \mathcal{P}(B_m) \longrightarrow \mathcal{P}(B_1) \times \cdots \times \mathcal{P}(B_m).$$

Inclusion on the product lattice $\mathcal{P}(B_1) \times \cdots \times \mathcal{P}(B_m)$ is componentwise. Accordingly, F is monotone if, whenever $X_i \subseteq Y_i$ for all i, then also $F_i(\overline{X}) \subseteq F_i(\overline{Y})$ for all i.

Everything said above about least and greatest fixed points carries over to simultaneous induction. In particular, a monotone operator F has a least fixed point $\mathbf{lfp}(F)$ which can be constructed inductively, starting with $\overline{X}^0 = (\emptyset, \dots, \emptyset)$ and iterating F until a fixed point $\overline{X}^\infty$ is reached.

One can extend the logic LFP by a simultaneous fixed point formation rule.

Definition 3.3.39. Simultaneous least fixed-point logic, denoted by S-LFP, is the extension of first-order logic by the following rule.

Syntax. Let $\psi_1(\overline{R}, \overline{x}_1), \dots, \psi_m(\overline{R}, \overline{x}_m)$ be formulae of vocabulary $\tau \cup \{R_1, \dots, R_m\}$, with only positive occurrences of $R_1, \dots, R_m$, and, for each $i \leq m$, let $\overline{x}_i$ be a sequence of variables matching the arity of R_i. Then

$$S := \begin{cases} R_1\overline{x}_1 & := \psi_1 \\ & \vdots \\ R_m\overline{x}_m & := \psi_m \end{cases}$$

is a *system of update rules*, which is used to build formulae $[\mathbf{lfp}\ R_i : S](\overline{t})$ and $[\mathbf{gfp}\ R_i : S](\overline{t})$ (for any tuple $\overline{t}$ of terms whose length matches the arity of R_i).

Semantics. On each structure $\mathfrak{A}$, S defines a monotone operator $S^{\mathfrak{A}} = (S_1, \dots, S_m)$ mapping tuples $\overline{R} = (R_1, \dots, R_m)$ of relations on A to $S^{\mathfrak{A}}(\overline{R}) = (S_1(\overline{R}), \dots, S_m(\overline{R}))$ where $S_i(\overline{R}) := \{\overline{a} : (\mathfrak{A}, \overline{R}) \models \psi_i(\overline{R}, \overline{a})\}$. As the operator is monotone, it has a least fixed point $\mathbf{lfp}(S^{\mathfrak{A}}) = (R_1^\infty, \dots, R_m^\infty)$. Now $\mathfrak{A} \models [\mathbf{lfp}R_i : S](\overline{a})$ if $\overline{a} \in R_i^\infty$. Similarly for greatest fixed points.

Example 3.3.40. We return to the circuit value problem for circuits with fan-in 2 and NAND gates (see Example 3.2.14). Simultaneous LFP-definitions of the nodes evaluating to true and false in the given circuit (V, E, I^+, I^-) are given by the formulae $[\mathbf{lfp}T : S](z)$ and $[\mathbf{lfp}F : S](z)$, respectively, where S is the system

$$\begin{aligned} Tz &:= I^+z \lor \exists x(Exz \land Fx) \\ Fz &:= I^-z \lor \exists x \exists y(Exz \land Eyx \land x \neq y \land Tx \land Ty). \end{aligned}$$

Elimination of Simultaneous Fixed-Points

The question arises of whether simultaneous fixed points provide more expressive power than simple ones. We shall prove that this is not the case. Simultaneous least fixed points can be simulated by nested simple ones, via a technique that is sometimes called the Bekic principle [5]. We shall consider only the case of two monotone operators

$$\begin{aligned} F &: \mathcal{P}(A) \times \mathcal{P}(B) \to \mathcal{P}(A) \\ G &: \mathcal{P}(A) \times \mathcal{P}(B) \to \mathcal{P}(B). \end{aligned}$$

We write (F^∞, G^∞) for the least fixed point of the combined operator (F, G). For any fixed $X \subseteq A$, the operator $G_X : \mathcal{P}(B) \to \mathcal{P}(B)$ with $G_X(Y) := G(X, Y)$ is also monotone, and therefore has a least fixed point $\mathbf{lfp}(G_X) \subseteq B$.

Lemma 3.3.41. *The operator E on $\mathcal{P}(A)$, defined by $E(X) := F(X, \mathbf{lfp}(G_X))$, is monotone and has the least fixed point $\mathbf{lfp}(E) = F^\infty$.*

Proof. If $X \subseteq X'$, then a trivial induction shows that $G_X^\alpha \subseteq G_{X'}^\alpha$ for all stages G_X^α and $G_{X'}^\alpha$ of the induced operators G_X and $G_{X'}$. As a consequence, $\mathbf{lfp}(G_X) \subseteq \mathbf{lfp}(G_{X'})$ and $E(X) = F(X, \mathbf{lfp}(G_X)) \subseteq F(X', \mathbf{lfp}(G_{X'})) = E(X')$. This shows that E is monotone.

Note that $\mathbf{lfp}(G_{F^\infty}) \subseteq G^\infty$, because $G_{F^\infty}(G^\infty) = G(F^\infty, G^\infty) = G^\infty$. Hence G^∞ is a fixed point of G_{F^∞} and therefore contains the least fixed point $\mathbf{lfp}(G_{F^\infty})$. Further,

$$E(F^\infty) = F(F^\infty, \mathbf{lfp}(G_{F^\infty})) \subseteq F(F^\infty, G^\infty) = F^\infty.$$

As $\mathbf{lfp}(E) = \bigcap\{X : E(X) \subseteq X\}$ it follows that $\mathbf{lfp}(E) \subseteq F^\infty$.

It remains to show that $F^\infty \subseteq \mathbf{lfp}(E)$. We proceed by induction, showing that the stages (F^α, G^α) of the operator (F, G) and the stages E^α of E satisfy

$$(F^\alpha, G^\alpha) \subseteq (\mathbf{lfp}(E), \mathbf{lfp}(G_{\mathbf{lfp}(E)})).$$

For $\alpha = 0$, this is clear. Further,

$$\begin{aligned} F^{\alpha+1} &= F(F^\alpha, G^\alpha) \subseteq F(\mathbf{lfp}(E), \mathbf{lfp}(G_{\mathbf{lfp}(E)})) = E(\mathbf{lfp}(E) = \mathbf{lfp}(E) \\ G^{\alpha+1} &= G(F^\alpha, G^\alpha) \subseteq G(\mathbf{lfp}(E), \mathbf{lfp}(G_{\mathbf{lfp}(E)})) = G_{\mathbf{lfp}(E)}(\mathbf{lfp}(G_{\mathbf{lfp}(E)}) \\ &= \mathbf{lfp}(G_{\mathbf{lfp}(E)}). \end{aligned}$$

Finally, for limit ordinals the induction argument is trivial. □

We are now ready to show that for any system

$$S := \begin{cases} R_1\overline{x}_1 & := \psi_1 \\ & \vdots \\ R_m\overline{x}_m & := \psi_m \end{cases}$$

the formulae $[\mathbf{lfp}\ R_i : S](\overline{x})$ are equivalent to simple LFP formulae. Further, the translation does not increase the number and arity of the fixed-point variables $R_1, \ldots, R_m$, nor the alternation depth (i.e. the changes between least and greatest fixed points). It therefore remains valid for interesting fragments of LFP, such as monadic LFP and alternation-free LFP, and also for the modal μ-calculus (see [5]). It does, however, increase the nesting depth of fixed-point operators. (We remark that there are alternative elimination techniques that do not increase the nesting depth, but instead augment the arity of the fixed-point operators.)

Theorem 3.3.42. S-LFP $\equiv$ LFP.

Proof. Obviously LFP is contained in S-LFP. For the converse, we restrict our attention to simultaneous inductions over two formulae. The general case is treated by analogous arguments.

Given a system
$$S := \begin{cases} R\overline{x} := \psi(R,T) \\ T\overline{y} := \varphi(R,T) \end{cases}$$
we claim that
$$[\mathbf{lfp}\ R : S](\overline{u}) \equiv [\mathbf{lfp} R\overline{x} \,.\, \psi(R, [\mathbf{lfp} T\overline{y} \,.\, \varphi])](\overline{u})$$
$$[\mathbf{lfp}\ T : S](\overline{v}) \equiv [\mathbf{lfp} T\overline{y} \,.\, \varphi([\mathbf{lfp} R\overline{x} \,.\, \psi], T)](\overline{v}).$$

We shall prove the first equivalence. We fix a structure $\mathfrak{A}$ and consider the operator $S^{\mathfrak{A}} = (F, G)$ with $F : (R,T) \mapsto \{\overline{a} : \mathfrak{A} \models \psi(R,T,\overline{a})\}$ and $G : (R,T) \mapsto \{\overline{a} : \mathfrak{A} \models \varphi(R,T,\overline{a})\}$. Writing (F^∞, G^∞) for the least fixed point of (F,G) we have that $\mathfrak{A} \models [\mathbf{lfp}\ R : S](\overline{a})$ iff $\overline{a} \in F^\infty$.

The formula $\psi(R, [\mathbf{lfp} T\overline{y} \,.\, \varphi])$ defines on $\mathfrak{A}$ the operator $E : R \mapsto F(R, \mathbf{lfp}(G_R))$ with $G_R : T \mapsto G(R,T)$, and we have that $\mathfrak{A} \models [\mathbf{lfp} R\overline{x} \,.\, \psi(R, [\mathbf{lfp} T\overline{y} \,.\, \varphi])](\overline{a})$ iff $\overline{a} \in \mathbf{lfp}(E)$. But, by the previous lemma, $F^\infty = \mathbf{lfp}(E)$. $\square$

While we have shown that simultaneous fixed points do not provide more expressive power, they permit us to write formulae in a more modular and more readable form.

Positive LFP

While LFP and the modal μ-calculus allow arbitrary nesting of least and greatest fixed points, and arbitrary interleaving of fixed points with Boolean operations and quantifiers, classical studies of inductive definability over first-order logic (such as [85]) focus on a more restricted logic. Let LFP_1 (sometimes also called positive LFP) be the extension of first-order logic that is obtained by taking least fixed points of positive first-order formulae (without parameters) and closing them under disjunction, conjunction, and existential and universal quantification, but *not* under negation (for a more formal definition, see the Chap. 2. LFP_1 can be conveniently characterized in terms of simultaneous least fixed points. We just state the result; for a proof see Chap. 2 again.

Theorem 3.3.43. *A query is definable in* LFP_1 *if and only if it is definable by a formula of the form* $[\mathbf{lfp}R : S](\overline{x})$*, where* S *is a system of update rules* $R_i\overline{x} := \varphi_i(\overline{R}, \overline{x})$ *with first-order formulae* φ_i*. Moreover, we can require, without diminishing the expressive power, that each of the formulae* φ_i *in the system is either a purely existential formula or a purely universal formula.*

3.3.8 Inflationary Fixed-Point Logic

LFP is only one instance of a logic with an explicit operator for forming fixed points. A number of other fixed-point extensions of first-order logic (or fragments of it) have been extensively studied in finite model theory. These include inflationary, partial, non-deterministic, and alternating fixed point logics. All of these have in common that they allow the construction of fixed points of operators that are not necessarily monotone.

An operator $G : \mathcal{P}(B) \to \mathcal{P}(B)$ is called **inflationary** if $G(X) \supseteq X$ for all $X \subseteq B$. With any operator F one can associate an inflationary operator G, defined by $G(X) := X \cup F(X)$. In particular, inflationary operators are inductive, so iterating G yields a fixed point, called the **inflationary fixed point** of F.

Exercise 3.3.44. Prove the following facts. (1) Monotone operators need not be inflationary, and inflationary operators need not be monotone. (2) An inflationary operator need not have a least fixed point. (3) The least fixed point of an inflationary operator (if it exists) may be different from the inductive fixed point. (4) However, if F is a monotone operator, then its inflationary fixed point and its least fixed point coincide.

The logic IFP is defined with a syntax similar to that of LFP, but without the requirement that the fixed-point variable occurs only positively in the formula, and with a semantics given by the associated inflationary operator.

Definition 3.3.45. IFP is the extension of first-order logic by the following fixed-point formation rule. For every formula $\psi(R, \overline{x})$, every tuple $\overline{x}$ of variables, and every tuple $\overline{t}$ of terms (such that the lengths of $\overline{x}$ and $\overline{t}$ match the arity of R), we can build a formula $[\mathbf{ifp}R\overline{x} \,.\, \psi](\overline{t})$.

Semantics. On a given structure $\mathfrak{A}$, we have that $\mathfrak{A} \models [\mathbf{ifp}R\overline{x} \,.\, \psi](\overline{t})$ if $\overline{t}^{\mathfrak{A}}$ is contained in the union of the stages R^α of the *inflationary operator* G_ψ defined by $G_\psi(R) := R \cup F_\psi(R)$.

By the last item of Exercise 3.3.44, least and inflationary inductions are equivalent for positive formulae, and hence IFP is at least as expressive as LFP. On finite structures, inflationary inductions reach the fixed point after a polynomial number of iterations, hence every IFP-definable class of finite structures is decidable in polynomial time.

Proposition 3.3.46. IFP *captures* PTIME *on ordered finite structures.*

Least Versus Inflationary Fixed-Points

As both logics capture PTIME, IFP and LFP are equivalent on ordered finite structures. What about unordered structures? It was shown by Gurevich and Shelah [63] that the equivalence of IFP and LFP holds on all finite structures. Their proof does not work on infinite structures, and indeed there are some important aspects in which least and inflationary inductions behave differently. For instance, there are first-order operators (on arithmetic, say) whose inflationary fixed point is not definable as the least fixed point of a first-order operator. Further, the alternation hierarchy in LFP is strict, whereas IFP has a positive normal form (see Exercise 3.3.52 below). Hence it was conjectured by many that IFP might be more powerful than LFP. However, Kreutzer [80] showed recently that IFP is equivalent to LFP on arbitrary structures. Both proofs, by Gurevich and Shelah and by Kreutzer, rely on constructions showing that the *stage comparison relations* of inflationary inductions are definable by **lfp** inductions.

Definition 3.3.47. For every inductive operator $F : \mathcal{P}(B) \to \mathcal{P}(B)$, with stages X^α and an inductive fixed point X^∞, the ***F*-rank** of an element $b \in B$ is $|b|_F := \min\{\alpha : b \in X^\alpha\}$ if $b \in X^\infty$, and $|b|_F = \infty$ otherwise. The **stage comparison relations** of G are defined by

$$\begin{aligned} a \leq_F b \quad &\text{iff} \quad |a|_F \leq |b|_F < \infty \\ a \prec_F b \quad &\text{iff} \quad |a|_F < |b|_F. \end{aligned}$$

Given a formula $\varphi(R, \overline{x})$, we write $\leq_\varphi$ and $\prec_\varphi$ for the stage comparison relations defined by the operator F_φ (assuming that it is indeed inductive), and $\leq_\varphi^{\text{inf}}$ and $\prec_\varphi^{\text{inf}}$ for the stage comparison relations of the associated inflationary operator $G_\varphi : R \mapsto R \cup \{\overline{a} : \mathfrak{A} \models \varphi(R, \overline{a})\}$.

Example 3.3.48. For the formula $\varphi(T, x, y) := Exy \vee \exists z(Exz \wedge Tyz)$ the relation $\prec_\varphi$ on a graph (V, E) is distance comparison:

$$(a, b) \prec_\varphi (c, d) \text{ iff } \operatorname{dist}(a, b) < \operatorname{dist}(c, d).$$

Stage comparison theorems are results about the definability of stage comparison relations. For instance, Moschovakis [85] proved that the stage comparison relations $\leq_\varphi$ and $\prec_\varphi$ of any positive first-order formula φ are definable by a simultaneous induction over positive first-order formulae. For results on the equivalence of IFP and LFP one needs a stage comparison theorem for IFP inductions.

We first observe that the stage comparison relations for IFP inductions are easily definable in IFP. For any formula $\varphi(T, \bar{x})$, the stage comparison relation $\prec_\varphi^{\text{inf}}$ is defined by the formula

$$[\mathbf{ifp}\overline{x} \prec \overline{y} \,.\, \varphi[T\overline{u}/\overline{u} \prec \overline{x}](\overline{x}) \wedge \neg\varphi[T\overline{u}/\overline{u} \prec \overline{x}](\overline{y})](\overline{x}, \overline{y}).$$

However, what we need to show is that the stage comparison relation for IFP inductions is in fact LFP-definable.

Theorem 3.3.49 (Inflationary Stage Comparison). *For any formula* $\varphi(R, \overline{x})$ *in* FO *or* LFP*, the stage comparison relation* $\prec_\varphi^{\text{inf}}$ *is definable in* LFP*. On finite structures, it is even definable in positive* LFP.

See [38, 63] for proofs in the case of finite structures and [80] for the more difficult construction in the general case. From this result, the equivalence of LFP on IFP follows easily.

Theorem 3.3.50 (Kreutzer). *For every* IFP*-formula, there is an equivalent* LFP*-formula.*

Proof. For any formula $\varphi(R, \overline{x})$, $[\mathbf{ifp} R\overline{x} \,.\, \varphi](\overline{x}) \equiv \varphi(\{\overline{y} : \overline{y} \prec_\varphi^{\text{inf}} \overline{x}\}, \overline{x})$. □

Stage comparison theorems also have other interesting consequences. For instance, Moschovakis's Theorem implies that on finite structures, greatest fixed points (i.e. negations of least fixed points) can be expressed in positive LFP. This gives a normal form for LFP and IFP (see [67]).

Theorem 3.3.51 (Immerman). *On finite structures, every* LFP*-formula (and hence also every* IFP*-formula) is equivalent to a formula in* LFP_1.

This result fails on infinite structures. On infinite structures, there exist LFP formulae that are not equivalent to positive formulae, and in fact the alternation hierarchy of least and greatest fixed points is strict (see [20, 85]).

Exercise 3.3.52. Prove that every IFP-formula is equivalent to one that uses **ifp**-operators only positively. *Hint*: assuming that structures contain at least two elements and that a constant 0 is available, a formula $\neg[\mathbf{ifp} R\overline{x} \,.\, \psi(R, \overline{x})]$ is equivalent to an inflationary induction on a predicate $T\overline{x}\,y$ which, for $y \neq 0$, simulates the induction defined by ψ, checks whether the fixed point has been reached, and then makes atoms $T\overline{x}0$ true if $\overline{x}$ is not contained in the fixed point.

In finite model theory, owing to the Gurevich-Shelah Theorem, the two logics LFP and IFP have often been used interchangeably. However, there are significant differences that are sometimes overlooked. Despite the equivalence of IFP and LFP, inflationary inductions are a more powerful concept than monotone inductions. The translation from IFP-formulae to equivalent LFP-formulae can make the formulae much more complicated, requires an increase in the arity of fixed-point variables and, in the case of infinite structures, introduces alternations between least and greatest fixed points. Therefore it is often more convenient to use inflationary inductions in explicit constructions, the advantage being that one is not restricted to inductions over positive formulae. For an example, see the proof of Theorem 3.5.26 below. Furthermore, IFP is more robust, in the sense that inflationary fixed points remain well defined even when other non-monotone operators (e.g. generalized quantifiers) are added to the language (see, for instance, [35]).

The differences between least and inflationary fixed points are particularly significant in the context of modal logic, i.e. when we compare the modal μ-calculus L_μ with its inflationary counterpart. For instance, L_μ has the finite-model property, the satisfiability problem is decidable (complete for EXPTIME), the model-checking problem is in NP $\cap$ Co-NP (and conjectured by many to be solvable in polynomial time), and there are practical, automata-based techniques for solving the algorithmic problems associated with L_μ. Finally, in terms of expressive power, L_μ can be characterized as the bisimulation-invariant fragment of monadic second-order logic (MSO) [69]. On the other hand, the inflationary counterpart of L_μ, the **model iteration calculus** (MIC) [33], behaves very differently. The finite-model property fails, the satisfiability problem is undecidable (and not even in the arithmetic hierarchy), the model-checking problem is PSPACE-complete, and the expressive power goes beyond monadic second-order logic even on words. The appropriate model-checking games for inflationary fixed-point logics such as IFP and MIC are **backtracking games** [34]. These games are a generalization of parity games with an additional rule allowing players, under certain conditions, to return to an earlier position in the play and revise a choice or to force a countback on the number of moves. This new feature makes backtracking games more powerful so that they can capture inflationary inductions. Accordingly, winning strategies become more complex objects and computationally harder than for parity games.

3.3.9 Partial Fixed-Point Logic

Another fixed-point logic that is relevant to finite structures is the partial fixed-point logic (PFP). Let $\psi(R,\overline{x})$ be an arbitrary formula defining on a finite structure $\mathfrak{A}$ a (not necessarily monotone) operator $F_\psi : R \mapsto \{\overline{a} : \mathfrak{A} \models \psi(R,\overline{a})\}$, and consider the sequence of its finite stages $R^0 := \emptyset$, $R^{m+1} = F_\psi(R^m)$.

This sequence is not necessarily increasing. Nevertheless, as $\mathfrak{A}$ is finite, the sequence either converges to a fixed point, or reaches a cycle with a period greater than one. We define the **partial fixed point** of F_ψ as the fixed point that is reached in the former case, and as the empty relation otherwise. The logic PFP is obtained by adding to first-order logic the **partial-fixed-point formation rule**, which allows us to build from any formula $\psi(R,\overline{x})$ a formula $[\mathbf{pfp}\ R\overline{x}\ .\ \psi(R,\overline{x})](\overline{t})$, saying that $\overline{t}$ is contained in the partial fixed point of the operator F_ψ.

Note that if R occurs only positively in ψ, then

$$[\mathbf{lfp}\ R\overline{x}\ .\ \psi(R,\overline{x})](\overline{t}) \equiv [\mathbf{pfp}\ R\overline{x}\ .\ \psi(R,\overline{x})](\overline{t}),$$

so we have that LFP $\leq$ PFP. However, PFP seems to be much more powerful than LFP. For instance, while a least-fixed-point induction on finite structures always reaches the fixed point in a polynomial number of iterations, a partial-fixed-point induction may need an exponential number of stages.

Example 3.3.53. Consider the sequence of stages R^m defined by the formula

$$\psi(R,x) := \Big(Rx \wedge \exists y(y < x \wedge \neg Ry)\Big) \vee \Big(\neg Rx \wedge \forall y(y < x \rightarrow Ry)\Big) \vee \forall y Ry$$

on a finite linear order $(A, <)$. It is easily seen than the fixed point reached by this induction is the set $R = A$, but before this fixed point is reached, the induction goes in lexicographic order through all possible subsets of A. Hence the fixed point is reached at stage $2^n - 1$, where $n = |A|$.

Simultaneous Inductions.

As in the case of LFP, one can also extend IFP and PFP by simultaneous inductions over several formulae, but again, the simultaneous fixed-point logics S-IFP and S-PFP are not more expressive than their simple variants. However, the proof is a little different than in the case of LFP. It requires that one encodes several relations into one and hence increases the arity of the fixed point variables. As a consequence, it seems to be unknown whether simultaneous *monadic* PFP collapses to simple monadic PFP.

Complexity

Although a PFP induction on a finite structure may go through exponentially many stages (with respect to the cardinality of the structure), each stage can be represented with polynomial storage space. As first-order formulae can be evaluated efficiently, it follows by a simple induction that PFP-formulae can be evaluated in polynomial space.

Proposition 3.3.54. *For every formula* $\psi \in$ PFP*, the set of finite models of* ψ *is in* PSPACE*; in short:* PFP $\subseteq$ PSPACE*.*

On ordered structures, one can use techniques similar to those used in previous capturing results, to simulate polynomial-space-bounded computation by PFP-formulae [2, 99].

Theorem 3.3.55 (Abiteboul, Vianu, and Vardi). *On ordered finite structures,* PFP *captures* PSPACE*.*

Proof. It remains to prove that every class $\mathcal{K}$ of finite ordered structures that is recognizable in PSPACE, can be defined by a PFP-formula.

Let M be a polynomially space-bounded deterministic Turing machine with state set Q and alphabet Σ, recognizing (an encoding of) an ordered structure $(\mathfrak{A}, <)$ if and only if $(\mathfrak{A}, <) \in \mathcal{K}$. Without loss of generality, we can make the following assumptions. For input structures of cardinality n, M requires space less than $n^k - 2$, for some fixed k. For any configuration C of M, let $\text{Next}(C)$ denote its successor configuration. The transition function of M is adjusted so that $\text{Next}(C) = C$ if, and only if, C is an accepting configuration.

We represent any configuration of M with a current state q, tape inscription $w_1 \cdots w_m$, and head position i, by the word $\# w_1 \cdots w_{i-1}(q w_i) w_{i+1} \cdots w_{m-1} \#$ over the alphabet $\Gamma := \Sigma \cup (Q \times \Sigma) \cup \{\#\}$, where $m = n^k$ and $\#$ is merely used as an end marker to make the following description more uniform. When moving from one configuration to the next, Turing machines make only local changes. We can therefore associate with M a function $f : \Gamma^3 \to \Gamma$ such that, for any configuration $C = c_0 \cdots c_m$, the successor configuration $\text{Next}(C) = c'_0 \cdots c'_m$ is determined by the rules

$$c'_0 = c'_m = \# \quad \text{and} \quad c'_i = f(c_{i-1}, c_i, c_{i+1}) \text{ for } 1 \leq i \leq m-1.$$

Recall that we encode structures so that there exist first-order formulae $\beta_\sigma(\overline{y})$ such that $(\mathfrak{A}, <) \models \beta_\sigma(\overline{a})$ if and only the $\overline{a}$th symbol of the input configuration of M for input $code(\mathfrak{A}, <)$ is σ. We now represent any configuration C in the computation of M by a tuple $\overline{C} = (C_\sigma)_{\sigma \in \Gamma}$ of k-ary relations, where

$$C_\sigma := \{\overline{a} : \text{ the } \overline{a}\text{-th symbol of } C \text{ is } \sigma\}.$$

The configuration at time t is the stage $t+1$ of a simultaneous **pfp** induction on $(\mathfrak{A}, <)$, defined by the rules

$$C_\# \overline{y} := \forall z(\overline{y} \leq \overline{z}) \lor \forall \overline{z}(\overline{z} \leq \overline{y})$$

and, for all $\sigma \in \Gamma - \{\#\}$,

$$C_\sigma \overline{y} := \Big(\beta_\sigma(\overline{y}) \land \bigwedge_{\gamma \in \Gamma} \forall \overline{x} \neg C_\gamma \overline{x}\Big) \lor$$
$$\exists \overline{x} \exists \overline{z} \Big(\overline{x} + 1 = \overline{y} \land \overline{y} + 1 = \overline{z} \land \bigvee_{f(\alpha,\beta,\gamma)=\sigma} C_\alpha \overline{x} \land C_\beta \overline{y} \land C_\gamma \overline{z})\Big)$$

The first rule just says that each stage represents a word starting and ending with $\#$. The other rules ensure that (1) if the given sequence $\overline{C}$ contains only empty relations (i.e. if we are at stage 0), then the next stage represents the input configuration, and (2) if the given sequence represents a configuration, then the following stage represents its successor configuration.

By our convention, M accepts its input if and only the sequence of configurations becomes stationary (i.e. reaches a fixed point). Hence M accepts $code(\mathfrak{A}, <)$ if and only if the relations defined by the simultaneous **pfp** induction on $\mathfrak{A}$ of the rules described above are non-empty. Hence $\mathcal{K}$ is PFP-definable. $\square$

An alternative characterization of PSPACE is possible in terms of the database query language *while* consisting essentially of first-order relational updates and while-loops. Vardi [99] proved that *while* captures PSPACE on ordered finite structures and Abiteboul and Vianu proved that *while* and PFP are equivalent on finite structures.

Least Versus Partial Fixed-Point Logic

From the capturing results for PTIME and PSPACE we immediately obtain the result that PTIME = PSPACE if, and only if, LFP = PFP on ordered finite structures. The natural question arises of whether LFP and PFP can be separated on the domain of all finite structures. For a number of logics, separation results on arbitrary finite structures can be established by relatively simple methods, even if the corresponding separation on ordered structures would solve a major open problem in complexity theory. For instance, we have proved by quite a simple argument that DTC $\subsetneq$ TC, and it is also not very difficult to show that TC $\subsetneq$ LFP (indeed, TC is contained in stratified Datalog, which is also strictly contained in LFP; see Sect. 3.3.10). Further, it is trivial that LFP is less expressive than Σ_1^1 on all finite structures. However the situation is different for LFP vs. PFP.

Theorem 3.3.56 (Abiteboul and Vianu). LFP *and* PFP *are equivalent on finite structures if, and only if,* PTIME = PSPACE*.*

3.3.10 Datalog and Stratified Datalog

Datalog and its extensions are a family of rule-based database query languages that extend the conjunctive queries by a relational recursion mechanism similar to the one used in fixed-point logics. Indeed, as we shall see, Datalog can be seen as a fragment of least fixed point logic. For the purpose of this section we simply identify a relational database with a finite relational structure. This is not adequate for all aspects of database theory, but for the questions considered here it is appropriate. For further information on databases, see [1], for example.

Definition 3.3.57. A **Datalog rule** is an expression of the form $H \leftarrow B_1 \wedge \cdots \wedge B_m$, where H, the **head** of the rule, is an atomic formula $R\overline{u}$, and $B_1 \wedge \cdots \wedge B_m$, the **body** of the rule, is a conjunction of literals (i.e. atoms or negated atoms) of the form $S\overline{v}$ or $\neg S\overline{v}$ where $\overline{u}, \overline{v}$ are tuples of variables or constants. The relation symbol R is called the **head predicate** of the rule. We also allow Boolean head predicates. A Datalog rule is **positive** if it does not contain negative literals.

A **Datalog program** Π is a finite collection of rules such that none of its head predicates occurs negated in the body of any rule. The predicates that appear only in the bodies of the rules are called **input predicates**. The **input vocabulary** of Π is the set of input predicates and constants appearing in Π.

Example 3.3.58. The Datalog program Π_{reach} consists of the three rules

$$Txy \leftarrow Exy, \qquad Txz \leftarrow Txy \wedge Tyz, \qquad Ry \leftarrow Tay.$$

The input vocabulary is $\{E, a\}$, and the head predicates are T and R.

Given a structure $\mathfrak{A}$ over the input vocabulary, the program computes an interpretation of the head predicates, i.e. it defines an expansion $\Pi(\mathfrak{A}) := (\mathfrak{A}, R_1, \dots, R_k)$ of $\mathfrak{A}$, where the R_i are the values of the head predicates as computed by Π. This interpretation can be defined in several equivalent ways, for instance via *minimal-model semantics* or *fixed-point semantics*. We can read a Datalog rule $\varphi_r := R\overline{x} \leftarrow B_1 \wedge \dots \wedge B_m$, and associate with the program Π the universal closure of the conjunction over these formulae:

$$\psi[\Pi] := \forall \overline{z} \bigwedge_{\varphi_r \in \Pi} \varphi_r .$$

We can compare expansions of $\mathfrak{A}$ by componentwise inclusion of the additional predicates: $(\mathfrak{A}, R_1, \dots, R_k) \subseteq (\mathfrak{A}, R'_1 \dots, R'_k)$ if $R_i \subseteq R'_i$ for all i. Acording to the **minimal-model semantics**, $\Pi(\mathfrak{A})$ is the minimal expansion $(\mathfrak{A}, R^\mu_1, \dots, R^\mu_k)$ that satisfies $\psi[\Pi]$.

Example 3.3.59. The formula associated with the program Π_{reach} of Example 3.3.58 is

$$\forall x \forall y \forall z ((Txy \leftarrow Exy) \wedge (Txz \leftarrow Txy \wedge Tyz) \wedge (Ry \leftarrow Tay)).$$

The minimal expansion of a graph $G = (V, E)$ with a distinguished node a is $\Pi_{reach}(G, a) = (G, a, T, R)$ where T is the transitive closure of E and R is the set of points reachable by a path from a.

Exercise 3.3.60. Prove that minimal-model semantics is well-defined: for every Datalog program Π and every input database $\mathfrak{A}$, there is a unique minimal expansion of $\mathfrak{A}$ that is a model of $\psi[\Pi]$.

For the case of **fixed-point semantics**, we read a rule $R\overline{x} \leftarrow \beta(\overline{x}, \overline{y})$ as an update operator: whenever an instantiation $\beta(\overline{a}, \overline{b})$ of the body of the rule is true for the current interpretation of the head predicates, make the corresponding instantiation $R\overline{a}$ of the head true. Initially, let all head predicates be empty. At each stage, apply simultaneously the update operators for all rules of the program to the current interpretation of $(R_1, \dots, R_k)$. Iterate this operation until a fixed point $(R^\infty_1, \dots, R^\infty_k)$ is reached. Now let $\Pi(\mathfrak{A}) := (\mathfrak{A}, R^\infty_1, \dots, R^\infty_k)$.

Exercise 3.3.61. Prove that minimal-model semantics and fixed-point semantics coincide: for all Π and $\mathfrak{A}$, $(R^\mu_1, \dots, R^\mu_k) = (R^\infty_1, \dots, R^\infty_k)$.

Definition 3.3.62. A **Datalog query** is a pair (Π, R) consisting of a Datalog program Π and a designated head predicate R of Π. With every structure $\mathfrak{A}$, the query (Π, R) associates the result $(\Pi, R)^{\mathfrak{A}}$, the interpretation of R as computed by Π from the input $\mathfrak{A}$.

We now relate Datalog to LFP. We shall show that each Datalog query (Π, R) is equivalent to a formula $\psi(\overline{x}) \in$ LFP, in fact one of very special form.

Let Π be a Datalog program with input vocabulary τ and head predicates $R_1, \dots, R_k$. We first normalize the rules such that all rules with head predicate R_i have the same head $R_i x_1 \cdots x_{k_i}$. This can be done by appropriate substitutions in the rule body and by adding equalities. For instance, a rule $Rxyyx \leftarrow \beta(x, y, z)$ can be rewritten as $Rx_1x_2x_3x_4 \leftarrow \beta(x_1, x_2, y) \wedge x_3 = x_2 \wedge x_4 = x_1$. We then have a program containing, for each head predicate R_i, rules r_{ij} of the form $R_i\overline{x} \leftarrow \beta_{ij}(\overline{x}, \overline{y})$, where β_{ij} is a conjunction of literals and equalities. We then combine the update operators associated with the same head predicate and describe the update of R_i by the existential first-order formula $\gamma_i(\overline{x}) := \bigvee_j \exists\overline{y}\beta_{ij}(\overline{x}, \overline{y})$. As a consequence, the fixed-point semantics of Π is described by the system

$$S := \begin{cases} R_1\overline{x} := \gamma_1 \\ \qquad \vdots \\ R_k\overline{x} := \gamma_k \end{cases}$$

of first-order update rules, and the query (Π, R_i) is equivalent to the formula $[\mathbf{lfp} R_i : S](\overline{x})$. Hence every Datalog query is equivalent to an LFP-formula, in which fixed-point operators are applied only to existential formulae.

Definition 3.3.63. Existential fixed-point logic, denoted EFP, is the set of (simultaneous) LFP-formulae without universal quantifiers and without **gfp**-operators, and where negations are applied to atomic formulae only.

We have seen that Datalog $\subseteq$ EFP. The converse is also true, which can be established by a straightforward induction: with every formula $\psi \in$ EFP one associates a Datalog program Π_ψ with a distinguished head predicate H_ψ such that the query (Π_ψ, H_ψ) is equivalent to ψ. We leave the details as an exercise.

Proposition 3.3.64. *Datalog is equivalent to* EFP.

We know that LFP captures PTIME on ordered finite structures. The question arises of whether Datalog is sufficiently powerful to do the same. The answer depends on the precise variant of Datalog and on the notion of ordered structures that is used. We distinguish three cases.

(1) A simple monotonicity argument shows that Datalog is weaker than PTIME on structures where only a linear order, but not a successor relation, is given. If $\mathfrak{A}$ is a substructure of $\mathfrak{B}$, then $(\Pi, R)^{\mathfrak{A}} \subseteq (\Pi, R)^{\mathfrak{B}}$ for every Datalog query (Π, R). Of course, there even exist very simple *first-order* queries that are not monotone in this sense. Note that this argument does break down on databases where a successor relation S (rather than just a linear order), and constants 0 and e for the first and last elements are given. Exercise: why?

(2) In the literature, Datalog programs are often required to contain only positive rules, i.e. the input predicates also can be used only positively. This restricted variant is too weak to capture PTIME, even on successor structures.

If input predicates can be used only positively, then queries are monotone under extensions of the input relations: if a database $\mathfrak{B}$ is obtained from $\mathfrak{A}$ by augmenting some of the input relations, then again $(\Pi, R)^{\mathfrak{A}} \subseteq (\Pi, R)^{\mathfrak{B}}$.

Exercise 3.3.65. Prove this monotonocity property, and give examples of first-order queries that cannot be defined by Datalog programs.

(3) In the case of programs with negations of input predicates and databases with a successor relation and constants 0 and e for the first and last elements, we can capture PTIME by Datalog. This was originally established in [13, 91] and is implicit also in [67].

Theorem 3.3.66 (Blass, Gurevich, and Papadimitriou). *On successor structures, Datalog (with negations of input predicates) captures* PTIME*.*

Proof. This result can be established in several ways, for instance by a reduction from Σ^1_1-HORN (making use of the fact that PTIME is closed under complement). Instead, we give a direct proof.

It is clear that Datalog queries are computable in polynomial time. It remains to prove that every class $\mathcal{K}$ of finite successor structures that is recognizable in PTIME can be defined by a Boolean Datalog query.

Let M be a polynomial-time Turing machine with state set Q and alphabet Σ, recognizing (an encoding of) a successor structure $\mathfrak{A}$ if and only if $\mathfrak{A} \in \mathcal{K}$. We denote the cardinality of the input structure $\mathfrak{A}$ by n and assume that the computation time of M on $\mathfrak{A}$ is less than n^k.

The construction is similar to the proof of Theorem 3.3.55. Configurations of M are represented by words $\#w_1 \cdots w_{i-1}(qw_i)w_{i+1} \cdots w_{m-1}\#$ over the alphabet $\Gamma := \Sigma \cup (Q \times \Sigma) \cup \{\#\}$, where $m = n^k$, and we describe the behaviour of M by a function $f : \Gamma^3 \to \Gamma$ such that, for any configuration $C = c_0 \cdots c_m$, the successor configuration $\mathrm{Next}(C) = c'_0 \cdots c'_m$ is determined by the rules

$$c'_0 = c'_m = \# \quad \text{and} \quad c'_i = f(c_{i-1}, c_i, c_{i+1}) \text{ for } 1 \le i \le m-1.$$

Let S be a $2k$-ary relation symbol and let Π_S be a Datalog program with head predicate S, computing the successor relation on k-tuples (associated with the lexicographic order defined by the given successor relation). Recall that we can encode successor structures so that there exist quantifier-free formulae $\beta_\sigma(\overline{y})$ such that $\mathfrak{A} \models \beta_\sigma(\overline{a})$ if, and only if, the $\overline{a}$th symbol of the input configuration of M for $code(\mathfrak{A})$ is σ. Let (Π_σ, H_σ) be a Datalog query equivalent to $\beta_\sigma(\overline{y})$.

We represent the computation of M by a tuple $\overline{C} = (C_\sigma)_{\sigma \in \Gamma}$ of $2k$-ary relations, where

$$C_\sigma := \{(\overline{a}, \overline{t}) : \text{ the } \overline{a}\text{th symbol of the configuration at time } \overline{t} \text{ is } \sigma\}.$$

The Datalog program associated with M consists of

(1) the program Π_S defining the successor relation on k-tuples;
(2) the programs Π_σ for describing the input;
(3) the rules

$$\begin{aligned}&C_{\#}\overline{0}\,\overline{t}\\&C_{\#}\overline{e}\,\overline{t}\\&C_\sigma\overline{y}\,\overline{0} \leftarrow H_\sigma\overline{y} \text{ for all } \sigma \in \Gamma - \{\#\};\end{aligned}$$

(4) for all $\alpha, \beta, \gamma, \sigma$ with $f(\alpha, \beta, \gamma) = \sigma$, the rule

$$C_\sigma\overline{y}\,\overline{t}' \leftarrow S\overline{x}\,\overline{y} \wedge S\overline{y}\,\overline{z} \wedge S\overline{t}\,\overline{t}' \wedge C_\alpha\overline{x}\overline{t} \wedge C_\beta\overline{y}\,\overline{t} \wedge C_\gamma\overline{z}\,\overline{t};$$

(5) the rule

$$\text{Acc} \leftarrow C_{qw}\overline{x}\,\overline{t} \quad \text{for any } \textit{accepting} \text{ state } q \text{ and any symbol } w.$$

The first two rules in (3) say that each configuration starts and ends with #; the following set of rules ensures that the configuration at time 0 is the input configuration. The rules in (4) imply that from time $\overline{t}$ to time $\overline{t}' = \overline{t} + 1$ the computation proceeds as required by M, and the last rule makes the Boolean predicate Acc true if and only if an accepting state has been reached. Obviously, M accepts the input structure $\mathfrak{A}$ if, and only if, the query (Π_M, Acc) evaluates to true on $\mathfrak{A}$. □

Almost the same proof shows that the expression complexity of Datalog (and hence of LFP) is EXPTIME-complete (see also Theorem 3.3.36).

Theorem 3.3.67. *The evaluation problem for Datalog programs (with head predicates of unbounded arity) is complete for* EXPTIME*, even for programs with only positive rules, and for a fixed database with only two elements.*

Proof. By the results of Section 3.3.5, LFP-formulae, and hence also Datalog programs, can be evaluated in polynomial time with respect to the size of the input structure and in exponential time with respect to the length of the formula (or program).

To prove completeness, we fix a database $\mathfrak{A}$ with two elements and constant symbols 0, 1 (or, alternatively, two unary relations $P_0 = \{0\}$ and $P_1 = \{1\}$). Let M be a deterministic Turing machine that accepts or rejects input words $w = w_0 \cdots w_{m-1} \in \{0,1\}^*$ in time 2^{m^d} (for some fixed d). For every input x for M, we construct a Datalog program $\Pi_{M,w}$ which evaluates, on the fixed database $\mathfrak{A}$, a Boolean head predicate Acc to true if, and only if, M accepts w.

The construction is similar to that in the proof of Theorem 3.3.66, with the following two differences. Whereas in the previous proof k was fixed and n depended on the input, it is now the other way round, with $n := 2$ and $k := m^d$. Further, the description of the input configuration is now simpler:

we just explicitly list the atomic facts defining the input configuration for the given input w. Note that this is the only part of the program that depends on w; the remaining rules depend only on M and the *length* of the input. Finally note that the program contains only positive rules. □

Stratified Datalog

Datalog defines in a natural way queries that require recursion (such as transitive closure), but is very weak in other respects, mainly because it does not include negation.

There exist various possible ways to add negation to Datalog.

Definition 3.3.68. A **stratified Datalog program** is a sequence $\Pi = (\Pi_0, \dots, \Pi_r)$ of basic Datalog programs, which are called the **strata** of Π, such that each of the head predicates of Π is a head predicate in precisely one stratum Π_i and is used as an input predicate only in higher strata Π_j, where $j > i$. In particular, this means that

(1) if a head predicate of stratum Π_j occurs *positively* in the body of a rule of stratum Π_i, then $j \leq i$, and
(2) if a head predicate of stratum Π_j occurs *negatively* in the body of a rule of stratum Π_i, then $j < i$.

The semantics of a stratified program is defined stratum by stratum. The input predicates of a stratum Π_i are either input predicates of the entire program Π or are head predicates of a lower stratum. Hence, once the lower strata are evaluated, we can compute the interpretation of the head predicates of Π_i as in the case of basic Datalog programs.

Clearly the power of stratified Datalog is between that of Datalog and LFP, and hence stratified Datalog provides yet another formalism that captures PTIME on ordered structures. On unordered structures stratified Datalog is strictly more expressive than Datalog (as it includes all of first-order logic) but strictly less powerful than LFP. The main example separating LFP from Stratified Datalog is the GAME query, which defines the winning positions of Player 0 in a strictly alternating game. It is defined by the LFP formula $[\mathbf{lfp} Wx \,.\, \exists y(Exy \wedge \forall z(Eyz \rightarrow Wz)](x)$. This involves a recursion through a universal quantifier, which in general cannot be done in stratified Datalog [31, 79].

Theorem 3.3.69 (Dahlhaus and Kolaitis). *No stratified Datalog program can express the* GAME *query. Hence* stratified Datalog $\subsetneq$ LFP*.*

Example 3.3.70. Another interesting class of examples showing the limits of stratified Datalog is that of well-foundedness properties, or statements saying that on all infinite paths one will eventually hit a node with a certain property

P. These are typical statements in the field of verification (expressed in CTL by the formula $\mathbf{AF}P$).

In LFP, the well-foundedness of a partial order $\prec$ would be expressed as $\forall y[\mathbf{lfp} Wy \,.\, \forall x(x \prec y \rightarrow Wx)](y)$. The CTL-formula $\mathbf{AF}P$ is expressed in L_μ by $\mu X.P \vee \Box X$ and in LFP by $[\mathbf{lfp} Rx \,.\, Px \vee \forall y(Exy \rightarrow Ry)](x)$.

On *finite* structures, such properties are definable by stratified Datalog programs, since they are essentially negations of reachability problems for cycles. Indeed, $\mathbf{AF}P$ means that there is no path that eventually cycles and on which P is globally false. This can be expressed by the following stratified program:

$$\begin{array}{ll}
Txy \leftarrow \neg Px \wedge Exy \wedge \neg Py & \quad Txz \leftarrow Txy \wedge Eyz \wedge \neg Pz \\
Sx \leftarrow Txx & \quad Sx \leftarrow \neg Px \wedge Exy \wedge Sy \\
Rx \leftarrow \neg Sx &
\end{array}$$

The first stratum computes the set T of all pairs of nodes (u, v) such that there exists a path from u to v on which P is false, and the set S of all nodes from which there exists such a path that eventually cycles. Here the finiteness of the graph is used in an essential way, because only this guarantees that every infinite path eventually reaches a cycle. The second stratum takes the complement of S.

However, it can be shown that no stratified Datalog program can express such statements on *infinite* structures (even countable ones).

Another variant of Datalog, called Datalog LITE, which can express all CTL properties and moreover admits linear-time evaluation algorithms (and which is incomparable with stratified Datalog), has been defined and studied in [45].

A stratified Datalog program is **linear** if in the body of each rule there is at most one occurrence of a head predicate of the same stratum (but there may be arbitrary many occurrences of head predicates from lower strata).

Example 3.3.71. The program Π_{reach} in Example 3.3.58 is not linear, but by replacing the second, non-linear rule $Txz \leftarrow Txy \wedge Tyz$ by the linear rule $Txz \leftarrow Txy \wedge Eyz$ we obtain an equivalent linear program. However, one pays a price for the linearization. The original program reaches the fixed point after $O(\log m)$ iterations, while the linear program needs m iterations, where m is the length of the longest path in the graph.

Linear programs suffice to define transitive closures, so it follows by a straightforward induction that TC $\subseteq$ linear stratified Datalog. The converse is also true (see [38, 46]).

Proposition 3.3.72. *Linear stratified Datalog is equivalent to* TC.

Corollary 3.3.73. *On ordered structures, linear stratified Datalog captures* NLOGSPACE.

3.4 Logics with Counting

From the point of view of expressiveness, first-order logic has two main deficiencies: it lacks the power to express anything that requires recursion (the simplest example is transitive closure) and it cannot count, as witnessed by the impossibility to express that a structure has even cardinality, or, more generally, by the 0-1 law. We have already discussed a number of logics that add recursion in one way or another to FO (or part of it), notably the various forms of fixed-point logic. On ordered finite structures, some of these logics can express precisely the queries that are computable in PTIME or PSPACE. However, on arbitrary finite structures they do not, and almost all known examples showing this involve counting. Whereas in the presence of an ordering, the ability to count is inherent in fixed-point logic, hardly any of this ability is retained in its absence. For instance, as LFP and PFP are fragments of $L^{\omega}_{\infty\omega}$, the 0-1 law also holds for them.

Therefore Immerman proposed that counting quantifiers should be added to logics and asked whether a suitable variant of fixed-point logic with counting would suffice to capture PTIME. Although Cai, Fürer and Immerman [23] eventually answered this question negatively, fixed-point logic with counting has turned out to be an important and robust logic, that defines a natural level of expressiveness and allows one to capture PTIME on interesting classes of structures.

3.4.1 Logics with Counting Terms

There are different ways of adding counting mechanisms to a logic, which are not necessarily equivalent. The most straightforward possibility is the addition of quantifiers of the form $\exists^{\geq 2}$, $\exists^{\geq 3}$, etc., with the obvious meaning. While this is perfectly reasonable for bounded-variable fragments of first-order logic or infinitary logic (see e.g. [58, 89]), it is not general enough for fixed-point logic, because it does not allow for recursion over the counting parameters i in quantifiers $\exists^{\geq i}x$. In fact, if the counting parameters are fixed numbers, then adjoining the quantifiers $\exists^{\geq i}x$ does not give additional power to logics such as FO or LFP, since they are closed under the replacement of $\exists^{\geq i}$ by i existential quantifiers (where as their restrictions to bounded width are not). These counting parameters should therefore be considered as variables that range over natural numbers. To define in a precise way a logic with counting and recursion, one extends the original objects of study, namely finite (one-sorted) structures $\mathfrak{A}$, to two-sorted auxiliary structures $\mathfrak{A}^*$ with a second numerical (but also finite) sort.

Definition 3.4.1. With any one-sorted finite structure $\mathfrak{A}$ with universe A, we associate the two-sorted structure $\mathfrak{A}^* := \mathfrak{A} \,\dot\cup\, \langle\{0,\dots,|A|\}; \leq, 0, e\rangle$, where $\leq$ is the canonical ordering on $\{0,\dots,|A|\}$, and 0 and e stand for the first and the last element. Thus, we have taken the disjoint union of $\mathfrak{A}$ with a linear order of length $|A|+1$.

We start with first-order logic over two-sorted vocabularies $\sigma \cup \{\leq, 0, e\}$, with semantics over structures $\mathfrak{A}^*$ defined in the obvious way. We shall use Latin letters $x, y, z, \dots$ for the variables over the first sort, and Greek letters $\lambda, \mu, \nu, \dots$ for variables over the second sort. The two sorts are related by **counting terms**, defined by the following rule. Let $\varphi(x)$ be a formula with a variable x (over the first sort) among its free variables. Then $\#_x[\varphi]$ is a term in the second sort, with the set of free variables $\text{free}(\#_x[\varphi]) = \text{free}(\varphi) - \{x\}$. The value of $\#_x[\varphi]$ is the number of elements a that satisfy $\varphi(a)$.

Counting logics of this form were introduced by Grädel and Otto [54] and have been studied in detail in [89]. We start with first-order logic with counting, denoted by (FO + C), which is the closure of two-sorted first-order logic under counting terms. Here are two simple examples that illustrate the use of counting terms.

Example 3.4.2. On a undirected graph $G = (V, E)$, the formula $\forall x \forall y (\#_z[Exz] = \#_z[Eyz])$ expresses the assertion that every node has the same degree, i.e., that G is regular.

Example 3.4.3. We present below a formula $\psi(E_1, E_2) \in$ (FO + C) which expresses the assertion that two equivalence relations E_1 and E_2 are isomorphic; of course a necessary and sufficient condition for this is that for every i, they have the same number of elements in equivalence classes of size i:

$$\psi(E_1, E_2) \equiv (\forall \mu)(\#_x[\#_y[E_1xy] = \mu] = \#_x[\#_y[E_2xy] = \mu]).$$

3.4.2 Fixed-Point Logic with Counting

We now define **(inflationary) fixed point logic with counting** (IFP + C) and **partial fixed point logic with counting** (PFP + C) by adding to (FO + C) the usual rules for building inflationary or partial fixed points, ranging over both sorts.

Definition 3.4.4. Inflationary fixed point logic with counting, (IFP + C), is the closure of two-sorted first-order logic under the following rules:

(1) The rule for building counting terms.
(2) The usual rules of first-order logic for building terms and formulae.
(3) The fixed-point formation rule. Suppose that $\psi(R, \overline{x}, \overline{\mu})$ is a formula of vocabulary $\tau \cup \{R\}$ where $\overline{x} = x_1, \dots, x_k$, $\overline{\mu} = \mu_1, \dots, \mu_\ell$, and R has mixed arity (k, ℓ), and that $(\overline{u}, \overline{\nu})$ is a $k + \ell$-tuple of first- and second-sort terms, respectively. Then

$$[\textbf{ifp}\ R\overline{x}\overline{\mu} \,.\, \psi](\overline{u}, \overline{\nu})$$

is a formula of vocabulary τ.

The semantics of [**ifp** $R\overline{x}\overline{\mu}$. ψ] on $\mathfrak{A}^*$ is defined in the same way as for the logic IFP, namely as the inflationary fixed point of the operator

$$F_\psi : R \longmapsto R \cup \{(\overline{a}, \overline{i}) \mid (\mathfrak{A}^*, R) \models \psi(\overline{a}, \overline{i})\}.$$

The definition of (PFP + C) is analogous, where we replace inflationary fixed points by partial ones. In the literature, one also finds different variants of fixed-point logic with counting where the two sorts are related by *counting quantifiers* rather than counting terms. Counting quantifiers have the form $(\exists i\ x)$ for 'there exist at least i x', where i is a second-sort variable. It is obvious that the two definitions are equivalent. In fact, (IFP + C) is a very robust logic. For instance, its expressive power does not change if one permits counting over tuples, even of mixed type, i.e. terms of the form $\#_{\overline{x},\overline{\mu}}\varphi$. One can of course also define least fixed-point logic with counting, (LFP + C), but one has to be careful with the positivity requirement (which is more natural when one uses counting quantifiers rather than counting terms). The equivalence of LFP and IFP readily translates to (LFP + C) $\equiv$ (IFP + C). Further, there are a number of other logical formalizations of the concept of inductive definability with counting that turn out to have the same expressive power as (IFP + C) (see [54] and Sect.3.4.3 below for details).

Example 3.4.5. An interesting example of an (IFP + C)-definable query is the method of *stable colourings* for graph-canonization. Given a graph G with a colouring $f : V \to 0, \dots, r$ of its vertices, we define a refinement f' of f, giving to a vertex x the new colour $f'x = (fx, n_1, \dots, n_r)$ where $n_i = \#y[Exy \wedge (fy = i)]$. The new colours can be sorted lexicographically so that they again form an initial subset of $\mathbb{N}$. Then the process can be iterated until a fixed point, the *stable colouring* of G is reached. It is easy to see that the stable colouring of a graph is polynomial-time computable and uniformly definable in (IFP + C).

On many graphs, the stable colouring uniquely identifies each vertex, i.e. no two distinct vertices get the same stable colour. This is the case, for instance, for all trees. Further, Babai, Erdös, and Selkow [8] proved that the probability that this happens on a random graph with n nodes approaches 1 as n goes to infinity. Thus stable colourings provide a polynomial-time graph canonization algorithm for almost all finite graphs.

We now discuss the expressive power and evaluation complexity of fixed-point logic with counting. We are mainly interested in (IFP + C)-formulae and (PFP + C)-formulae without free variables over the second sort, so that we can compare them with the usual logics without counting.

Exercise 3.4.6. Even without making use of counting terms, IFP over two-sorted structures $\mathfrak{A}^*$ is more expressive than IFP over $\mathfrak{A}$. To prove this, construct a two-sorted IFP-sentence ψ such that $\mathfrak{A}^* \models \psi$ if, and only if, $|A|$ is even.

It is clear that counting terms can be computed in polynomial-time. Hence the data complexity remains in PTIME for (IFP + C) and in PSPACE for (PFP + C). We shall see below that these inclusions are strict.

Theorem 3.4.7. *On finite structures,*

(1) IFP $\subsetneq$ (IFP + C) $\subsetneq$ PTIME*.*
(2) PFP $\subsetneq$ (PFP + C) $\subsetneq$ PSPACE*.*

Infinitary Logic with Counting

Let $C^k_{\infty\omega}$ be the infinitary logic with k variables $L^k_{\infty\omega}$, extended by the quantifiers $\exists^{\geq m}$ ('there exist at least m') for all $m \in \mathbb{N}$. Further, let $C^\omega_{\infty\omega} := \bigcup_k C^k_{\infty\omega}$.

Proposition 3.4.8. (IFP + C) $\subseteq C^\omega_{\infty\omega}$.

Due to the two-sorted framework, the proof of this result is a bit more involved than for the corresponding result without counting, but not really difficult. We refer to [54, 89] for details.

The separation of (IFP + C) from PTIME has been established by Cai, Fürer, and Immerman [23]. The proof also provides an analysis of the method of stable colourings for graph canonization. We have deswcribed this method in its simplest form in Example 3.4.5. More sophisticated variants compute and refine colourings of k-tuples of vertices. This is called the *k-dimensional Weisfeiler–Lehman method* and, in logical terms, it amounts to labelling each k-tuple by its type in $k + 1$-variable logic with counting quantifiers. It was conjectured that this method could provide a polynomial-time algorithm for graph isomorphism, at least for graphs of bounded degree. However, Cai, Fürer, and Immerman were able to construct two families $(G_n)_{n\in\mathbb{N}}$ and $(H_n)_{n\in\mathbb{N}}$ of graphs such that on one hand, G_n and H_n have $O(n)$ nodes and degree three, and admit a linear-time canonization algorithm, but on the other hand, in first-order (or infinitary) logic with counting, $\Omega(n)$ variables are necessary to distinguish between G_n and H_n. In particular, this implies Theorem 3.4.7.

Inflationary vs. Partial Fixed-Points

By Theorem 3.3.56, partial fixed-point logic collapses to inflationary fixed-point logic if, and only if, PTIME = PSPACE. The analogous result in the presence of counting is also true [54, 89]: PTIME = PSPACE $\Longleftrightarrow$ (IFP + C) = (PFP + C).

3.4.3 Datalog with Counting

Fixed-point formulae have the reputation of being difficult to read, and many people find formalisms such as Datalog easier to understand. In the presence of a successor relation, Datalog (with negation over input predicates) is sufficient to capture PTIME and hence is equally expressive as LFP. In general, however, Datalog and even its most natural extensions, notably stratified Datalog, are weaker than LFP.

Counting terms can also be added to Datalog. We conclude this section by discussing Datalog with counting. We show that (Datalog + C) is closed under negation and equivalent to (IFP + C). In the presence of counting, the common extensions of Datalog, notably stratified Datalog, are therefore equivalent to Datalog.

Definition 3.4.9. Datalog with counting, denoted by (Datalog + C), extends Datalog by allowing two-sorted head predicates and counting terms. The two-sorted head atoms have the form $R\overline{x}\overline{\mu}$, where $\overline{x}$ ranges over the first sort, i.e. over elements of the input database $\mathfrak{A}$, and $\overline{\mu}$ ranges over the second sort. For any atom $Rx\overline{y}\overline{\mu}$ we have a counting term $\#_x[Rx\overline{y}\overline{\mu}]$. A term over the second sort is called an **arithmetical term**. The arithmetical terms are either 0, e, counting terms, or $t+1$, where t is also an arithmetical term. Thus, a program in (Datalog + C) is a finite set of clauses of the form

$$H \leftarrow B_1 \wedge \cdots \wedge B_m$$

where the head H is an atomic formula $R(\overline{x}, \overline{\mu})$, and $B_1, \ldots, B_m$ are atomic formulae $R\overline{x}\overline{\mu}$ or equalities of terms (over the first or the second sort).

For every input database, the program computes intensional relations via the inflationary fixed-point semantics. Note that for classical Datalog programs, it makes no difference whether the fixed-point semantics is defined to be inflationary or not, since the underlying operator is monotone anyway. However, for programs in (Datalog + C), the semantics has to be inflationary, since otherwise, the equalities of arithmetical terms give rise to non-monotone operators. For the same reason, the minimum-model semantics will no longer be defined. Since inflationary fixed-point semantics is one of the various equivalent ways to define the semantics of Datalog, both the syntax and the semantics of (Datalog + C) generalize Datalog in a natural way.

One could also introduce counting in an (at first sight) more general form, namely by allowing counting terms of the form $\#_{\overline{x},\overline{\mu}}[R\overline{x}\overline{\mu}\overline{y}\overline{\nu}]$. While this may be convenient for writing a program in shorter and more understandable form, it does not affect the power of (Datalog + C).

Exercise 3.4.10. [54] Prove that counting over tuples, even of mixed type, does not increase the expressive power of (Datalog + C).

Hence cardinalities of arbitrary predicates can be equated in a Datalog program: we take the liberty of writing equalities such as $|Q| = |R|$ in the body of a rule, for simplicity. The following technical lemma is essential for reducing (IFP + C) to (Datalog + C).

Lemma 3.4.11. *Let Π be a* (Datalog + C) *program with head predicates $Q_1, \dots, Q_r$. There exists another* (Datalog + C) *program Π', whose head predicates include $Q_1, \dots, Q_r$ and a Boolean control predicate C^* such that*

- *$(\Pi', Q_i) = (\Pi, Q_i)$ for all i;*
- *(Π', C^*) is true on all databases and C^* becomes true only at the last stage of the evaluation of Π'.*

Proof. In addition to C^*, we add a unary head predicate C^0 and, for every head predicate Q_i of Π a new head predicate Q'_i of the same arity. Then, Π' is obtained by adding the following clauses to Π:

$$\begin{aligned} &C^0 x \\ Q'_i\overline{x\mu} &\leftarrow Q_i\overline{x\mu} \quad \text{for } 1 \le i \le r \\ C^* &\leftarrow C^0 x \wedge (|Q_1| = |Q'_1|) \wedge \cdots \wedge (|Q_r| = |Q'_r|) \end{aligned}$$

Observe that Q'_i simply lags one step behind Q_i. The atom $C^0 x$ is necessary to avoid the possibility that C^* is set to true in the first stage. □

Lemma 3.4.11 essentially says that we can attach to any program a Boolean control predicate which becomes true when the evaluation of the program is terminated. We can then compose two Datalog programs while making sure that the evaluation of the second program starts only after the first has been terminated. As an initial application, we shall show that (Datalog + C) is closed under negation.

Lemma 3.4.12. *The complement of a* (Datalog + C) *query is also a* (Datalog + C) *query.*

Proof. Let (Π, Q) be a (Datalog + C) query, and let Π' be the program specified in Lemma 3.4.11. Take a new variable z, and new head predicates $\tilde{Q}$ and R with arity(R) = arity(Q) and arity$(\tilde{Q})$ = arity$(Q) + 1$. Construct Π'' by adding to Π' the rules

$$\begin{aligned} \tilde{Q}\overline{x\mu}z &\leftarrow Q\overline{x\mu} \\ R\overline{x\mu} &\leftarrow C^* \wedge (\#_z[\tilde{Q}\overline{x\mu}z] = 0). \end{aligned}$$

The query (Π'', R) is the complement of (Π, Q). □

Difficulties in expressing negation are the reason why, in the absence of counting (or of an ordering), Datalog is weaker than fixed-point logic. Also, the limited form of negation that is available in Stratified Datalog (which does not allow for 'recursion through negation') does not suffice to express all fixed-point queries. (Datalog + C) does not have these limitations, and is equally expressive as (IFP + C).

Theorem 3.4.13 (Grädel and Otto). (Datalog + C) $\equiv$ (IFP + C).

It is obvious that (Datalog + C) $\subseteq$ (IFP + C). For the converse, we can construct by induction, for every formula $\psi \in$ (IFP + C), a (Datalog + C) program Π_ψ with goal predicate Q_ψ such that (Π_ψ, Q_ψ) is equivalent to ψ.

Exercise 3.4.14. For atomic formulae, disjunctions, and existential quantification the construction is obvious, and closure under negation has already been proved. Complete the proof for applications of counting terms, i.e formulae $\psi(\overline{y}, \overline{\mu}, \nu) := \#_x[\varphi(\overline{x}, \overline{y}, \overline{\mu})] = \nu$, and fixed point formulae $\psi := [\mathbf{ifp} R\overline{x\mu} \,.\, \varphi(R, \overline{x}, \overline{\mu})](\overline{y}, \overline{\nu})$. The construction makes use of Lemma 3.4.11.

Example 3.4.15. To illustrate the expressive power of (Datalog + C) we show below a program for the GAME query (for strictly alternating games). The GAME query is the canonical example that separates LFP from Stratified Datalog [31, 79]. GAME is definable in fixed-point logic, by the formula $[\mathbf{lfp} Wx \,.\, \exists y(Exy \land \forall z(Eyz \rightarrow Wz))](x)$ that defines the winning positions for Player 0.

Here is a (Datalog + C) program with goal predicate Z, defining GAME:

$$\begin{aligned} Wx\lambda &\leftarrow Exy \land Vy\mu \land \lambda = \mu + 1 \\ Fyz\mu &\leftarrow Eyz \land Wz\mu \\ Vy\mu &\leftarrow \#_z[Eyz] = \#_z[Fyz\mu] \\ Zx &\leftarrow Wx\mu \end{aligned}$$

The evaluation of this program on a game graph G assigns to W (or V) a set of pairs $(x, \mu) \in V \times \mathbb{N}$, such that Player 0 has a winning strategy from position x in at most μ moves when she (or Player 1, respectively) begins the game.

3.5 Capturing PTIME via Canonization

We have seen that there are a number of logics that capture polynomial time on ordered finite structures, but none of them suffices to express all of PTIME in the absence of a linear order. Indeed, it has been conjectured that no logic whatsoever can capture PTIME on the domain of all finite structures. We shall discuss this problem further at the end of this section. But, of course, even if this conjecture should turn out to be true, it remains an important issue to capture PTIME on other relevant domains besides ordered structures.

3.5.1 Definable Linear Orders

An obvious approach is to try to *define* linear orders and then apply the known results for capturing complexity classes on ordered structures.

Definition 3.5.1. Let $\mathcal{D}$ be a domain of finite structures and let L a logic. We say that $\mathcal{D}$ admits ***L*-definable linear orders** if, for every vocabulary τ, there exists a formula $\psi(x, y, \bar{z}) \in L(\tau)$ such that there exists in every structure $\mathfrak{A} \in \mathcal{D}(\tau)$ a tuple $\bar{c}$ for which the relation $\{(a, b) : \mathfrak{A} \models \psi(a, b, \bar{c})\}$ is a linear order on A. The elements in $\bar{c}$ are called the parameters of the order defined by ψ on $\mathfrak{A}$.

Example 3.5.2. Let $\mathcal{D}$ consist of all structures $(A, E, R_1, \ldots,)$ such that (A, E) is an undirected cycle. $\mathcal{D}$ admits LFP-definable linear orders (with two parameters), via the formula

$$\psi(x, y, z_1, z_2) := Ez_1z_2 \land [\mathbf{lfp} Rxy \,.\, (x = z_1 \land y = z_2) \lor \exists u(Rxu \land Euy \land y \neq z_1) \\ \lor \exists u(Ruy \land Eux \land x \neq y](x, y).$$

Furthermore, straightforward automorphism arguments show that we cannot define linear orders with fewer than two parameters.

Exercise 3.5.3. Let $\mathcal{D}$ be the domain of structures $(A, E, R_1, \ldots,)$ such that (A, E) is isomorphic to a finite rectangular grid. Show that $\mathcal{D}$ admits LFP-definable linear orders.

Exercise 3.5.4. Let $\mathcal{K}$ be a class of τ-structures with the following property. For every $m \in \mathbb{N}$, there exists a structure $\mathfrak{A} \in \mathcal{K}$ such that for every m-tuple $\bar{a}$ in $\mathfrak{A}$ there exists a non-trivial automorphism of $\mathfrak{A}, \bar{a}$. Then $\mathcal{K}$ does not admit definable orders in any logic.

On any domain that admits LFP-definable linear orders, we can capture PTIME by using LFP-formulae that express polynomial-time properties on ordered structures, and modify them appropriately.

Proposition 3.5.5. *If $\mathcal{D}$ admits* LFP*-definable linear orders, then* LFP *captures polynomial time on $\mathcal{D}$.*

Proof. It only remains to show that every polynomial-time model class $\mathcal{K} \subseteq \mathcal{D}(\tau)$ is L-definable. Let $\varphi(x, y, \bar{z})$ be a formula defining a linear order on the structures in $\mathcal{D}(\tau)$. As LFP captures PTIME on ordered structures, there exists a formula $\psi \in \mathrm{LFP}(\tau \cup \{<\})$ such that, for every structure $\mathfrak{A} \in \mathcal{D}(\tau)$ and every linear order $<$ on A, we have that $(\mathfrak{A}, <) \models \psi$ iff $\mathfrak{A} \in \mathcal{K}$. It follows that

$$\mathfrak{A} \in \mathcal{K} \iff \mathfrak{A} \models \exists \bar{z} \Big(\text{“}\{(x, y) : \varphi(x, y, \bar{z})\} \text{ is a linear order”} \land \\ \psi[u < v / \varphi(u, v, \bar{z})] \Big),$$

where $\psi[u < v/\varphi(u, v, \bar{z})]$ is the formula obtained from ψ by replacing every atom of the form $u < v$ by $\varphi(u, v, \bar{z})$. □

3.5.2 Canonizations and Interpretations

Let S be any set and let $\sim$ be an equivalence relation on S. A *canonization function* for $(S, \sim)$ is a function $f : S \to S$ associating with every element a canonical member of its equivalence class. That means that $f(s) \sim s$ for all $s \in S$, and $f(s) = f(s')$ whenever $s \sim s'$.

In finite model theory, we are interested in canonization algorithms for finite structures, either up to isomorphism or up to a coarser equivalence relation, such as indistinguishability in some logic or bisimulation. As algorithms take encodings of structures as inputs, and as the encoding of a structure is determined by an ordering of its universe, we can view canonization of structures as an operation that associates with every structure $\mathfrak{A}$ an ordered one, say $(\mathfrak{A}', <)$, such that $\mathfrak{A}'$ is equivalent to $\mathfrak{A}$, and such that equivalent structures are mapped to the same ordered structure (and hence the same encoding).

For a class $\mathcal{K}$ of structures, we write $\mathcal{K}^<$ for the class of expansions $(\mathfrak{A}, <)$ of structures $\mathfrak{A} \in \mathcal{K}$ by some linear order.

Definition 3.5.6. Let $\mathcal{K}$ be a class of finite τ-structures, and let $\sim$ be an equivalence relation on $\mathcal{K}$. A **canonization function** for $\sim$ on $\mathcal{K}$ is a function $f : \mathcal{K} \to \mathcal{K}^<$ that associates with every structure $\mathfrak{A} \in \mathcal{K}$ an ordered structure $f(\mathfrak{A}) = (\mathfrak{A}', <)$ with $\mathfrak{A}' \sim \mathfrak{A}$, such that $f(\mathfrak{A}) \cong f(\mathfrak{B})$ whenever $\mathfrak{A} \sim \mathfrak{B}$.

Interpretations

We are especially interested in canonizations that are defined by interpretations. The notion of an interpretation is very important in mathematical logic, and for model theory in particular. Interpretations are used to define a copy of a structure inside another one, and thus permit us to transfer definability, decidability, and complexity results between theories.

Definition 3.5.7. Let L be a logic, let σ, τ be vocabularies, where $\tau = \{R_1, \dots, R_m\}$ is relational, and let r_i be the arity of R_i. A **(one-dimensional) $L[\sigma, \tau]$-interpretation** is given by a sequence I of formulae in $L(\sigma)$ consisting of

- $\delta(x)$, called the domain formula,
- $\varepsilon(x, y)$, called the equality formula, and,
- for every relation symbol $R \in \tau$ (of arity r), a formula $\varphi_R(x_1, \dots, x_r)$.

An $L[\sigma, \tau]$-interpretation induces two mappings, one between structures, and the other between formulae. For a τ-structure $\mathfrak{A}$ and a σ-structure $\mathfrak{B}$, we say that I **interprets** $\mathfrak{A}$ in $\mathfrak{B}$ (in short, $I(\mathfrak{B}) = \mathfrak{A}$) if there exists a surjective map $h : \delta^{\mathfrak{B}} \to A$, called the *coordinate map*, such that

- for all $b, c \in \delta^{\mathfrak{B}}$,
$$\mathfrak{B} \models \varepsilon(b, c) \iff h(b) = h(c);$$

- for every relation R of $\mathfrak{A}$ and all $b_1, \dots, b_r \in \delta^{\mathfrak{B}}$,

$$\mathfrak{B} \models \varphi_R(b_1, \dots, b_k) \iff (h(b_1), \dots, h(b_k)) \in R,$$

  i.e. $h^{-1}(R) = (\delta^{\mathfrak{B}})^k \cap \varphi_R^{\mathfrak{B}}$.

Hence $I = \langle \delta, \varepsilon, \varphi_{R_1}, \dots, \varphi_{R_m} \rangle$ defines (together with the function $h : \delta^{\mathfrak{B}} \to A$) an interpretation of $\mathfrak{A} = (A, R_1, \dots, R_m)$ in $\mathfrak{B}$ if and only if $\varepsilon(x,y)$ defines a congruence on the structure $(\delta^{\mathfrak{B}}, \varphi_{R_1}^{\mathfrak{B}}, \dots, \varphi_{R_m}^{\mathfrak{B}})$ and h is an isomorphism from the quotient structure $(\delta^{\mathfrak{B}}, \varphi_{R_1}^{\mathfrak{B}}, \dots, \varphi_{R_m}^{\mathfrak{B}})/\varepsilon^{\mathfrak{B}}$ to $\mathfrak{A}$.

Besides the mapping $\mathfrak{B} \mapsto I(\mathfrak{B})$ from σ-structures to τ-structures, I also defines a mapping from τ-formulae to σ-formulae. With every τ-formula ψ it associates a σ-formula ψ^I, which is obtained by relativizing every quantifier Qx to $\delta(x)$, replacing equalities $u = v$ by $\varepsilon(u,v)$, and replacing every atom $R\overline{u}$ by the corresponding formula $\varphi_R(\overline{u})$.

Lemma 3.5.8 (Interpretation Lemma). *For every interpretation I and every structure $\mathfrak{A}$, we have that*

$$\mathfrak{A} \models \psi^I \iff I(\mathfrak{A}) \models \psi.$$

We shall omit δ or ε from an interpretation if they are trivial, in the sense that $\delta(x)$ holds for all x and that $\varepsilon(x,y)$ is equivalent to $x = y$. The notion of an interpretation can be generalized in various ways. In particular, a **k-dimensional interpretation** is given by a sequence $\delta(\overline{x}), \varepsilon(\overline{x}, \overline{y}), \varphi_{R_1}(\overline{x}_1, \dots, \overline{x}_{r_1}), \dots, \varphi_{R_m}(\overline{x}_1, \dots, \overline{x}_{r_m})$, where $\overline{x}, \overline{y}, \overline{x}_1, \dots$ are disjoint k-tuples of distinct variables. A k-dimensional interpretation of $\mathfrak{A}$ in $\mathfrak{B}$ represents elements of A by elements or equivalence classes of B^k, rather than B.

Exercise 3.5.9. Show that up to first-order interpretation, all finite structures are graphs (see e.g. [66, Chapter 5] and [38, Chapter 11.2]). More precisely, for every vocabulary τ, construct an FO$[\{E\}, \tau]$-interpretation I and an FO$[\tau, \{E\}]$-interpretation J such that, for every finite structure $\mathfrak{A}$ (with at least two elements), $I(\mathfrak{A})$ is a graph and $J(I(\mathfrak{A})) \cong \mathfrak{A}$. It then follows that for every model class $\mathcal{K} \subseteq \mathrm{Fin}(\tau)$, $\mathcal{K}$ is decidable in polynomial time if, and only if, the class of graphs $\{I(\mathfrak{A}) : \mathfrak{A} \in \mathcal{K}\}$ is so.

Definition 3.5.10. Let L be a logic and $\sim$ an equivalence relation on a class $\mathcal{K}$ of τ-structures. We say that $(\mathcal{K}, \sim)$ **admits L-definable canonization** if there exists an $L[\tau, \tau \cup \{<\}]$-interpretation I such that the function $\mathfrak{A} \mapsto I(\mathfrak{A})$ is a canonization function for $\sim$. For any domain $\mathcal{D}$ of structures, we say that $(\mathcal{D}, \sim)$ admits L-definable canonization if $(\mathcal{D}(\tau), \sim)$ does for every vocabulary τ. Finally, we say that $\mathcal{D}$ admits L-definable canonization if $(\mathcal{D}, \cong)$ does.

Example 3.5.11. (**Definable canonization versus definability of order.**) Whenever $\mathcal{D}$ admits L-definable linear orders, and L is closed under first-order operations, $\mathcal{D}$ also admits L-definable canonization. This is obvious if the formula $\varphi_<$ defining the order has no parameters. If it uses parameters, then it may define, for each structure $\mathfrak{A}$, a *family* of ordered expansions $(\mathfrak{A}, <)$. But these expansions can be compared by use of the lexicographic order of their encodings. As L is closed under first-order operations, the minimal expansion with respect to this lexicographical order is L-definable, which gives an L-definable canonization.

Note, however, that there exist definable canonizations even in cases where no order is definable. Consider for instance the class of finite directed paths P_n (for $n \in \mathbb{N}$), and take their 'double graphs' (see Section 3.2.5), i.e. the graphs $2P_n = (V, E)$, where $V = \{0, \dots, n-1\} \times \{0, 1\}$ and $E = \{\langle (m, i), (m+1, j) \rangle : 0 \le m < n-1, i, j \in \{0, 1\}\}$. On this class, no order is definable in any logic and with any finite number of parameters (to see this use Exercise 3.5.4). However, the class admits DTC-definable canonization.

We shall explain the construction, which is uniform for all n, informally. The obvious equivalence relation on $2P_n$, where $(m, i) \sim (m', j)$ iff $m = m'$, is first-order definable, and so P_n is interpretable in $2P_n$. Further, the nodes 0 and $n-1$ are definable in P_n, and so $(C_n, 0)$, the directed n-cycle with a distinguished point, is interpretable in $2P_n$ as well. It therefore suffices to show that an ordered copy of $2P_n$ is interpretable in $(C_n, 0)$. We represent nodes of $2P_n$ by edges and inverse edges of C_n: the node $(m, 0)$ is represented in C_n by the pair $(m, m+1)$ and the node $(m, 1)$ by the pair $(m+1, m)$. The order on these pairs is

$$(0,1) < (1,0) < (1,2) < (2,1) < \cdots < (n-2, n-1) < (n-1, n-2).$$

The domain formula for the interpretation (of $2P_n$ in C_n) is $\delta(x, y) := Exy \vee Eyz$. It is not difficult to see that the edge relation and the linear order are definable using DTC operators. The details are left to the reader.

A simple but interesting example of definable canonization is tree canonization via fixed-point logic with counting.

Proposition 3.5.12. *The class of (directed) trees admits* (IFP + C)*-definable canonization.*

Proof. The interpretation I that we construct maps a tree $\mathcal{T} = (V, E)$ (with n nodes) to an ordered tree $I(\mathcal{T}) = (\{1, \dots, n\}, E', <)$, where $<$ is the natural order. That is, the interpretation is one-dimensional, maps nodes to numbers, and is defined by the formulae $\delta(\mu) := \exists \nu (\nu < \mu)$, $\varphi_<(\mu, \nu) := \mu < \nu$, and a formula $\varphi_{E'}(\mu, \nu)$ that we do not explicitly construct.

The construction of E' is based on an inductively defined ternary relation $F \subseteq V \times \{1, \dots, n\}^2$ that encodes the sequence of binary relations $F_v := \{(i, j) : (v, i, j) \in F\}$. For each node v of $\mathcal{T}$, let $\mathcal{T}_v$ denote the subtree of $\mathcal{T}$ with root v, and let $\mathcal{S}_v$ be the graph $(\{1, \dots, |\mathcal{T}_v|\}, F_v)$. The construction will ensure that $\mathcal{S}_v$ is isomorphic to $\mathcal{T}_v$.

If v is a leaf, let $F_v = \emptyset$. Suppose now that v has children $v_1, \dots, v_m$, and that the graphs $\mathcal{S}_{v_1}, \dots, \mathcal{S}_{v_m}$ have already been constructed. To define $\mathcal{S}_v$, we compute the code words $w_i = code(\mathcal{S}_{v_i}, <)$ (where $<$ is the natural order) and arrange them in lexicographic order. Now let $\mathcal{S}_v$ be the graph with nodes $1, \dots, |\mathcal{T}_v|$, obtained by first taking a copy of the $\mathcal{S}_{v_i}$ with the smallest code word, then taking a copy of the second, and so on, and finally adding another node that is connected to the roots of the copies of the $\mathcal{S}_{v_i}$. Obviously, $\mathcal{S}_v$ determines F_v, and $\mathcal{S}_v \cong \mathcal{T}_v$.

It is clear that the inductive construction of F can be done via an (IFP + C)-formula $\psi_F(x, \mu, \nu)$. Now take $\varphi_{E'}(\mu, \nu) := \exists x \psi_F(x, \mu, \nu)$. □

Theorem 3.5.13. *Let $\mathcal{D}$ be a domain of (finite) structures, and let L be a logic that captures* PTIME *on $\mathcal{D}^<$. If $\mathcal{D}$ admits L-definable canonization, then L captures* PTIME *on $\mathcal{D}$ also.*

Proof. Let $\mathcal{K} \in \mathcal{D}(\tau)$ be a model class tht is decidable in polynomial time, and let $\psi \in L(\tau \cup \{<\})$ be a formula defining $\mathcal{K}^<$ inside $\mathcal{D}^<(\tau)$. Further, let I be an $L[\tau, \tau \cup \{<\}]$-interpretation that defines a canonization on $\mathcal{D}(\tau)$. By the Interpretation Lemma,

$$\mathfrak{A} \models \psi^I \iff I(\mathfrak{A}) \models \psi \iff I(\mathfrak{A}) \in \mathcal{K}^< \iff \mathfrak{A} \in \mathcal{K}.$$

Hence L captures PTIME on $\mathcal{D}$. □

This result is important because it has been shown, in particular in the work of Grohe [58–60], that a number of interesting domains admit canonization via fixed-point logic with counting (IFP + C). Among these are

(1) the domain of finite (labelled) trees (see Proposition 3.5.12);
(2) the class of planar graphs [58] and, more generally, any domain of structures, whose Gaifman graphs are embeddable in a fixed surface [59];
(3) any domain of structures of bounded tree width [60].

Corollary 3.5.14. (IFP + C) *captures* PTIME *on any of these domains.*

Further, the results extend to domains that can be reduced to any of the domains mentioned above by simple definable operations such as adding or deleting a vertex or edge. An example is that of nearly planar (or apex) graphs, which become planar when one vertex is removed.

3.5.3 Capturing PTIME up to Bisimulation

In mathematics, we consider isomorphic structures as identical. Indeed, it almost goes without saying that relevant mathematical notions do not distinguish between isomorphic objects. As classical algorithmic devices work on ordered *representations of structures* rather than the structures themselves, our capturing results rely on an ability to reason about canonical ordered representations of isomorphism classes of finite structures.

However, in many application domains of logic, structures are distinguished only up to equivalences coarser than isomorphism. Perhaps the best-known example is the modelling of the computational behaviour of (concurrent) programs by transition systems. The meaning of a program is usually not captured by a unique transition system. Rather, transition systems are distinguished only up to appropriate notions of behavioural equivalence, the most important of these being *bisimulation*.

In such a context, the idea of a logic capturing PTIME gets a new twist. One would like to express in a logic precisely those properties of structures that are

(1) decidable in polynomial time, and
(2) invariant under the notion of equivalence being studied.

Let us look at one specific problem in this context, the problem of bisimulation-invariant properties of transition systems.

Definition 3.5.15. Let $G = (V, (E_a)_{a\in A}, (P_b)_{b\in B})$ and $G' = (V', (E'_a)_{a\in A}, (P'_b)_{b\in B})$ be two transition systems of the same vocabulary. A **bisimulation** between G and G' is a non-empty relation $Z \subseteq V \times V'$, respecting the P_b in the sense that $v \in P_b$ iff $v' \in P'_b$, for all $b \in B$ and $(v, v') \in Z$, and satisfying the following back and forth conditions.

Forth. for all $(v, v') \in Z$, $a \in A$ and every w such that $(v, w) \in E_a$, there exists a w' such that $(v', w') \in E'_a$ and $(w, w') \in Z$.

Back. for all $(v, v') \in Z$, $a \in A$ and every w' such that $(v', w') \in E'_a$, there exists a w such that $(v, w) \in E_a$ and $(w, w') \in Z$.

A **rooted transition system** is a pair (G, u), where G is a transition system G and u is a node of G. Two rooted transition systems (G, u) and (G', u') are **bisimilar**, denoted by $G, u \sim G', u'$, if there is a bisimulation Z between G and G' with $(u, u') \in Z$.

Exercise 3.5.16. Bisimulation is a greatest fixed point. Prove that two nodes u, u' of a transition system G are bisimilar, i.e. $(G, u) \sim (G, u')$ if, and only if,

$$\begin{aligned} G \models [\mathbf{gfp} Rxy \,.\, & \bigwedge_{b\in B} P_b x \leftrightarrow P_b y \wedge \\ & \bigwedge_{a\in A} (\forall x' \,.\, E_a x x')(\exists y' \,.\, E_a y y') R x' y' \wedge \\ & \bigwedge_{a\in A} (\forall y' \,.\, E_a y y')(\exists x' \,.\, E_a x x') R x' y'](u, u'). \end{aligned}$$

A class S of rooted transition systems is **invariant under bisimulation** if, whenever $(G, u) \in S$ and $(G, u) \sim (G', u')$, then also $(G', u') \in S$. We say that a class S of finite rooted transition systems is in **bisimulation-invariant**

PTIME if it is invariant under bisimulation, and if there exists a polynomial-time algorithm deciding whether a given pair (G, u) belongs to S. A logic L is invariant under bisimulation if all L-definable properties of rooted transition systems are.

Exercise 3.5.17. Prove that ML, the modal μ-calculus L_μ, and the infinitary modal logic ML^∞ are invariant under bisimulation.

Clearly, $L_\mu \subseteq$ bisimulation-invariant PTIME. However, as pointed out in Section 3.3.3, L_μ is far too weak to *capture* this class, mainly because it is essentially a monadic logic. Instead, we have to consider a *multidimensional* variant L_μ^ω of L_μ.

But before we define this logic, we should explain the main technical step, which relies on definable canonization, but of course with respect to bisimulation rather than isomorphism. For simplicity of notation, we consider only transition systems with a single transition relation E. The extension to the case of several transition relations E_a is completely straightforward.

With a rooted transition system $G = (V, E, (P_b)_{b \in B}), u$, we associate a new transition system

$$G_u^\sim := (V_u^\sim, E^\sim, (P_b^\sim)_{b \in B}),$$

where $V_u^\sim$ is the set of all $\sim$-equivalence classes $[v]$ of nodes $v \in V$ that are reachable from u. More formally, let $[v]$ denote the bisimulation equivalence class of a node $v \in V$. Then

$$\begin{aligned} V_u^\sim &:= \{[v] : \text{there is a path in } G \text{ from } u \text{ to } v\} \\ P_b^\sim &:= \{[v] \in V_u^\sim : v \in P_b\} \\ E^\sim &:= \{([v], [w]) : (v, w) \in E\}. \end{aligned}$$

As shown in the following exercise, the pair $G_u^\sim, [u]$ is, up to isomorphism, a *canonical representant* of the bisimulation equivalence class of G, u.

Exercise 3.5.18. Prove that (1) $(G, u) \sim (G_u^\sim, [u])$, and (2) if $(G, u) \sim (H, v)$, then $(G_u^\sim, [u]) \cong (H_v^\sim, [v])$.

It follows that a class S of rooted transition systems is bisimulation-invariant if and only if $S = \{(G, u) : (G_u^\sim, [u]) \in S\}$. Let $\mathcal{CR}^\sim$ be the domain of canonical representants of finite transition systems, i.e.

$$\mathcal{CR}^\sim := \{(G, u) : (G_u^\sim, [u]) \cong (G, u)\}.$$

Proposition 3.5.19. *$\mathcal{CR}^\sim$ admits* LFP*-definable linear orderings.*

Proof. We show that for every vocabulary $\tau = \{E\} \cup \{P_b : b \in B\}$, there exists a formula $\psi(x, y) \in \mathrm{LFP}(\tau)$ which defines a linear order on every transition system in $\mathcal{CR}^\sim(\tau)$.

Recall that bisimulation equivalence on a transition system is a greatest fixed point. Its complement, bisimulation inequivalence, is a least fixed point, which is the limit of an increasing sequence $\not\sim_i$ defined as follows: $u \not\sim_0 v$ if u and v do not have the same atomic type, i.e. if there exists some b such that one of the nodes u, v has the property P_b and the other does not. Further, $u \not\sim_{i+1} v$ if the sets of $\sim_i$-classes that are reachable in one step from u and v are different. The idea is to refine this inductive process, by defining relations $\prec_i$ that order the $\sim_i$-classes. On the transition system itself, these relations are pre-orders. The inductive limit $\prec$ of the pre-orders $\prec_i$ defines a linear order of the bisimulation equivalence classes. But in transition systems in $\mathcal{CR}^\sim$, bisimulation classes have only one element, so $\prec$ actually defines a linear order on the set of nodes.

To make this precise, we choose an order on B and define $\prec_0$ by enumerating the $2^{|B|}$ atomic types with respect to the propositions P_b, i.e.

$$x \prec_0 y := \bigvee_{b \in B} \Big(\neg P_b x \wedge P_b y \wedge \bigwedge_{b' < b} P_{b'} x \leftrightarrow P_{b'} y \Big).$$

In what follows, $x \sim_i y$ can be taken as an abbreviation for $\neg(x \prec_i y \vee y \prec_i x)$, and similarly for $x \sim y$. We define $x \prec_{i+1} y$ by the condition that either $x \prec_i y$, or $x \sim_i y$ and the set of $\sim_i$-classes reachable from x is lexicographically smaller than the set of $\sim_i$-classes reachable from y. Note that this inductive definition of $\prec$ is not monotone, so it cannot be directly captured by an LFP-formula. However, as we know that LFP $\equiv$ IFP, we can use an IFP-formula instead. Explicitly, $\prec$ is defined by $[\mathbf{ifp} x \prec y \,.\, \psi(\prec, x, y)](x, y)$, where

$$\begin{aligned} \psi(\prec, x, y) := & \, x \prec_0 y \vee \Big(x \sim y \wedge \\ & (\exists y' \,.\, Eyy') \Big((\forall x' \,.\, Exx') x' \not\sim y' \wedge \\ & \qquad (\forall z . z \prec y') \big(\exists x'' (Exx'' \wedge x'' \sim z) \leftrightarrow \\ & \qquad\qquad \exists y'' (Eyy'' \wedge y'' \sim z) \big) \Big) \Big). \end{aligned}$$

□

Exercise 3.5.20. Complete the proof by showing that the formula $[\mathbf{ifp} x \prec y \,.\, \psi(\prec, x, y)](x, y)$ indeed defines the order described above.

Corollary 3.5.21. *On the domain $\mathcal{CR}^\sim$,* LFP *captures* PTIME.

In fact, this result already suffices to give an *abstract capturing result* for bisimulation-invariant PTIME (in the sense of the following section): by composing the mapping from rooted transition systems to their canonical representants with LFP queries on these representants, we obtain an abstract logic with recursive syntax and polynomial-time semantics that describes precisely the polynomial-time computable, bisimulation-invariant queries on rooted transition systems.

In many situations (such as for polynomial time on arbitrary finite structures), we would actually be quite happy with such an abstract capturing result. However, in the bisimulation-invariant scenario we can do better and capture PTIME in terms of a natural logic, the multidimensional μ-calculus L_μ^ω.

Definition 3.5.22. The syntax of the k**-dimensional** μ**-calculus** L_μ^k (for transition systems $G = (V, E, (P_b)_{b\in B}))$ is the same as the syntax of the usual μ-calculus L_μ with modal operators $\langle i\rangle$, $[i]$ for $a \in A, i = 1, \dots, k$, and $\langle\sigma\rangle, [\sigma]$ for every substitution $\sigma : \{1, \dots, k\} \to \{1, \dots, k\}$. Let $S(k)$ be the set of all these substitutions.

The semantics is different, however. A formula ψ of L_μ^k is interpreted on a transition system $G = (V, E, (P_b)_{b\in B})$ at node v by evaluating it as a formula of L_μ on the modified transition system

$$G^k = (V^k, (E_i)_{1\le i\le k}, (E_\sigma)_{\sigma\in S(k)}, (P_{b,i})_{b\in B, 1\le i\le k})$$

at node $\underline{v} := (v, v, \dots, v)$. Here $V^k = V \times \cdots \times V$ and

$$\begin{aligned}
E_i &:= \{(\overline{v}, \overline{w}) \in V^k \times V^k : (v_i, w_i) \in E \text{ and } v_j = w_j \text{ for } j \neq i\}\\
E_\sigma &:= \{(\overline{v}, \overline{w}) \in V^k \times V^k : w_i = v_{\sigma(i)} \text{ for all } i\}\\
P_{b,i} &:= \{\overline{v} \in V^k : v_i \in P_b\}
\end{aligned}$$

That is, $G, v \models_{L_\mu^k} \psi$ iff $G^k, (v, \dots, v) \models_{L_\mu} \psi$. The **multidimensional** μ**-calculus** is $L_\mu^\omega = \bigcup_{k<\omega} L_\mu^k$.

Remark. Instead of evaluating a formula $\psi \in L_\mu^k$ at single nodes v of G, we can also evaluate it at k-tuples of nodes: $G, \overline{v} \models_{L_\mu^k} \psi$ iff $G^k, \overline{v} \models_{L_\mu} \psi$.

Example 3.5.23. Bisimulation is definable in L_μ^2 (in the sense of the remark just made). Let

$$\psi^\sim := \nu X \,.\, \Big(\bigwedge_{b\in B} (P_{b,1} \leftrightarrow P_{b,2}) \wedge [1]\langle 2\rangle X \wedge [2]\langle 1\rangle X\Big).$$

For every transition system G, we have that $G, v_1, v_2 \models \psi^\sim$ if, and only if, v_1 and v_2 are bisimilar in G. Further, we have that

$$G, v \models \mu Y \,.\, \langle 2\rangle(\psi^\sim \vee \langle 2\rangle Y)$$

if, and only if, there exists in G a point w that is reachable from v (by a path of length ≥ 1) and bisimilar to v.

Exercise 3.5.24. Prove that L_μ^ω is invariant under bisimulation. Further, show that L_μ^ω can be embedded in LFP.

This exercise establishes the easy direction of the desired result: $L_\mu^\omega \subseteq$ bisimulation-invariant PTIME. For the converse, it suffices to show that LFP and L_μ^ω are equivalent on the domain $\mathcal{CR}^\sim$. Let S be a class of rooted transition systems in bisimulation-invariant PTIME. For any (G, u), we have that $(G, u) \in S$ if its canonical representant $(G_u^\sim, [u]) \in S$. If LFP and L_μ^ω are equivalent on $\mathcal{CR}^\sim$, then there exists a formula $\psi \in L_\mu^\omega$ such that $G_u^\sim, [u] \models \psi$ iff $(G_u^\sim, [u]) \in S$. By the bisimulation invariance of ψ, it follows that $G, u \models \psi$ iff $(G, u) \in S$.

Proposition 3.5.25. *On the domain $\mathcal{CR}^\sim$,* LFP $\leq L_\mu^\omega$*. More precisely, for each formula $\psi(x_1, \dots, x_{k+1}) \in$* LFP *of width $\leq k+1$, there exists a formula $\psi^* \in L_\mu^{k+1}$ such that for each $(G, u) \in \mathcal{CR}^\sim$, we have that $G \models \psi(u, \overline{v})$ iff $G, u, \overline{v} \models \psi^*$.*

Note that although, ultimately, we are interested only in formulae $\psi(x)$ with just one free variable, we need more general formulae, and evaluation of L_μ^k-formulae over k-tuples of nodes, for the inductive treatment. In all formulae, we shall have at least x_1 as a free variable, and we always interpret x_1 as u (the root of the transition system). We remark that, by an obvious modification of the formula given in Exercise 3.5.23, we can express in L_μ^k the assertion that $x_i \sim x_j$ for any i, j.

Atomic formulae are translated from LFP to L_μ^ω according to

$$\begin{aligned}
(x_i = x_j)^* &:= x_i \sim x_j \\
(P_b x_i)^* &:= P_{b,i}\overline{x} \\
(E x_i x_j)^* &:= \langle i \rangle x_i \sim x_j \\
(X x_{\sigma(1)} \cdots x_{\sigma(r)})^* &:= \langle \sigma \rangle X.
\end{aligned}$$

Boolean connectives are treated in the obvious way, and *quantifiers* are translated by use of fixed points. To find a witness x_j satisfying a formula ψ, we start at u (i.e. set $x_j = x_1$), and search along transitions (i.e. use the μ-expression for reachability). That is, let $j/1$ be the substitution that maps j to 1 and fixes the other indices, and translate $\exists x_j \psi(\overline{x})$ into

$$\langle j/1 \rangle \mu Y \, . \, \psi^* \vee \langle j \rangle Y.$$

Finally, *fixed points* are first brought into normal form so that variables appear in the right order, and then they are translated literally, i.e. $[\mathbf{lfp} X\overline{x} \, . \, \psi](\overline{x})$ translates into $\mu X \, . \, \psi^*$.

The proof that the translation has the desired property is a straightforward induction, which we leave as an exercise (see [90] for details). Altogether we have established the following result.

Theorem 3.5.26 (Otto). *The multidimensional μ-calculus captures bisimulation-invariant* PTIME.

Otto has also established capturing results with respect to other equivalences. For finite structures $\mathfrak{A}, \mathfrak{B}$, we say that $\mathfrak{A} \equiv_k \mathfrak{B}$ if no first-order sentence of width k can distinguish between $\mathfrak{A}$ and $\mathfrak{B}$. Similarly, $\mathfrak{A} \equiv_k^C \mathfrak{B}$ if $\mathfrak{A}$ and $\mathfrak{B}$ are indistinguishable by first-order sentences of width k with counting quantifiers of the form $\exists^{\geq i}x$, for any $i \in \mathbb{N}$.

Theorem 3.5.27 (Otto). *There exist logics that effectively capture $\equiv_2$-invariant* PTIME *and $\equiv_2^C$-invariant* PTIME *on the class of all finite structures.*

For details, see [89].

3.5.4 Is There a Logic for PTIME?

To discuss the problem of whether PTIME can be captured on the domain of *all* finite structures, we need to make precise the notion of a logic, and to refine the notion of a logic capturing a complexity class, so as to exclude pathological examples such the following, which is due to Gurevich [61].

Example 3.5.28. Let the syntax of our 'logic' consist of all pairs (M, k), where M is a Turing machine, and k a natural number. A finite τ-structure $\mathfrak{A}$ is a model of (M, k) if there exists a model class $\mathcal{K} \subseteq \text{Fin}(\tau)$ such that $\mathfrak{A} \in \mathcal{K}$, and M accepts an encoding $code(\mathfrak{B}, <)$ of a finite τ-structure $\mathfrak{B}$ in time $|B|^k$ if, and only if, $\mathfrak{B} \in \mathcal{K}$. Note that this 'logic' captures PTIME on finite structures. But the example is pathological, not mainly because of its unusual format, but because its semantics is not effective: it is undecidable whether a Turing machine accepts an isomorphism-closed class of structures.

Another example of this kind is *order-invariant* LFP. The τ-sentences of this logic are the LFP-sentences of vocabulary $\tau \cup \{<\}$ such that, for all finite τ-structures $\mathfrak{A}$ and all linear orders $<, <'$ on $\mathfrak{A}$, we have that $(\mathfrak{A}, <) \models \psi$ if and only if $(\mathfrak{A}, <') \models \psi$. This defines the syntax. The semantics is the obvious one: a structure $\mathfrak{A}$ is a model of ψ if, and only if, $(\mathfrak{A}, <) \models \psi$ for some, and hence all, linear orders on $\mathfrak{A}$. This 'logic' also captures PTIME, but again it has an undesirable feature: it is undecidable whether a given sentence $\psi \in$ LFP is order-invariant (compare Exercise 3.1.12), so the 'logic' does not have an effective syntax.

We start by defining a general notion of a logic on finite structures by imposing two requirements: an effective syntax and an isomorphism-invariant semantics.

Definition 3.5.29. A **logic** on a domain $\mathcal{D}$ of finite structures is a pair $(L, \models)$, where L is a function that assigns to each vocabulary τ a decidable set $L(\tau)$ (whose elements are called τ-sentences), and $\models$ is a binary relation between sentences and finite structures, so that for each sentence $\psi \in L(\tau)$, the class $\{\mathfrak{A} \in \mathcal{D}(\tau) : \mathfrak{A} \models \psi\}$ is closed under isomorphism.

Recall that, by Definition 3.2.10, a logic captures PTIME on a domain $\mathcal{D}$ if every polynomial-time decidable model class in $\mathcal{D}$ is definable in that logic, and if, for every sentence of the logic, the model-checking problem on $\mathcal{D}$ can be solved in polynomial time. To exclude pathological examples such the first one above, we impose in addition the condition that for each sentence, a polynomial-time model-checking algorithm can be effectively constructed.

Definition 3.5.30. A logic $(L, \models)$ **effectively captures** PTIME on a domain $\mathcal{D}$ of finite structures if it captures PTIME in the sense of Definition 3.2.10 and, moreover, there exists a computable function, which associates with every sentence $\psi \in L(\tau)$ an algorithm M and a polynomial p, such that M decides $\{\mathfrak{A} \in \mathcal{D}(\tau) : \mathfrak{A} \models \psi\}$ in time $p(n)$. We simply say that $(L, \models)$ effectively captures PTIME if it does so on the class of all finite structures.

This definition can be modified in the obvious way to other complexity classes. All capturing results that we have proved so far are effective in this sense.

Exercise 3.5.31. A complexity class $\mathcal{C}$ is **recursively indexable** on a domain $\mathcal{D}$ if there is a recursive index set I, a computable function f mapping every $i \in I$ to (the code of) a Turing machine M_i, and an appropriate resource bound (e.g. a polynomial bounding the running time of M_i) such that:

(1) The class $\mathcal{K}_i$ of all structures from $\mathcal{D}$ accepted by M_i is in $\mathcal{C}$, and, moreover, M_i together with the given resource bound witnesses the membership of $\mathcal{K}_i$ in the complexity class $\mathcal{C}$.
(2) For each model class $\mathcal{K} \in \mathcal{C}$ on the domain $\mathcal{D}$, there is an $i \in I$ such that M_i decides $\mathcal{K}$.

Prove that there is a logic that effectively captures $\mathcal{C}$ on the domain $\mathcal{D}$ if, and only if, $\mathcal{C}$ is recursively indexable on $\mathcal{D}$.

The above definition of a logic may seem too abstract for practical purposes. However, it is justified by the equivalence with recursive indexings, as described in the exercise above, and by a result of Dawar [32], which shows that if there is any logic that effectively captures PTIME, then there also exists a natural one. More precisely, Dawar proved that, from any logic effectively capturing PTIME, one could extract a model class $\mathcal{K}$ that is complete for PTIME under first-order reductions. As a consequence, PTIME would also be effectively captured by the logic FO$[\mathcal{Q}^{\omega}_{\mathcal{K}}]$, which adjoins to FO the vectorized Lindstöm quantifiers associated with $\mathcal{K}$ (see [32, 38] for more information).

Exercise 3.5.32. Many finite-model theorists conjecture that there is no logic that effectively captures PTIME on finite structures. If you are the first to prove this, you may win one million dollars. Why?

3.6 Algorithmic Model Theory

3.6.1 Beyond Finite Structures

For a long time, descriptive complexity theory has been concerned almost exclusively with finite structures. Although important problems remain open, the relationship between definability and complexity on finite structures is now fairly well understood, and there are interesting connections to fields such as databases, knowledge representation, and computer-aided verification.

However, for many applications, the strict limitation to finite structures is too restrictive. In most of the fields mentioned above, there have been considerable efforts to extend the relevant methodology from finite structures to suitable domains of infinite ones. In particular, this is the case for databases and computer-aided verification where infinite structures (like constraint databases or transition systems with infinite state spaces) are of increasing importance.

Finite model theory should therefore be generalized to a more comprehensive *algorithmic model theory* that extends the research programme, the general approach, and the methods of finite model theory to interesting domains of infinite structures. From a more general theoretical point of view, one may ask what domains of infinite structures are suitable for such an extension. More specifically, one may ask what conditions must be satisfied by a domain $\mathcal{D}$ of structures that are not necessarily finite such that the approach and methods of finite model theory make sense. There are two obvious and fundamental conditions:

Finite representations. Every structure $\mathfrak{A} \in \mathcal{D}$ should be representable in a finite way (e.g. by a binary string, an algorithm, a collection of automata, an axiomatization in some logic, an interpretation, ...).

Effective semantics. For the relevant logics (e.g. first-order logic), the model-checking problem on $\mathcal{D}$ should be decidable. That is, given a sentence $\psi \in L$ and a representation of a structure $\mathfrak{A} \in \mathcal{D}$, it should be decidable whether $\mathfrak{A} \models \psi$.

These are just minimal requirements, which may need to be refined according to the context and the questions to be considered. We may, for instance, also require the following:

Closure. For every structure $\mathfrak{A} \in \mathcal{D}$ and every formula $\psi(\overline{x})$, the expansion $(\mathfrak{A}, \psi^{\mathfrak{A}})$ of $\mathfrak{A}$ with the relation defined by ψ, should as well be contained in $\mathcal{D}$.

Effective query evaluation. Suppose that we have fixed a way of representing structures. Given a representation of $\mathfrak{A} \in \mathcal{D}$ and a formula $\psi(\overline{x})$, we should be able to compute a representation of $\psi^{\mathfrak{A}}$ (or of the expanded structure $(\mathfrak{A}, \psi^{\mathfrak{A}})$).

Note that, contrary to the case of finite structures, query evaluation does not necessarily reduce to model checking. Further, instead of just effectiveness of these tasks, it may be required that they can be performed within some complexity bounds.

3.6.2 Finitely Presentable Structures

We briefly survey here some domains of infinite but finitely presentable structures which may be relevant to algorithmic model theory. We shall then discuss in a more detailed way *metafinite structures*, for which descriptive complexity issues have already been studied quite intensively.

Recursive structures are countable structures whose functions and relations are computable and therefore finitely presentable. They have been studied quite intensively in model theory since the 1960s (see e.g. [6, 42]). Although recursive model theory is very different from finite model theory, there have been some papers studying classical issues of finite model theory on recursive structures and recursive databases [50, 64, 65, 94]. However, for most applications, the domain of recursive structures is far too large. In general, only quantifier-free formulae admit effective evaluation algorithms.

Constraint databases provide a database model that admits infinite relations that are finitely presented by quantifier-free formulae (constraints) over some fixed background structure. For example, to store geometrical data, it is useful not just to have a finite set as the universe of the database, but to include all real numbers 'in the background'. Also, the presence of interpreted functions on the real numbers, such as addition and multiplication, is desirable. The constraint database framework introduced by Kanellakis, Kuper, and Revesz [74] meets both requirements. Formally, a constraint database consists of a *context structure* $\mathfrak{A}$, such as $(\mathbb{R}, <, +, \cdot)$, and a set $\{\varphi_1, \ldots, \varphi_m\}$ of quantifier-free formulae defining the database relations. Constraint databases are treated in detail in [81] and in Chap. 5 of this book.

Automatic structures are structures whose functions and relations are represented by finite automata. Informally, a relational structure $\mathfrak{A} = (A, R_1, \ldots, R_m)$ is automatic if we can find a regular language $L_\delta \subseteq \Sigma^*$ (which provides names for the elements of $\mathfrak{A}$) and a function $\nu : L_\delta \to A$ mapping every word $w \in L_\delta$ to the element of $\mathfrak{A}$ that it represents. The function ν must be surjective (every element of $\mathfrak{A}$ must be named) but need not be injective (elements can have more than one name). In addition, it must be recognizable by finite automata (reading their input words synchronously) whether two words in L_δ name the same elements, and, for each relation R_i of $\mathfrak{A}$, whether a given tuple of words in L_δ names a tuple in R_i.

Example 3.6.1. (1) All finite structures are automatic.

(2) Some important examples of automatic structures are Presburger arithmetic $(\mathbb{N}, +)$, and its expansions $\mathfrak{N}_p := (\mathbb{N}, +, |_p)$ by the relation $x \,|_p\, y$ which says that x is a power of p dividing y. Using p-ary encodings (starting with the least significant digit), it is not difficult to construct automata recognizing equality, addition, and $|_p$.

(3) For $p \in \mathbb{N}$, let $\mathit{Tree}(p) := (\{0, \dots, p-1\}^*, (\sigma_i)_{i<p}, <, \mathrm{el})$, where $\sigma_i(x) := xi$, $x < y$ means that $xz = y$ for some z, and $\mathrm{el}(x, y)$ means that x and y have equal length. Obviously, these structures are automatic as well.

Automatic structures provide a vast playground for finite-model theorists, with many examples of high relevance to computer science. There are also interesting connections to computational group theory, where *automatic groups* have already been studied quite intensively [41, 44]. The general notion of structures presentable by automata was proposed in [75], and their theory has been developed in [16, 18, 19, 92].

The notion of an automatic structure can be modified and generalized in many directions. By using automata over infinite words, we obtain the notion of **ω-automatic structures** (which, unlike automatic structures, may have uncountable cardinality).

Example 3.6.2. (1) All automatic structures are ω-automatic.

(2) The additive group of reals, $(\mathbb{R}, +)$, and indeed the expanded structure $\mathfrak{R}_p := (\mathbb{R}, +, \leq, |_p, 1)$ are ω-automatic, where

$$x \mid_p y \text{ iff } x = p^n \text{ and } y = kx \text{ for some } n, k \in \mathbb{Z}.$$

(3) The tree structures $\mathit{Tree}(p)$ can be extended in a natural way to the (uncountable) ω-automatic structures $\mathit{Tree}^\omega(p) = (\{0, \dots, p-1\}^{\leq \omega}, (\sigma_i)_{i<p}, \preceq, \mathrm{el})$.

Unlike the class of recursive structures, automatic structures and ω-automatic structures admit effective (in fact, automatic) evaluation of all first-order queries and possess many other pleasant algorithmic properties.

Theorem 3.6.3. *The model checking problems for first-order logic on the domains of automatic or ω-automatic structures are decidable.*

There are a number of extensions of this result, for instance to the extension of first-order logic by the quantifier 'there exist infinitely many' [19]. There also are model-theoretic characterizations of automatic and ω-automatic structures, in terms of *interpretations* into appropriate expansions of Presburger arithmetic, trees, or the additive group of reals (see Examples 3.6.1 and 3.6.2). We write $\mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{B}$ to denote that there exists a first-order interpretation of $\mathfrak{A}$ in $\mathfrak{B}$. Note that the domains of automatic and ω-automatic structures are closed under fist-order interpretations.

Theorem 3.6.4 (Blumensath and Grädel). *(1) For every structure $\mathfrak{A}$, the following are equivalent:*

(i) $\mathfrak{A}$ *is automatic.*
(ii) $\mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{N}_p$ *for some (and hence all)* $p \geq 2$.
(iii) $\mathfrak{A} \leq_{\mathrm{FO}} \mathit{Tree}(p)$ *for some (and hence all)* $p \geq 2$.

(2) For every structure $\mathfrak{A}$, the following are equivalent:

(i) $\mathfrak{A}$ *is ω-automatic.*
(ii) $\mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{R}_p$ *for some (and hence all) $p \geq 2$.*
(iii) $\mathfrak{A} \leq_{\mathrm{FO}} \mathit{Tree}^{\omega}(p)$ *for some (and hence all) $p \geq 2$.*

For a proof, see [19] There are similar characterizations for tree-automatic structures [16]. For further results on automatic structures, see [10, 16, 18, 19, 75–78, 92].

The model-theoretic characterizations of automatic and ω-automatic structures in terms of interpretability suggest a general way to obtain other domains of infinite structures that may be interesting for algorithmic model theory: fix a structure $\mathfrak{A}$ with 'nice' (algorithmic and/or model-theoretic) properties and an appropriate notion of interpretation, and consider the class of all structures that are interpretable in $\mathfrak{A}$. Obviously, each structure in this class is finitely presentable (by an interpretation). Further, many 'nice' properties are preserved by interpretations, and so every structure in the class inherits them from $\mathfrak{A}$. In particular, every class of queries that is effective on $\mathfrak{A}$ and closed under first-order operations is effective on the closure of $\mathfrak{A}$ under first-order interpretations. This approach is also relevant to the domain of structures that we discuss next.

Tree-interpretable structures are structures that are interpretable in the infinite binary tree $\mathcal{T}^2 = (\{0,1\}^*, \sigma_0, \sigma_1)$ via a (one-dimensional) MSO-interpretation. By Rabin's Theorem, monadic second-order formulae can be effectively evaluated on $\mathcal{T}^2$. Since MSO is closed under one-dimensional interpretations, the Interpretation Lemma implies that tree-interpretable structures admit effective evaluation for MSO. Tree-interpretable structures generalize various notions of infinite graphs that have been studied in logic, automata theory and, verification. Some examples are **context-free graphs** [87, 88], which are the configuration graphs of pushdown automata, **HR-equational** and **VR-equational graphs** [27], which are defined via certain graph grammars, and **prefix-recognizable graphs** [25], which can for instance be defined as graphs of the form $(V, (E_a)_{a\in A})$, where V is a regular language and each edge relation E_a is a finite union of sets $X(Y \times Z) = \{(xy, xz) : x \in X, y \in Y, z \in Z\}$, for regular languages X, Y, Z. In fact, some of these classes coincide with the class of tree-interpretable graphs (see [17]).

Theorem 3.6.5. *For any graph $G = (V, (E_a)_{a\in A})$, the following are equivalent:*

(i) G is tree-interpretable.
(ii) G is VR-equational.
(iii) G is prefix-recognizable.
(iv) G is the restriction to a regular set of the configuration graph of a pushdown automaton with ε-transitions.

On the other hand, the classes of context-free graphs and of HR-equational graphs are strictly contained in the class of tree-interpretable graphs.

Exercise 3.6.6. Prove that every tree-interpretable structure is automatic. Is the converse also true?

Tree-Constructible Structures: the Caucal Hierarchy

The question arises of whether there are even more powerful domains than the tree-interpretable structures on which monadic second-order logic is effective. An interesting way to obtain such domains is to use tree constructions that associate with any structure a kind of tree unravelling. A simple variant is the **unfolding** of a labelled graph G from a given node v to the tree $\mathcal{T}(G, v)$. Courcelle and Walukiewicz [28, 29] have shown that the MSO-theory of $\mathcal{T}(G, v)$ can be effectively computed from the MSO-theory of (G, v). A more general operation, applicable to relational structures of any kind, has been invented by Muchnik. Given a relational structure $\mathfrak{A} = (A, R_1, \dots, R_m)$, let its **iteration** $\mathfrak{A}^* = (A^*, R_1^*, \dots, R_m^*, \mathit{suc}, \mathit{clone})$ be the structure with universe A^*, relations $R_i^* = \{(wa_1, \dots, wa_r) : w \in A^*, (a_1, \dots, a_r) \in R_i\}$, the successor relation $\mathit{suc} = \{(w, wa) : w \in A^*, a \in A\}$, and the predicate *clone* consisting of all elements of the form waa. It is not difficult to see that unfoldings of graphs are first-order interpretable in their iterations. Muchnik's Theorem states that the monadic theory of $\mathfrak{A}^*$ is decidable if the monadic theory of $\mathfrak{A}$ is so (for proofs, see [11, 101]). We define the domain of **tree-constructible structures** to be the closure of the domain of finite structures under (one-dimensional) MSO-interpretations and iterations. By Muchnik's Theorem, and since effective MSO model checking is preserved under interpretations, the tree constructible structures are finitely presentable and admit effective evaluation of MSO-formulae.

The tree-constructible graphs form the **Caucal hierarchy**, which was defined in [26] in a slighly different way. The definition is easily extended to arbitrary structures: let $\mathcal{C}_0$ be the class of finite structures, and let $\mathcal{C}_{n+1}$ be the class of structures that are interpretable in the iteration $\mathfrak{A}^*$ of a structure $\mathfrak{A} \in \mathcal{C}_n$. There are a number of different, but equivalent, ways to define the levels of the Caucal hierarchy. For instance, one can use the inverse rational mappings given in [25] rather than monadic interpretations, and simple unfoldings rather than iterations without changing the hierarchy [24]. Equivalently, the hierarchy can be defined via higher-order pushdown automata. It is known that the Caucal hierarchy is strict, and that it does not exhaust the class of all structures with a decidable MSO-theory. We refer to [24, 98] for details and further information.

3.6.3 Metafinite Structures.

The class of infinite structures for which descriptive complexity theory has been studied most intensively is the class of the metafinite structures, proposed by Grädel and Gurevich [48], and studied also in [30, 49, 53, 84]. These structures are somewhat reminiscent of the two-sorted structures that we used to define fixed-point logic with counting, (IFP + C). There, the second sort was a finite linear order $(\{0, \dots, n\}, <)$. Metafinite structures are similar two-sorted structures, with the essential differences that (1) the numerical sort need not be finite, (2) the structures may contain functions from the first to the second sort, and (3) operations more general than counting are considered.

Definition 3.6.7. A **(simple) metafinite structure** is a triple $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ consisting of the following:

(i) A finite structure $\mathfrak{A}$, called the primary part of $\mathfrak{D}$.
(ii) A finite or infinite structure $\mathfrak{R}$, called the secondary (or numerical) part of $\mathfrak{D}$. We always assume that $\mathfrak{R}$ contains two distinguished elements 0 and 1 (or *true* and *false*).
(iii) A finite set W of functions $w : A^k \to R$.

The *vocabulary* of $\mathfrak{D}$ is the triple $\tau(\mathfrak{D}) = (\tau_a, \tau_r, \tau_w)$, where each component of $\tau(\mathfrak{D})$ is the set of relation or function symbols in the corresponding component of $\mathfrak{D}$. (We always consider constants as functions of arity 0.) The two distinguished elements 0, 1 of $\mathfrak{R}$ are named by constants of τ_r.

Example 3.6.8. **($\mathbb{R}$-structures)** The descriptive complexity theory over the real numbers developed by Grädel and Meer [53] (see Sect. 3.6.5) is based on $\mathbb{R}$-structures, which are simple metafinite structure with a secondary part $\mathfrak{R} = (\mathbb{R}, +, -, \cdot, /, \leq, (c_r)_{r\in\mathbb{R}})$. It is convenient to include subtraction and division as primitive operations and assume that every element $r \in \mathbb{R}$ is named by a constant c_r, so that any rational function $g : \mathbb{R}^k \to \mathbb{R}$ (i.e. any quotient of two polynomials) can be written as a term.

There are many variations of metafinite structures. An important one is **metafinite structures with multiset operations**. Any function $f : A \to R$ defines a multiset $\mathrm{mult}(f) = \{\!\{ f(a) : a \in A \}\!\}$ over R (where the notation $\{\!\{ \dots \}\!\}$ indicates that we may have multiple occurrences of the same element). For any set R, let $\mathrm{fm}(R)$ denote the class of all finite multisets over R. In some of the metafinite structures that we consider, the secondary part $\mathfrak{R}$ is not just a (first-order) structure in the usual sense, but instead it comes with a collection of *multiset operations* $\Gamma : \mathrm{fm}(R) \to R$, mapping finite multisets over R to elements of R. Some natural examples on, say, the real numbers are addition, multiplication, counting, mean, maximum, and minimum. The use of multiset operations will become clearer when we introduce logics for metafinite structures. Let us just remark that multiset operations are a natural way to make precise the notion of *aggregates* in database query languages such as SQL.

Example 3.6.9. (Arithmetical structures). Of particular interest to us are metafinite structures, whose secondary part is a structure $\mathfrak{N}$ over the natural numbers such that

- $\mathfrak{N}$ includes at least the constants 0, 1, the functions $+, \cdot$, the ordering relation $<$, and the multiset operations $\max, \min, \sum$ (sum), and $\prod$ (product).
- All functions, relations, and multiset operations of $\mathfrak{N}$ can be evaluated in polynomial time.

We call metafinite structures of this kind **arithmetical structures**. A *simple arithmetical structure* is obtained from an arithmetical structure by omitting the multiset operations.

By itself, the notion of metafinite structures contains nothing revolutionary: they are just a special kind of two-sorted structures. The interesting feature of metafinite model theory is not just the structures themselves, but the logics, which access the primary and the secondary part in different ways and are designed so that the approach and methods of finite model theory remain meaningful and applicable. An important feature of these logics is that they contain, besides formulae and terms in the usual sense, a calculus of *weight terms* from the primary to the secondary part.

Definition 3.6.10. Let L be any of the logics for finite structures, such as FO, LFP, ... as described in the previous sections, and let $\tau = (\tau_a, \tau_r, \tau_w)$ be a vocabulary for metafinite structures (where τ_r may or may not have names for multiset operations). The appropriate modification of L for reasoning about metafinite structures $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ of vocabulary τ is defined as follows. We fix a countable set $V = \{x_0, x_1, \ldots\}$ of variables *ranging over elements of the primary part A only.* The **point terms** (defining functions $f : A^k \to A$), the **weight terms** (defining functions $w : A^k \to R$), and the **formulae** (defining relations $R \subseteq A^k$) of $L[\tau]$ are defined inductively as follows:

(1) Point terms are defined in the usual way, by closing the set of variables V under application of function symbols from τ_a.
(2) Weight terms can be built by applying weight function symbols from τ_w to point terms, and function symbols from τ_r to previously defined weight terms. Note that there are no variables ranging over R.
(3) Atomic formulae are equalities of point terms, equalities of weight terms, expressions $Pt_1 \cdots t_r$ containing relations symbols $P \in \tau_a$ and point terms $t_1, \ldots, t_r$, or expressions $Qf_1 \cdots f_r$ containing predicates $Q \in \tau_r$ and weight terms $f_1, \ldots, f_r$.
(4) All the rules of L for building formulae (via propositional connectives, quantifiers, and other operators) may be applied, taking into account the condition that only variables from V may be used.
(5) In addition, we have the **characteristic function rule**: if $\varphi(\overline{x})$ is a formula, then $\chi[\varphi](\overline{x})$ is a weight term.
(6) If τ_w contains **multiset operations**, these provide additional means for building new weight terms. Let $F(\overline{x}, \overline{y})$ be a weight term, $\varphi(\overline{x}, \overline{y})$ a formula (both with free variables among $\overline{x}, \overline{y}$), and Γ a multiset operation. The expression
$$\Gamma_{\overline{x}}(F(\overline{x}, \overline{y}) : \varphi)$$

is then a weight term with free variables $\overline{y}$. (If $\varphi = \mathit{true}$, we simplify this notation to $\Gamma_{\overline{x}} F(\overline{x}, \overline{y})$.)

The semantics for (1)–(4) is the obvious one. A term $\chi[\varphi](\overline{x})$ evaluates to 1 if $\varphi(\overline{x})$ is true, and to 0 otherwise. Finally, let $G(\overline{y})$ be a weight term $\Gamma_{\overline{x}}(F(\overline{x}, \overline{y}) : \varphi)$ formed by application of a multiset operation. The weight term $F(\overline{x}, \overline{y})$ defines, on a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$, a function $F^{\mathfrak{D}} : A^{k+m} \to R$. For any fixed tuple $\overline{b}$, the collection of values $F^{\mathfrak{D}}(\overline{a}, \overline{b})$, as $\overline{a}$ ranges over those tuples such that $\varphi(\overline{a}, \overline{b})$ is *true*, forms a finite multiset

$$(F : \varphi)^{\mathfrak{D}}(\overline{b}) := \{\!\{ F^{\mathfrak{D}}(\overline{a}, \overline{b}) : \overline{a} \in A^k \text{ such that } \mathfrak{D} \models \varphi(\overline{a}, \overline{b}) \}\!\}.$$

The interpretation of $G(\overline{b})$ on $\mathfrak{D}$ is obtained by applying Γ to this multiset, i.e.

$$G^{\mathfrak{D}}(\overline{b}) := \Gamma((F : \varphi)^{\mathfrak{D}}(\overline{b})).$$

Example 3.6.11. (Binary representations.) Consider arithmetic structures with a primary part of the form $\mathfrak{A} = (\{0, \dots, n-1\}, <, P)$ where P is a unary relation. P is interpreted as a bit sequence $u_0 \cdots u_{n-1}$ representing the natural number $\sum_{i=0}^{n-1} u_i 2^i$ (where $u_i = 1$ iff $\mathfrak{A} \models P(i)$). The number represented by P is definable by the term

$$\sum_x \Big(\chi[Px] \prod_y (2 : y < x) \Big).$$

Example 3.6.12. (Counting elements.) On arithmetic structures, first-order logic can count. For any formula $\varphi(\overline{x})$, there is a weight term $\#_{\overline{x}}[\varphi(\overline{x})]$ counting the number of tuples $\overline{a}$ such that $\varphi(\overline{a})$ is true, namely

$$\#_{\overline{x}}[\varphi(\overline{x})] := \sum_{\overline{x}} \chi[\varphi].$$

3.6.4 Metafinite Spectra

Does descriptive complexity theory generalize in a meaningful way from finite to metafinite structures? To give some evidence that such generalizations are indeed possible and fruitful, we focus here on generalizations of Fagin's Theorem to (1) arithmetical structures, and (2) $\mathbb{R}$-structures (see the examples given above).

Recall that Fagin's Theorem says that generalized spectra (or, equivalently, the properties of finite structures that are definable in existential second-order logic) coincide with the complexity class NP. To discuss possible translations to metafinite structures, we need to make precise two notions:

- The notion of a **metafinite spectrum**, i.e. a generalized spectrum of metafinite structures.
- The notion of complexity (in particular, deterministic and non-deterministic polynomial time) in the context of metafinite structures.

For a fixed structure $\mathfrak{R}$, let $M_\tau[\mathfrak{R}]$ denote the class of metafinite structures with a secondary part $\mathfrak{R}$ and vocabulary $\tau = (\tau_a, \tau_r, \tau_w)$ (where, of course, τ_r is the vocabulary of $\mathfrak{R}$). We start with two notions of metafinite spectra.

Definition 3.6.13. A class $\mathcal{K} \subseteq M_\tau[\mathfrak{R}]$ is a **metafinite spectrum** if there exists a first-order sentence ψ of a vocabulary $\tau' \supseteq \tau$ such that $\mathfrak{D} \in \mathcal{K}$ if and only if there exists an expansion $\mathfrak{D}' \in M_{\tau'}[\mathfrak{R}]$ of $\mathfrak{D}$ with $\mathfrak{D}' \models \psi$. (Note that the secondary part is not expanded.) A **primary metafinite spectrum** is defined in a similar way, except that only the primary part of the structures is expanded, and not the set of weight functions. This means that the expanded structures $\mathfrak{D}'$ have the same set of weight functions as $\mathfrak{D}$.

These two notions of metafinite spectra correspond to two variants of **existential second-order logic**. The more restrictive variant allows second-order quantification over primary relations only, whereas the general one allows quantification over weight functions as well. Thus, a primary metafinite spectrum is the class of structures $\mathfrak{D} \in M_\tau[\mathfrak{R}]$ which are models of an existential second-order sentence of the form $\exists R_1 \cdots \exists R_m \psi$, where $R_1, \dots, R_m$ are relation variables over the primary part, and ψ is first-order. Since relations over the primary part can be replaced by their characteristic functions, a metafinite spectrum in the more general sense is the class of models of a sentence $\exists F_1 \cdots \exists F_m \psi$, where the F_i are function symbols ranging over weight functions. We shall see that both notions of metafinite spectra capture (suitable variants of) non-deterministic polynomial-time in certain contexts, but fail to do so in others.

In general, the notion of *complexity* for problems on metafinite strucures depends on the computation model used and on the **cost** (or **size**) associated with the elements of the secondary part. For instance, if the secondary part consists of natural numbers or binary strings, then a natural notion of cost is given by the number of bits. On the other hand, below we shall study complexity over real numbers with respect to the Blum–Shub–Smale model, and there every element of $\mathbb{R}$ will be treated as a basic entity of cost one.

Let $\|r\|$ denote the cost of r. For a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W) \in M_\tau[\mathfrak{R}]$, let $|\mathfrak{D}| := |\mathfrak{A}|$ and let $\max \mathfrak{D} := \max_{w \in W} \max_{\overline{a}} \|w(\overline{a})\|$, the cost of the maximal weight. Assuming $\mathfrak{R}$ and τ to be fixed, then $\|\mathfrak{D}\|$, the cost of representing $\mathfrak{D}$, is polynomially bounded in $|\mathfrak{A}|$ and $\max \mathfrak{D}$ (via a polynomial that depends only on the vocabulary of $\mathfrak{D}$). Since most of the popular complexity classes are invariant under polynomial increase of the relevant input parameters, it therefore makes sense to measure the complexity in terms of $|\mathfrak{D}|$ and $\max \mathfrak{D}$. For instance, an algorithm on a class of metafinite structures runs in polynomial time or in logarithmic space if, for every input $\mathfrak{D}$, the computation terminates in at most $q(|\mathfrak{D}|, \max \mathfrak{D})$ steps, for some polynomial q, or uses at most $O(\log |\mathfrak{D}| + \log \max \mathfrak{D})$ of work space, respectively.

We first discuss arithmetical structures, as described in Example 3.6.9, assuming that the cost of natural numbers is given by the length of their binary representations. So the question is whether, or under what circumstances, NP is captured by the class of metafinite spectra or primary metafinite spectra. The original proof of Fagin's Theorem generalizes to the case of arithmetical structures with weights that are not too large.

Definition 3.6.14. A class $\mathcal{K}$ of metafinite structures has **small weights** if there exists a $k \in \mathbb{N}$ such that $\max \mathfrak{D} \leq |\mathfrak{D}|^k$ for all $\mathfrak{D} \in \mathcal{K}$. As $\max \mathfrak{D}$ stands for the *cost* of the largest weight this means that the values of the weights are bounded by a function $2^{p(|\mathfrak{D}|)}$ for some polynomial p.

We obtain the following first generalization of Fagin's result.

Theorem 3.6.15 (Grädel and Gurevich). *Let $\mathcal{K} \subseteq M_\tau[\mathfrak{N}]$ be a class of arithmetical structures with small weights which is closed under isomorphisms. The following are equivalent:*

(i) $\mathcal{K}$ is in NP.
(ii) $\mathcal{K}$ is a primary generalized spectrum.

Proof. It is obvious that *(ii)* implies *(i)*. The converse can be reduced to Fagin's Theorem as follows. We assume that for every structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{N}, W)$ in $\mathcal{K}$, we have that $\max \mathfrak{D} \leq n^k$, where $n = |\mathfrak{D}| = |\mathfrak{A}|$; further, we suppose without loss of generality, that an ordering $<$ on A is available (otherwise we expand the vocabulary with a binary relation $<$ and add a conjunct $\beta(<)$ asserting that $<$ is a linear order). We can then identify A^k with the initial subset $\{0, \dots, n^k - 1\}$ of $\mathbb{N}$, viewed as bit positions of the binary representations of the weights of $\mathfrak{D}$. With every $\mathfrak{D} \in \mathcal{K}$ we associate a finite structure $\mathfrak{D}_f$ by expanding the primary part $\mathfrak{A}$ as follows: for every weight function $w \in W$ of arity j, we add a new relation P_w of arity $j + k$, where

$$P_w := \{(\overline{a}, \overline{t}) : \text{ the } \overline{t}\text{th bit of } w(\overline{a}) \text{ is } 1\}.$$

Then $\mathcal{K}$ is in NP if and only if $\mathcal{K}_f = \{\mathfrak{D}_f : \mathfrak{D} \in \mathcal{K}\}$ is an NP-set of finite structures, and, in fact, we can choose the encodings in such a way that $\mathfrak{D}$ and $\mathfrak{D}_f$ are represented by the same binary string. Thus, if $\mathcal{K}$ is in NP, then, by Fagin's Theorem, $\mathcal{K}_f$ is a generalized spectrum, defined by a first-order sentence ψ.

As in Example 3.6.11, one can construct a first-order sentence α (whose vocabulary consists of the weight functions $w \in \tau_w$ and the corresponding primary relations P_w) which expresses the assertion that the P_w encode the weight functions w in the sense defined above. Then $\psi \wedge \alpha$ is a first-order sentence witnessing that $\mathcal{K}$ is a primary metafinite spectrum. □

The above result also holds for arithmetical structures without multiset operations. However, without the restriction that the weights are small, it is no longer true that every NP-set is a primary metafinite spectrum. If we have inputs with huge weights compared with the primary part, then relations over the primary part cannot code enough information to describe computations that are bounded by a polynomial in the length of the weights.

It is tempting to use unrestricted metafinite spectra instead. However, metafinite spectra in the general sense capture a much larger class than NP.

Theorem 3.6.16 (Grädel and Gurevich). *On arithmetical structures, metafinite spectra capture the recursively enumerable sets.*

We sketch the proof here. It is not difficult to show that every metafinite spectrum of arithmetical structures is recursively enumerable. For the converse, we first note that any tuple $\overline{a} \in \mathbb{N}^k$ can be viewed as an arithmetical structure with an empty primary vocabulary and k nullary weight functions $a_1, \dots, a_k$. Thus an arithmetical relation $S \subseteq \mathbb{N}^k$ can be viewed as a special class of arithmetical structures. We show first that every recursively enumerable set $S \subseteq \mathbb{N}^k$ is a metafinite spectrum. In particular, there exist undecidable metafinite spectra.

By Matijasevich's Theorem (see [83]), every recursively enumerable set $S \subseteq \mathbb{N}^k$ is Diophantine, i.e. can be represented as

$$S = \{\overline{a} \in \mathbb{N}^k : \text{ there exists } b_1, \dots, b_m \in \mathbb{N} \text{ such that } Q(\overline{a}, \overline{b}) = 0\}$$

for some polynomial $Q \in \mathbb{Z}[x_1, \dots, x_k, y_1, \dots, y_m]$. Let $P, P' \in \mathbb{N}[\overline{x}, \overline{y}]$ such that $Q(\overline{x}, \overline{y}) = P(\overline{x}, \overline{y}) - P'(\overline{x}, \overline{y})$. Thus S is a metafinite spectrum; the desired first-order sentence uses additional weight functions $b_1, \dots, b_m$ and asserts that $P(\overline{a}, \overline{b}) = P'(\overline{a}, \overline{b})$.

This can be extended to any recursively enumerable class of arithmetical structures, with an arbitrary vocabulary. To see this, we encode structures $\mathfrak{D} \subseteq M_\tau[\mathfrak{N}]$ by tuples $c(\mathfrak{D}) \in \mathbb{N}^k$, where k depends only on τ. (In fact, it is no problem to reduce k to 1.) Similarly to the case of finite structures, an encoding involves the selection of a linear order on the primary part. In fact, it is often more convenient to have a **ranking** of the primary part rather than just a linear ordering.

Definition 3.6.17. Suppose that $\mathfrak{R}$ contains a copy of $(\mathbb{N}, <)$. A **ranking** of a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ is a bijection $r : A \to \{0, \dots, n-1\} \subseteq R$. A class $\mathcal{K} \subseteq M_\tau[\mathfrak{R}]$ is **ranked** if τ contains a weight function r whose interpretation on every $\mathfrak{D} \in \mathcal{K}$ is a ranking.

The **Coding Lemma** for arithmetical structures [48] says that for every vocabulary τ there exists an encoding function that associates with every ranked arithmetical τ-structure $\mathfrak{D}$ a tuple $\mathit{code}(\mathfrak{D}) \in \mathbb{N}^k$ with the following properties:

(1) *code* is definable by first-order terms.
(2) The primary part and the weight functions of $\mathfrak{D}$ can be reconstructed from $\mathit{code}(\mathfrak{D})$ in polynomial time.
(3) There exists a polynomial $p(n, m)$ such that $c_i(\mathfrak{D}) \leq 2^{p(|\mathfrak{D}|, \max \mathfrak{D})}$ for every $i \leq k$.

Now let $\mathcal{K} \subseteq M_\tau[\mathfrak{N}]$ be recursively enumerable. The set

$$code(\mathcal{K}) := \{code(\mathfrak{D}, r) : \mathfrak{D} \in \mathcal{K},\ r \text{ is a ranking of } \mathfrak{D}\} \subseteq \mathbb{N}^k$$

is then also recursively enumerable and therefore Diophantine. The desired first-order sentence ψ uses, besides the symbols of τ, a unary weight function r and nullary weight functions $b_1, \ldots, b_m$ and expresses the assertions (i) that r is a ranking and (ii) that $Q(code(\mathfrak{D}, r), \overline{b})) = 0$ for a suitable polynomial $Q \in \mathbb{Z}[x_1, \ldots, x_k, y_1, \ldots, y_m]$ defining $code(\mathcal{K})$.

3.6.5 Descriptive Complexity over the Real Numbers

There are other contexts in which metafinite spectra do indeed capture (a suitable notion of) non-deterministic polynomial time. An important example are computations over the real numbers based on the model of Blum, Shub, and Smale.

Computation over $\mathbb{R}$

In 1989 Blum, Shub, and Smale [15] introduced a model for computations over the real numbers (and other rings as well), which is now usually called the BSS machine. The important difference from, say, the Turing model is that real numbers are treated as basic entities and that arithmetic operations on the reals are performed in a single step, independently of the magnitude or complexity of the numbers involved. In particular, the model abstracts from the problems that in actual computers real numbers have to be approximated by bit sequences, that the complexity of arithmetic operations depends on the length of these approximate representations, that rounding errors occur, and that exact testing for 0 is impossible in practice. Similar notions of computations over arbitrary fields or rings had been investigated earlier in *algebraic complexity theory* (see [22] for a comprehensive treatment). A novelty of the approach of Blum, Shub, and Smale is that their model is uniform (for all input lengths) whereas the ideas explored in algebraic complexity (such as straight-line programs, arithmetic circuits, and decision trees) are typically non-uniform. One of the main purposes of the BSS approach was to create a uniform complexity theory dealing with problems that have an analytical and topological background, and to show that certain problems remain hard even if arbitrary reals are treated as basic entities.

Many basic concepts and fundamental results of classical computability and complexity theory reappear in the BSS model: the existence of universal machines, the classes $\mathrm{P}_\mathbb{R}$ and $\mathrm{NP}_\mathbb{R}$ (real analogues of P and NP), and the existence of $\mathrm{NP}_\mathbb{R}$-complete problems. Of course, these ideas appear in a different form, with a strong analytical flavour: typical examples of undecidable, recursively enumerable sets are complements of certain Julia sets, and the first problem that was shown to be $\mathrm{NP}_\mathbb{R}$-complete is the question of whether a given multivariate polynomial of degree four has a real root [15]. As in the classical setting, all problems in the class $\mathrm{NP}_{\mathbb{R}}$ are decidable within exponential time (but this is not as trivial as in the classical case), and the $\mathrm{P}_{\mathbb{R}}$ versus $\mathrm{NP}_{\mathbb{R}}$ question is one of the major open problems.

However, there also are many differences between classical and real complexity theory. Just to mention a few, we note that the meaning of space resources seems to be very different, that certain separation results between complexity classes can be established (such as $\mathrm{NC}_{\mathbb{R}} \subsetneq \mathrm{P}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{R}} \subsetneq \mathrm{EXP}_{\mathbb{R}}$) whose analogues in the classical theory are open, and that some discrete problems seem to change their complexity behaviour when considered in the BSS model. For a detailed treatment we refer the interested reader to the book [14].

The BSS Model

Let $\mathbb{R}^* := \bigcup_{k\in\mathbb{N}} \mathbb{R}^k$, or (almost) equivalently, the set of functions $X : \mathbb{N} \to \mathbb{R}$ with $X(n) = 0$ for all but finitely many n. For any $X \in \mathbb{R}^*$, we call $|X| := \max\{n : X(n) \neq 0\}$ the **length** of X. Note that $\mathbb{R}^* \times \mathbb{R}^*$ can be identified with $\mathbb{R}^*$ in a natural way by concatenation. A **Blum–Shub–Smale machine** – in what follows called a BSS machine – is essentially a Random Access Machine over $\mathbb{R}$ which can evaluate rational functions at unit cost and whose registers can store arbitrary real numbers.

Definition 3.6.18. A *BSS machine* M over $\mathbb{R}$ is given by a finite set I of instructions labelled by $0, \ldots, N$. The input and output spaces are subsets of $\mathbb{R}^*$. A configuration is a quadruple $(k, r, w, x) \in I \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}^*$, where k is the instruction currently being executed, r and w are the numbers of the so called 'copy registers' (see below) and x describes the content of the registers of the machine. Given an input $x \in \mathbb{R}^*$, the computation is started with a configuration $(0, 0, 0, x)$. If a configuration (k, r, w, x) with $k = N$ is reached, the computation stops; in that case the value of x is the output computed by the machine. The instructions of M are of the following types:

- Computation. An instruction k of this type performs an update $x_0 \leftarrow g_k(x)$ of the first register, where g_k is a rational function on $\mathbb{R}^m$ (for some m). Simultaneously, the copy registers may be updated by rules $r \leftarrow r + 1$ or $r \leftarrow 0$, and similarly for w. The other registers remain unchanged. The next instruction will be $k + 1$.
- Branch. k: **if** $x_0 \geq 0$ **goto** ℓ **else goto** $k+1$**.** The contents of the registers remain unchanged.
- Copy. $k : x_w \leftarrow x_r$, i.e. the content of the 'read register' is copied into the 'write register'. The next instruction is $k + 1$; all other registers remain unchanged.

A set $L \subseteq \mathbb{R}^*$ is in $\mathrm{P}_{\mathbb{R}}$ if there exists a BSS machine whose running time on every $X \in \mathbb{R}^*$ is bounded by a polynomial in $|X|$, and which accepts X if and only if $X \in L$. The analogue of NP is the class $\mathrm{NP}_{\mathbb{R}}$. A set $L \subseteq \mathbb{R}^*$ is in

$\mathrm{NP}_{\mathbb{R}}$ if there exists a set $L' \in \mathrm{P}_{\mathbb{R}}$ and a constant k such that $L = \{X \in \mathbb{R}^* : (\exists Y \in \mathbb{R}^*)(|Y| \leq |X|^k \wedge (X, Y) \in L')\}$. Equivalently, $\mathrm{NP}_{\mathbb{R}}$ can be defined as the class of problems over $\mathbb{R}^*$ that are decidable in polynomial time by a *non-deterministic* BSS machine, i.e. a BSS machine that can non-deterministically guess real numbers $Y \in \mathbb{R}$ at unit cost.

Encodings. Recall that $\mathbb{R}$-structures are metafinite structures $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ with a second sort $\mathfrak{R} = (\mathbb{R}, +, -, \cdot, /, \leq, (c_r)_{r \in \mathbb{R}})$. We want to relate decision problems for $\mathbb{R}$-structures (described by logical formulae) to decision problems on $\mathbb{R}^*$ (decided by BSS-machines). We first consider an example.

Example 3.6.19. (4-Feasibility.) The first problem that was shown to be $\mathrm{NP}_{\mathbb{R}}$-complete was the problem of whether a real polynomial of degree at most four in n unknowns (where n varies with the input) has a real zero. This problem can be considered as a decision problem on $\mathbb{R}$-structures as follows. Let $A = \{0, \ldots, n\}$. The coefficients of a *homogeneous* polynomial $g \in \mathbb{R}[X_0, \ldots, X_n]$ can be coded via a function $C : A^4 \to \mathbb{R}$, such that

$$g = \sum_{0 \leq i,j,k,\ell \leq n} C(i, j, k, \ell) X_i X_j X_k X_\ell .$$

We obtain an arbitrary (not necessarily homogeneous) polynomial $f \in \mathbb{R}[X_1, \ldots, X_n]$ of degree four by setting $X_0 = 1$ in g. Thus, every multivariate polynomial f of degree at most four is represented by the $\mathbb{R}$-structure $(\mathfrak{A}, \mathfrak{R}, \{C\})$, where $\mathfrak{A} = (\{0, \ldots, n\}, <, 0, n)$ and C is a function from A^4 into $\mathbb{R}$.

Observe that $\mathbb{R}^*$ can be viewed as the class of all $\mathbb{R}$-structures where the primary part is a finite linear order $(\{0, \ldots, n-1\}, <)$, and W consists of a single unary function $X : \{0, \ldots, n-1\} \to \mathbb{R}$. Hence decision problems on $\mathbb{R}^*$ can be regarded as a special case of decision problems on $\mathbb{R}$-structures (in the same way as words can be considered as special cases of finite structures). Conversely, $\mathbb{R}$-structures $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ can be encoded in $\mathbb{R}^*$. We choose a ranking on A and replace all functions and relations in the primary part by the appropriate characteristic functions $\chi : A^k \to \{0, 1\} \subseteq \mathbb{R}$. This gives a structure whose primary part is a plain set A, with functions $X_1, \ldots, X_t$ of the form $X_i : A^k \to \mathbb{R}$ and with the ranking $r : A \to \mathbb{R}$. Each of the functions X_i can be represented by a tuple $x_0, \ldots, x_{m-1} \in \mathbb{R}^m$, where $m = |A|^k$ and $x_i = X(\overline{a}(i))$, and where $\overline{a}(i)$ is the ith tuple in A^k with respect to the lexicographic order induced by r. The concatenation of these tuples gives an encoding $code(\mathfrak{D}, r) \in \mathbb{R}^*$ (which depends on the ranking r that was chosen).

Obviously, for structures $\mathfrak{D}$ of a fixed finite signature, the length of $code(\mathfrak{D}, r)$ is bounded by some polynomial n^ℓ, where $n = |\mathfrak{D}|$ and ℓ depends only on the signature. Thus we can also view $code(\mathfrak{D}, r) = (x_0, \ldots, x_{n^\ell - 1})$ as a single function $X_{\mathfrak{D}} : A^\ell \to \mathbb{R}$, where $X(\overline{a}(i)) = x_i$ for all $i < n^\ell$. Thus, encoding an $\mathbb{R}$-structure in $\mathbb{R}^*$ basically means representing the whole structure by a single function (of appropriate arity) from $\{0, \ldots, n-1\}$ into $\mathbb{R}$.

Furthermore, this encoding is first-order definable in the following sense.

Lemma 3.6.20. *For every signature τ, there is a first-order formula $\beta(X, r)$ of signature $\tau \cup \{X, r\}$ such that, for all $\mathbb{R}$-structures $\mathfrak{D}$ of signature τ, for all rankings r, and for all functions X,*

$$(\mathfrak{D}, X, r) \models \beta(X, r) \quad \textit{iff} \quad X = \textit{code}(\mathfrak{D}, r).$$

As in the case of finite structures, we say that a class $\mathcal{K}$ of $\mathbb{R}$-structures is in the complexity class $\mathrm{P}_{\mathbb{R}}$ or $\mathrm{NP}_{\mathbb{R}}$ if the set of its encodings is. Recall that a metafinite spectrum of $\mathbb{R}$-structures is a set $\mathcal{K}$ of $\mathbb{R}$-structures that is definable by an existential second-order sentence $\exists Y_1 \cdots \exists Y_r \psi$, where ψ is first-order and the variables Y_i range over weight functions $Y_i : A^k \to \mathbb{R}$. Fagin's Theorem now has the following analogue in the real setting.

Theorem 3.6.21 (Grädel and Meer). *Let $\mathcal{K}$ be a class of $\mathbb{R}$-structures. Then $\mathcal{K} \in \mathrm{NP}_{\mathbb{R}}$ if and only if $\mathcal{K}$ is a metafinite spectrum.*

Proof. It is easy to see that metafinite spectra are in $\mathrm{NP}_{\mathbb{R}}$. Suppose that $\psi = \exists Y_1 \cdots Y_r \varphi$. Given an input structure $\mathfrak{D}$, we guess assignments for all functions Y_i and evaluate φ on $(\mathfrak{D}, Y_1, \ldots, Y_r)$ in polynomial time.

For the converse, let $\mathcal{K} \in \mathrm{NP}_{\mathbb{R}}$ and let $\mathcal{K}'$ be the corresponding problem in $\mathrm{P}_{\mathbb{R}}$, with $\mathcal{K} = \{\mathfrak{D} : \exists Y((\mathfrak{D}, Y) \in \mathcal{K}')\}$. Let M be a polynomial-time BSS machine deciding $\mathcal{K}'$, and let m be a natural number such that M stops on encodings of $(\mathfrak{D}, Y)$ after less than n^m steps and uses at most $n^m - 3$ registers, where $n = |\mathfrak{D}|$.

We first suppose that we have a ranking $r : A \to \mathbb{R}$ available. From r, the induced (lexicographic) ranking $r_m : A^m \to R$ is first-order definable: we can identify the element in A of maximal rank and thus have the number term n available; we can then use $r_m(\overline{t})$ as an abbreviation for

$$r(t_1)n^{m-1} + \cdots r(t_{m-1})n + r(t_m).$$

We can then identify A^m with the initial subset $\{0, \ldots, n^m - 1\}$ of $\mathbb{N}$. Thus, in the formulae to be constructed below, m-tuples $\overline{t} = t_1, \ldots, t_m$ of variables are considered to range over natural numbers $t < n^m$. Conditions such as $\overline{t} = 0$ or $\overline{t} = \overline{s} + \overline{s}'$ can then be expressed by first-order formulae of vocabulary $\{r\}$.

The computation of M for a given input $\textit{code}(\mathfrak{D}, Y)$ can be represented by a function $Z : A^{2m} \to \mathbb{R}$ as follows:

- $Z(0, \overline{t})$ is the instruction executed by M at time $\overline{t}$.
- $Z(1, \overline{t})$ and $Z(2, \overline{t})$ are the indices of the read and write registers of M at time $\overline{t}$.
- $Z(\overline{j} + 3, \overline{t})$ is the content of register $\overline{j}$ at time $\overline{t}$.

We construct a first-order formula ψ with the property that, for all ranked structures $(\mathfrak{D}, Y)$ and all Z, we have that $(\mathfrak{D}, Y, Z) \models \psi$ iff Z represents an accepting computation of M for $\textit{code}(\mathfrak{D}, Y)$.

We first have to express the assertion that at time $t = 0$, the function Z encodes the input configuration of M on $(\mathfrak{D}, Y)$. Thus we need a subformula stating that $Z(i, 0) = 0$ for $i = 0, 1, 2$ and that the values $Z(\overline{j} + 3, 0)$ encode the input $(\mathfrak{D}, Y)$. By Lemma 3.6.20 this can be expressed in first-order logic.

Second, we have to ensure that for every $t < n^m - 1$, if the sequence $\langle Z(\overline{j}, \overline{t}) : \overline{j} = 0, \dots, n^m - 1\rangle$ represents a configuration of M, then the sequence of values $Z(\overline{j}, \overline{t}{+}1)$ represents the successor configuration. The formula asserting this has the form

$$\forall \overline{t} \bigwedge_{k=0}^{N} (Z(0, \overline{t}) = k \;\rightarrow\; \varphi_k)$$

where φ_k describes transitions performed by the instruction k.

Consider for example a computation instruction $k : x_0 \leftarrow g(x_0, \dots, x_\ell)$, and assume in addition that it increases the index of the read register by 1 and sets the index of the write register back to 0. The formula φ_k then has to express the following:

- $Z(0, \overline{t} + 1) = k + 1$ (the next instruction is $k + 1$);
- $Z(1, \overline{t} + 1) = Z(1, \overline{t}) + 1$ (the read register index is increased by 1);
- $Z(2, \overline{t} + 1) = 0$ (the write register index is set back to 0);
- $Z(3, \overline{t} + 1) = g(Z(3, \overline{t}), Z(4, \overline{t}), \dots, Z(\ell + 3, \overline{t}))$ (into register 0, M writes the result of applying the rational function g to the register contents at time $\overline{t}$).
- $Z(\overline{j}, \overline{t} + 1) = Z(\overline{j}, \overline{t})$ for all $\overline{j} > 3$ (the other registers remain unchanged).

Clearly, these conditions are first-order expressible. It should be noted that whenever $f_0, \dots, f_\ell$ are number terms and $g : \mathbb{R}^\ell \rightarrow \mathbb{R}$ is a rational function, then $g(f_0, \dots, f_\ell)$ is also a number term.

For another example illustrating the explicit use of the embedding function, consider a copy instruction $k : x_w \leftarrow x_r$. Here the formula has to express (besides the updating of the instruction number, etc. which is done as above), the assertion that the content of the register $Z(2, \overline{t})$ at time $\overline{t}{+}1$ is the same as the content of the register $Z(1, \overline{t})$ at time $\overline{t}$. This is expressed by the formula

$$\forall \overline{j} \forall \overline{j}' ([Z(1, \overline{t}) = r_m(\overline{j}) \wedge Z(2, \overline{t}) = r_m(\overline{j}')]$$
$$\rightarrow Z(\overline{j}' + 3, \overline{t} + 1) = Z(\overline{j} + 3, \overline{t})).$$

To express the assertion that M accepts its input, we just have to say that $Z(3, n^m - 1) = 1$ (by convention, the result of the computation, if it is a single number, is stored in register 0).

Combining all these subformulae in the appropriate way, we obtain the desired formula ψ. It then follows that for all structures $\mathfrak{D}$,

$$\mathfrak{D} \in \mathcal{K} \quad \text{iff } \mathfrak{D} \models (\exists Y)(\exists Z)\psi,$$

which proves the theorem for the case of ranked structures.

Finally, we do away with the assumption that the input structures are ranked. If no ranking is given on the input structures $\mathfrak{D}$, we can introduce one by existentially quantifying over the function r and adding a conjunct $\alpha(r)$ which asserts that r is one–one and that, for all t with $r(t) \neq 0$, there exists an element s such that $r(s) + 1 = r(t)$. It follows that

$$\mathcal{K} = \{\mathfrak{D} : \mathfrak{D} \models (\exists r)(\exists Y)(\exists Z)(\alpha \wedge \psi)\}.$$

□

Example 3.6.22. (Logical description of 4-Feasibility.) An existential second-order sentence for the 4-feasibility problem quantifies two functions $X : A \to \mathbb{R}$ and $Y : A^4 \to \mathbb{R}$ where $X(1), \dots, X(n)$ describes the zero and $Y(u)$ is the partial sum of all monomials up to $u \in A^4$ in $f(X_1, \dots, X_n)$ (according to the lexicographical order on A^4). Thus the 4-feasibility problem is described by the sentence

$$\begin{aligned}\psi := (\exists X)(\exists Y)\Big(& Y(\overline{0}) = C(\overline{0}) \wedge Y(\overline{n}) = 0 \wedge \forall \overline{u}(\overline{u} \neq \overline{0} \to \\ & Y(\overline{u}) = Y(\overline{u} - 1) + C(\overline{u}) \textstyle\prod_{i=1}^{4} X(u_i))\Big).\end{aligned}$$

Indeed, $\mathfrak{D} \models \psi$ if and only if the polynomial f of degree four defined by $\mathfrak{D}$ has a real zero.

Capturing Results for Other Complexity Classes

By combining the general ideas of descriptive complexity theory on finite structures with the approach described here, one can find logical characterizations for many other complexity levels, notably for polynomial time, provided that the given $\mathbb{R}$-structures are ranked (i.e. an ordering on the finite part is available). This is carried out in some detail in [30, 53].

Acknowledgements

I am grateful for comments, corrections, and other contributions to Dietmar Berwanger, Achim Blumensath, Anuj Dawar, Phokion Kolaitis, Stephan Kreutzer, Antje Nowack, and Martin Otto.

3.7 Appendix: Alternating Complexity Classes

Alternating algorithms are a generalization of non-deterministic algorithms, based on two-player games. Indeed, one can view non-deterministic algorithms as the restriction of alternating algorithms to solitaire (i.e. one-player) games.

Since complexity classes are mostly defined in terms of Turing machines, we focus on the model of alternating Turing machines. But note that alternating algorithms can be defined in terms of other computational models, also.

Definition 3.7.1. An **alternating Turing machine** is a non-deterministic Turing machine whose state set Q is divided into four classes $Q_\exists$, $Q_\forall$, Q_{acc}, and Q_{rej}. This means that there are existential, universal, accepting and rejecting states. States in $Q_{acc} \cup Q_{rej}$ are final states. A configuration of M is called existential, universal, accepting, or rejecting according to its state.

The computation graph $G_{M,x}$ of an alternating Turing machine M for an input x is defined in the same way as for a non-deterministic Turing machine. Nodes are configurations (instantaneous descriptions) of M, there is a distinguished starting node $C_0(x)$ which is the input configuration of M for input x, and there is an edge from configuration C to configuration C' if, and only if, C' is a successor configuration of C. Recall that for *non-deterministic* Turing machines, the acceptance condition is given by the REACHABILITY problem: M accepts x if, and only if, in the graph $G_{M,x}$ some accepting configuration C_a is reachable from $C_0(x)$. For *alternating* Turing machines, acceptance is defined by the GAME problem (see Sect. 3.1.3): the players here are called $\exists$ and $\forall$, where $\exists$ moves from existential configurations and $\forall$ from universal ones. Further, $\exists$ wins at accepting configurations and loses at rejecting ones. By definition, M accepts x if, and only if, Player $\exists$ has a winning strategy from $C_0(x)$ for the game on $G_{M,x}$.

Complexity Classes

Time and space complexity are defined as for nondeterministic Turing machines. For a function $F : \mathbb{N} \to \mathbb{R}$, we say that an alternating Turing machine M is F-time-bounded if for all inputs x, all computation paths from $C_0(x)$ terminate after at most $F(|x|)$ steps. Similarly, M is F-space-bounded if no configuration of M that is reachable from $C_0(x)$ uses more than $F(|x|)$ cells of work space. The complexity classes ATIME(F) and ASPACE(F) contain all problems that are decidable by, respectively, F-time bounded and F-space bounded alternating Turing machines.

The following classes are of particular interest:

- ALOGSPACE = ASPACE($O(\log n)$),
- APTIME = $\bigcup_{d\in\mathbb{N}}$ ATIME(n^d),
- APSPACE = $\bigcup_{d\in\mathbb{N}}$ ASPACE(n^d).

Alternating Versus Deterministic Complexity

There is a general slogan that parallel time complexity coincides with sequential space complexity. Indeed, by standard techniques of complexity theory, one can easily show that, for well-behaved (i.e. space-constructible) functions F, ATIME(F) $\subseteq$ DSPACE(F^2) and DSPACE(F) $\subseteq$ NSPACE(F) $\subseteq$ ATIME(F^2) (see [9] for details). In particular,

- APTIME = PSPACE;
- AEXPTIME = EXPSPACE.

On the other hand, alternating space complexity corresponds to exponential deterministic time complexity.

Theorem 3.7.2. *For any space-constructible function* $F(n) \geq \log n$*, we have that* ASPACE(F) = DTIME($2^{O(F)}$).

Proof. The proof is closely associated with the GAME problem. For any F-space-bounded alternating Turing machine M, one can, given an input x, construct the computation graph $G_{M,x}$ in time $2^{O(F(|x|)}$ and then solve the GAME problem in order to decide the acceptance of x by M.

For the converse, we shall show that for any $G(n) \geq n$ and any constant c, DTIME(G) $\subseteq$ ASPACE($c \cdot \log G$).

Let $L \in$ DTIME(G). There is then a deterministic one-tape Turing machine M that decides L in time G^2. Let $\Gamma = \Sigma \cup (Q \times \Sigma) \cup \{*\}$ and $t = G^2(n)$. Every configuration $C = (q, i, w)$ (in a computation on some input of length n) can be described by a word

$$\underline{c} = *w_0 \cdots w_{i-1}(qw_i)w_{i+1} \cdots w_t* \in \Gamma^{t+2}.$$

The ith symbol of the successor configuration depends only on the symbols at positions $i-1$, i, and $i+1$. Hence, there is a function $f_M : \Gamma^3 \to \Gamma$ such that, whenever symbols a_{-1}, a_0, and a_1 are at positions $i-1$, i and $i+1$ of some configuration $\underline{c}$, the symbol $f_M(a_{-1}, a_0, a_1)$ will be at position i of the successor configuration $\underline{c'}$.

The following alternating algorithm A decides L:

```
Input: $x$
Existential step:  guess $s \leq t$,
                   guess $(q^+a) \in Q_{acc} \times \Sigma$ , $i \in \{0, \ldots, s\}$
                   $b := (q^+a)$
for $j = 1 \ldots s$ do
    Existential step: guess $a_{-1}, a_0, a_1 \in \Gamma^3$
                      verify that $f_M(a_{-1}, a_0, a_1) = b$. If not, reject.
    Universal step:   choose $k \in \{-1, 0, 1\}$
                      $b := a_k$
                      $i := i + k$
    od
if $i$th symbol of input configuration of $M$ on $x$ equals $b$ then accept
                                                                 else reject.
```

The algorithm A needs space $O(\log G(n))$. If M accepts the input x, then Player $\exists$ has the following winning strategy for the game on $C_{A,x}$: the value chosen for s is the time at which M accepts x, and $(q^+a), i$ are chosen so that the configuration of M at time s is of the form $*w_0 \cdots w_{i-1}(q^+a)w_{i+1} \cdots w_t*$. At the jth iteration of the loop (that is, at configuration $s-j$), the symbols at positions $i-1, i, i+1$ of the configuration of M at time $s-j$ are chosen for a_{-1}, a_0, a_1.

Conversely, if M does not accept the input x, the ith symbol of the configuration at time s is not (q^+a). The following holds for all j: if, in the jth iteration of the loop, Player $\exists$ chooses a_{-1}, a_0, a_1, then either $f(a_{-1}, a_0, a_1) \neq b$, in which case Player $\exists$ loses immediately, or there is at least one $k \in \{-1, 0, 1\}$ such that the $(i+k)$th symbol of the configuration at time $s-j$ differs from a_k. Player $\forall$ then chooses exactly this k. At the end, a_k will then be different from the ith symbol of the input configuration, so Player $\forall$ wins.

Hence A accepts x if, and only if, M does so. □

In particular, it follows that

- ALOGSPACE = PTIME;
- APSPACE = EXPTIME.

The relationship between the major deterministic and alternating complexity classes is summarized by the following diagram:

$$\begin{array}{ccccccccccc}
\text{LOGSPACE} \subseteq & \text{PTIME} & \subseteq & \text{PSPACE} & \subseteq & \text{EXPTIME} & \subseteq & \text{EXPSPACE} & \subseteq & \dots \\
 & \| & & \| & & \| & & \| & & \\
 & \text{ALOGSPACE} & \subseteq & \text{APTIME} & \subseteq & \text{APSPACE} & \subseteq & \text{AEXPTIME} & \subseteq & \dots
\end{array}$$

Alternating Logarithmic Time

For time bounds $F(n) < n$, the standard model of alternating Turing machines needs to be modified a little by an indirect access mechanism. The machine writes down, in binary, an address i on an separate index tape to access the ith symbol of the input. Using this model, it makes sense to define, for instance, the complexity class ALOGTIME = ATIME($O(\log n)$).

Example 3.7.3. Construct an ALOGTIME algorithm for the set of palindromes (i.e., words that are same when read from right to left and from left to right).

Important examples of problems in ALOGTIME are

- the model-checking problem for propositional logic;
- the data complexity of first-order logic.

The results mentioned above relating alternating time and sequential space hold also for logarithmic time and space bounds. Note, however, that these do not imply that ALOGTIME = LOGSPACE, owing to the quadratic overheads. It is known that ALOGTIME $\subseteq$ LOGSPACE, but the converse inclusion is an open problem.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991.
3. H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.
4. A. Arnold. The mu-calculus alternation-depth is strict on binary trees. *RAIRO Informatique Théorique et Applications*, 33:329–339, 1999.
5. A. Arnold and D. Niwiński. *Rudiments of μ-Calculus.* North-Holland, 2001.
6. C. Ash and J. Knight. *Computable Structures and the Hyperarithmetical Hierarchy.* Elsevier, 2000.
7. G. Asser. Das Repräsentantenproblem im Prädikatenkalkül der ersten Stufe mit Identität. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 1:252–263, 1955.
8. L. Babai, P. Erdös, and S. Selkow. Random graph isomorphism. *SIAM Journal of Computing*, 9:628–635, 1980.
9. J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II.* Springer, 1990.
10. M. Benedikt, L. Libkin, T. Schwentick, and L. Segoufin. Definable relations and first-order query languages over strings. *Journal of the ACM*, 50:694–751, 2003.
11. D. Berwanger and A. Blumensath. The monadic theory of tree-like structures. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logic, and Infinite Games.* Springer, 2002.
12. H. Björklund, S. Sandberg, and S. Vorobyov. Memoryless determinacy of parity and mean payoff games: A simple proof. *Theoretical Computer Science*, 310:365–378, 2003.
13. A. Blass and Y. Gurevich. Existential fixed point logic. In E. Börger, editor, *Computation Theory and Logic*, Lecture Notes in Computer Science, No. 270, pages 20–36. Springer, 1987.
14. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer, 1998.
15. L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers. *Bulletin of AMS*, 21:1–46, 1989.
16. A. Blumensath. Automatic structures. Diplomarbeit, RWTH Aachen, 1999.
17. A. Blumensath. Prefix-recognisable graphs and monadic second-order logic. Technical Report AIB-06-2001, RWTH Aachen, 2001.
18. A. Blumensath and E. Grädel. Automatic structures. In *Proc. 15th IEEE Symp. on Logic in Computer Science*, pages 51–62, 2000.
19. A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, 37:641–674, 2004.
20. J. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195:133–153, 1998.
21. J. Bradfield and C. Stirling. Modal logics and mu-calculi. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 293–332. Elsevier, 2001.
22. P. Bürgisser, M. Clausen, and A. Shokrollahi. *Algebraic Complexity Theory.* Springer, 1997.

23. J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.
24. A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of FSTTCS*, Lecture Notes in Computer Science, No. 2914, Springer, 2003.
25. D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Automata, Languages and Programming, 23rd International Colloquium, ICALP96*, Lecture Notes in Computer Science, No. 1099, pages 194–205. Springer, 1996.
26. D. Caucal. On infinite terms having a decidable monadic theory. In *Proceedings of 27th International Symposium on Mathematical Foundations of Computer Science MFCS 02*, Lecture Notes in Computer Science, No. 2420, pages 165–176. Springer, 2002.
27. B. Courcelle. The monadic second-order logic of graphs II: Infinite graphs of bounded width. *Mathematical Systems Theory*, 21:187–221, 1989.
28. B. Courcelle. The monadic second-order logic of graphs IX: Machines and their behaviours. *Theoretical Computer Science*, 151:125–162, 1995.
29. B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.
30. F. Cucker and K. Meer. Logic which capture complexity classes over the reals. *Journal of Symbolic Logic*, 64:363–390, 1999.
31. E. Dahlhaus. Skolem normal forms concerning the least fixed point. In E. Börger, editor, *Computation Theory and Logic*, Lecture Notes in Computer Science, No. 270, pages 101–106, Springer, 1987.
32. A. Dawar. Generalized quantifiers and logical reducibilities. *Journal of Logic and Computation*, 5:213–226, 1995.
33. A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logic. *ACM Transactions on Computational Logic*, 5:282–315, 2004.
34. A. Dawar, E. Grädel, and S. Kreutzer. Backtracking games and inflationary fixed points. *Theoretical Computer Science*, 350:174–187, 2006.
35. A. Dawar and L. Hella. The expressive power of finitely many generalized quantifiers. *Information and Computation*, 123:172–184, 1995.
36. W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.
37. S. Dziembowski. Bounded-variable fixpoint queries are PSPACE-complete. In *10th Annual Conference on Computer Science Logic CSL 96, Selected papers*, Lecture Notes in Computer Science, No. 1258, pages 89–105. Springer, 1996.
38. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*, 2nd edition edition. Springer, 1999.
39. A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.
40. A. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, 1991.
41. D. Epstein, J. Cannon, D. Holt, S. Levy, M. Paterson, and W. Thurston. *Word Processing in Groups.* Jones and Bartlett, Boston, 1992.
42. Yu. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel. *Handbook of Recursive Mathematics.* North-Holland, 1998.

43. R. Fagin. Generalised first order spectra and polynomial time recognizable sets. In R. Karp, editor, *Complexity of Computation. SIAM-AMS Proceedings 7*, pages 43–73, 1974.
44. B. Farb. Automatic groups: A guided tour. *L'Enseignement Mathématique*, 38:291–313, 1992.
45. G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Transactions on Computational Logic*, 3:42–79, 2002.
46. E. Grädel. On transitive closure logic. In *Proceedings of 5th Workshop on Computer Science Logic CSL 91*, Lecture Notes in Computer Science, No. 626, pages 149–163, Springer, 1991.
47. E. Grädel. Capturing complexity classes by fragments of second-order logic. *Theoretical Computer Science*, 101:35–57, 1992.
48. E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140:26–81, 1998.
49. E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability, *17th ACM Symposium on Principles of Database Systems PODS 98*, ACM Press, 1998.
50. E. Grädel and A. Malmström. 0-1 laws for recursive structures. *Archive of Mathematical Logic*, 38:205–215, 1999.
51. E. Grädel and G. McColm. On the power of deterministic transitive closures. *Information and Computation*, 119:129–135, 1995.
52. E. Grädel and G. McColm. Hierarchies in transitive closure logic, stratified datalog and infinitary logic. *Annals of Pure and Applied Logic*, 77:166–199, 1996.
53. E. Grädel and K. Meer. Descriptive complexity theory over the real numbers. In J. Renegar, M. Shub, and S. Smale, editors, *Mathematics of Numerical Analysis: Real Number Algorithms*, Lectures in Applied Mathematics No. 32, pages 381–403. AMS, 1996.
54. E. Grädel and M. Otto. Inductive definability with counting on finite structures. *Computer Science Logic, 6th Workshop, CSL '92, Selected Papers*, Lecture Notes in Computer Science, No. 702, pages 231–247, Springer, 1993.
55. E. Grädel and M. Otto. On logics with two variables. *Theoretical Computer Science*, 224:73–113, 1999.
56. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games.* Lecture Notes in Computer Science No. 2500. Springer, 2002.
57. R. Greenlaw, J. Hoover, and W. Ruzzo. *Limits to Parallel Computation. P-Completeness Theory.* Oxford University Press, 1995.
58. M. Grohe. Fixed-point logics on planar graphs. In *Proc. 13th IEEE Symp. on Logic in Computer Science*, pages 6–15, 1998.
59. M. Grohe. Isomorphism testing for embeddable graphs through definability. In *Proc. 32nd ACM Symp. on Theory of Computing*, pages 63–72, 2000.
60. M. Grohe and J. Mariño. Definability and descriptive complexity on databases of bounded tree-width. In *Proceedings of ICDT 99*, Lecture Notes in Computer Science, No. 1540, pages 70–82, Springer, 1999.
61. Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Current Trends in Theoretical Computer Science*, pages 1–57. Computer Science Press, 1988.

62. Y. Gurevich and L. Harrington. Trees, automata and games. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC '82*, pages 60–65, 1982.
63. Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.
64. D. Harel. Towards a theory of recursive structures. In *Proceedings of 23rd International Symposium on Mathematical Foundations of Computer Science MFCS 98*, Lecture Notes in Computer Science, No. 1450, pages 36–53. Springer, 1998.
65. T. Hirst and D. Harel. More about recursive structures: Descriptive complexity and zero–one laws. In *Proc. 11th IEEE Symp. on Logic in Computer Science*, pages 334–348, 1996.
66. W. Hodges. *Model Theory.* Cambridge University Press, 1993.
67. N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.
68. A. Itai and J. Makowsky. Unification as a complexity measure for logic programming. *Journal of Logic Programming*, 4:105–117, 1987.
69. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *Proceedings of 7th International Conference on Concurrency Theory CONCUR '96*, Lecture Notes in Computer Science, No. 1119, pages 263–277. Springer, 1996.
70. N. Jones and W. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3:105–117, 1977.
71. N. Jones and A. Selman. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic*, 39:139–150, 1974.
72. M. Jurdziński. Deciding the winner in parity games is in UP $\cap$ Co-UP. *Information Processing Letters*, 68:119–124, 1998.
73. M. Jurdziński. Small progress measures for solving parity games. In *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, Lecture Notes in Computer Science, No. 1770, pages 290–301. Springer, 2000.
74. P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and Systems Sciences*, 51:26–52, 1995.
75. B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC '94: Selected Papers from the International Workshop on Logical and Computational Complexity*, Lecture Notes in Computer Science, No. 960, pages 367–392. Springer, 1995.
76. B. Khoussainov, S. Rubin, and F. Stephan. On automatic partial orders. *Proceedings of 18th Annual IEEE Symposium on Logic in Computer Science, LICS 03*, pages 168–177, 2003.
77. B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: Richness and limitations. *Proceedings of 19th Annual IEEE Symposium on Logic in Computer Science, LICS 04*, pages 44–53, 2004.
78. B. Khoussainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In *Proceedings of STACS 04*, pages 440–451, 2004.
79. P. Kolaitis. The expressive power of stratified logic programs. *Information and Computation*, 90:50–66, 1991.
80. S. Kreutzer. Expressive equivalence of least and inflationary fixed point logic. In *Proceedings of 17th IEEE Symp. on Logic in Computer Science LICS02*, pages 403–410, 2002.

81. G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer, 2000.
82. D. Martin. Borel determinacy. *Annals of Mathematics*, 102:336–371, 1975.
83. Y. Matijasevich. *Hilbert's Tenth Problem.* MIT Press, 1993.
84. K. Meer. Query languages for real number databases based on descriptive complexity over *R*. In *Proc. 24th International Symposium on Mathematical Foundations of Computer Science MFCS 99*, Lecture Notes in Computer Science Nr. 1672, pages 12–22. Springer, 1999.
85. Y. Moschovakis. *Elementary Induction on Abstract Structures.* North-Holland, 1974.
86. A. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdansk, 1991.
87. D. Muller and P. Schupp. Groups, the theory of ends, and context-free languages. *Journal of Computer and System Sciences*, 26:295–310, 1983.
88. D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.
89. M. Otto. *Bounded Variable Logics and Counting.* Springer, 1997.
90. M. Otto. Bisimulation-invariant Ptime and higher-dimensional mu-calculus. *Theoretical Computer Science*, 224:237–265, 1999.
91. C. Papadimitriou. A note on the expressive power of Prolog. *Bulletin of the EATCS*, 26:21–23, 1985.
92. S. Rubin. *Automatic Structures.* PhD thesis, University of Auckland, New Zealand, 2004.
93. H. Scholz. Ein ungelöstes Problem in der symbolischen Logik. *Journal of Symbolic Logic*, 17:160, 1952.
94. A. Stolboushkin. Towards recursive model theory. In J. Makowsky and E. Ravve, editors, *Logic Colloquium 95*, Lecture Notes in Logic, No. 11, pages 325–338. Springer, 1998.
95. H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, Boston, 1994.
96. W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of STACS 95*, Lecture Notes in Computer Science, No. 900, pages 1–13. Springer, 1995.
97. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages* Vol. 3, pages 389–455. Springer, 1997.
98. W. Thomas. Constructing infinite graphs with a decidable MSO-theory. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science MFCS 03*, Lecture Notes in Computer Science, No. 2747, pages 113-124, Springer, 2003.
99. M. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on the Theory of Computing*, pages 137–146, 1982.
100. M. Vardi. On the complexity of bounded-variable queries. In *Proc. 14th ACM Symp. on Principles of Database Systems*, pages 266–267, 1995.
101. I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275:311–346, 2001.
102. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.