Pure and Applied Fixed-Point Logics

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von
Diplom-Informatiker

Stephan Kreutzer

aus Aachen, Deutschland

Berichter: Universitätsprofessor Dr. Erich Grädel
Universitätsprofessor Dr. Wolfgang Thomas

Tag der mündlichen Prüfung: 17.12.2002

Abstract

Fixed-point logics are logics with an explicit operator for forming fixed points of definable mappings. They are particularly well suited for modelling recursion in logical languages and consequently they have found applications in various areas of theoretical computer science such as database theory, finite model theory, and computer-aided verification.

The topic of this thesis is the study of fixed-point logics with respect to their expressive power. Of particular interest are logics based on inflationary fixed points and their comparison to least fixed-point logics.

The first part focuses on fixed-point extensions of first-order logic. In the main result we show that *inflationary* and *least fixed-point logic* – the extensions of first-order logic by least and inflationary fixed points – have the same expressive power on all structures, i.e. LFP = IFP.

In the second part of this thesis, we study fixed-point extensions of modal logic. Such logics are widely used in the field of computer-aided verification. Again, the least fixed-point extension of modal logic, the *modal μ-calculus*, is of particular interest and is among the best studied logics in this area. The main contribution of the second part is the introduction and study of the corresponding inflationary fixed-point logic. Contrary to the case of first-order logic mentioned above, where least and inflationary fixed points lead to equivalent logics, it is shown that in the context of modal logic, inflationary fixed points are far more expressive than least fixed points. On the other hand, they are algorithmically far more complex.

Besides the two main results, we study a variety of different fixed-point logics and develop methods to compare their expressive power.

Finally, in the third part, we study fixed-point logics as query languages for constraint databases. It is shown that already relatively simple logics such as the transitive closure logic lead to undecidable query languages on constraint databases. Therefore we consider suitable restrictions of fixed-point logics to obtain tractable query languages, i.e. languages with polynomial time evaluation.

A detailed overview of the results presented in this thesis can be found in the second part of the introduction.

Zusammenfassung

Die vorliegende Dissertation beschäftigt sich mit der Untersuchung von Fixpunktlogiken hinsichtlich ihrer Ausdrucksstärke. Der Schwerpunkt liegt dabei auf inflationären Fixpunktlogiken und ihrer Abgrenzung von Logiken, die auf kleinsten Fixpunkten basieren. Im ersten Teil der Arbeit werden dazu die seit langem bekannten Fixpunkterweiterungen der Prädikatenlogik untersucht. Das Hauptergebnis ist der Beweis, daß die Logiken LFP und

IFP, also die Erweiterung der Prädikatenlogik um *kleinste* und *inflationäre Fixpunkte*, die gleiche Ausdrucksstärke haben. Es gilt also LFP = IFP.

Im zweiten Teil der Arbeit stehen dann Fixpunkterweiterungen der Modallogik im Vordergrund, wie sie intensiv im Bereich der automatischen Verifikation studiert werden. Während der *modale μ-Kalkül* (L_μ), die Erweiterung der Modallogik um kleinste Fixpunkte, schon seit Anfang der 80er Jahre eingehend untersucht wird, wird hier zum ersten Mal die entsprechende inflationäre Logik, der *modale Iterationskalkül* (MIC), betrachtet. Es zeigt sich, daß, im Gegensatz zum Fall der Prädikatenlogik, inflationäre Fixpunkte im modallogischen Kontext eine sehr viel größere Ausdrucksstärke bieten als kleinste. MIC ist also sehr viel ausdrucksstärker als L_μ, allerdings im Hinblick auf algorithmische Probleme auch erheblich komplexer.

Neben diesen beiden Hauptergebnissen werden in den ersten beiden Teilen der Arbeit noch weitere Arten von Fixpunktlogiken studiert und Methoden zum Vergleich ihrer Ausdrucksstärke entwickelt.

Im dritten und letzten Teil der Dissertation stehen sogenannte *constraint Datenbanken* im Zentrum der Betrachtungen. Hierbei handelt es sich um ein relativ neues Datenbankmodell, das sich besonders zur Speicherung geometrischer Daten eignet. Ähnlich wie bei relationalen Datenbanken können auch hier Fixpunktlogiken als Grundlage von Abfragesprachen dienen. In Teil III wird gezeigt, daß in diesem Bereich allerdings schon relativ einfache Fixpunktlogiken, wie die transitive Hüllenlogik, unentscheidbare Sprachen liefern. Anhand zweier auf kleinsten Fixpunkten basierenden Logiken wird jedoch demonstriert, daß durch geeignete Definition der Logiken auch im constraint Datenbankbereich algorithmisch handhabbare Abfragesprachen mit Hilfe von Fixpunktlogiken definiert werden können.

Eine ausführlichere Darstellung der in dieser Dissertation präsentierten Ergebnisse findet sich im zweiten Teil der Einleitung.

Acknowledgements

I have many people to thank and acknowledge. First of all, I want to thank my advisor, Erich Grädel, for all his support and encouragement during the last four years. I am grateful to Anuj Dawar for the very enjoyable collaboration on several parts of this thesis and in particular for his patience during the discussions about the equivalence of least and inflationary fixed-point logic. Many thanks also to my colleagues, in particular Achim Blumensath and Dietmar Berwanger. Often enough they had the misfortune of being in their office – Achim mostly before anyone else arrived and Dietmar primarily after everybody else left – when I needed someone to bother with questions. Further, I want to thank the database group in Limburg, in particular Jan Van den Bussche and Floris Geerts for the pleasant discussions we had on constraint databases. Very special thanks go to Jan Van den Bussche for attracting my attention to the expressive power of stratified fixed-point logic and transitive-closure logic on constraint databases. Unknowingly, his questions raised at an AFM-seminar in Aachen gave me the impulse to study fixed-point logics on infinite structures and in this sense made the results reported in the first part of this thesis possible. Finally, I want to thank all the other people who contributed to this thesis in some way, in particular Colin Hirsch and David Richerby for proofreading parts of the manuscript.

Contents

Chapter 1

Introduction

Formal logics have played a crucial role in the development of theoretical computer science. A feature that is pervasive to many diverse areas such as database theory, computer-aided verification, or computational and descriptive complexity theory are definitions by *recursion* or *iteration.*

Formalising recursive definitions in a logical language usually involves some kind of fixed-point construction. This can be incorporated into the logic in various ways. In second-order logic, recursion is modelled by quantifying over the individual stages of the iteration process, whereas in infinitary logics, the same is simulated by infinitary disjunctions defining arbitrary recursion depths. Another way of modelling recursive definitions is to incorporate an explicit operator for forming fixed points. Logics following this approach are called *fixed-point logics.* In the various areas of computer science where fixed-point logics have been deployed, a huge variety of such logics has evolved. Regardless of how great the differences are elsewhere, the fixed-point part of most logics is formed according to the same common principle.

Consider a first-order formula $\varphi(R, \overline{x})$ with a free second-order variable R of arity k, and k free first-order variables $\overline{x}$. On any structure $\mathfrak{A}$, such a formula induces an operator F_φ taking a set $P \subseteq A^k$ to the set $\{\overline{a} : (\mathfrak{A}, P) \models \varphi[\overline{a}]\}$. Recursive definitions are now modelled by considering the various kinds of fixed points such an operator may possess. Among these, *least fixed points* play a fundamental role.

Least fixed points are usually incorporated into a logic as follows. If φ is positive in R, the operator F_φ is monotone – meaning that $X \subseteq Y$ implies $F_\varphi(X) \subseteq F_\varphi(Y)$. Monotone operators always have a least fixed point $\mathbf{lfp}(F_\varphi) := \bigcap\{X : F_\varphi(X) = X\}$ and therefore, on any structure $\mathfrak{A}$, a first-order formula $\varphi(R, \overline{x})$ positive in R naturally induces a set $\mathbf{lfp}(F_\varphi)$. This forms the basis of *least fixed-point logic* (LFP), an extension of first-order logic (FO) equipped with an explicit construct $[\mathbf{lfp}_{R,\overline{x}}\, \varphi(R, \overline{x})]$, for φ positive in R, defining the least fixed point of φ.

Least fixed-point definitions are numerous in mathematics. Typical examples include the definition of regular expressions over an alphabet, formed by inductively closing the set of letters under concatenation, union, and the star operation, but also the inductive definition of primitive recursive functions, the syntax of first-order logic, or the definition of sub-groups generated by a set of elements.

A different type of fixed points can be obtained by an explicit induction process. Here, the formula $\varphi(R, \overline{x})$ is used to build up the following sequence $(R^\alpha)_{\alpha \in \mathrm{Ord}}$ of sets, indexed by ordinals α.

$$\begin{array}{rcll} R^0 & := & \varnothing & \\ R^{\alpha+1} & := & R^\alpha \cup \{\overline{a} : (\mathfrak{A}, R^\alpha) \models \varphi(\overline{a})\} & \\ R^\lambda & := & \bigcup_{\xi<\lambda} R^\xi & \text{for limit ordinals } \lambda. \end{array}$$

As this sequence is increasing, it leads to a fixed point $R^\infty := R^\alpha$ for the least ordinal α such that $R^\alpha = R^{\alpha+1}$. R^∞ is called the *inflationary fixed point* of φ and is used to form the *inflationary fixed-point logic* (IFP) as the extension of FO by an operator $[\mathbf{ifp}_{R,\overline{x}}\, \varphi(R, \overline{x})](\overline{x})$ defining the inflationary fixed point of φ. This fixed point of a formula exists independently of whether φ is positive in R. However, due to a theorem by Knaster and Tarski, if φ is positive in R, the inflationary and the least fixed point coincide. Thus, LFP $\subseteq$ IFP.

Following work in recursion theory on inductive definitions in arithmetic, the first systematic study of such definitions on abstract structures occurred in the 1970's. Then, research was focused on least and inflationary fixed-point inductions – called monotone and non-monotone inductions – on first-order formulae. In particular, no explicit construct to form fixed points was considered and therefore fixed points could not be negated. Nevertheless, many fundamental methods in the theory of fixed-point logics date back to the investigations done at that time. See [Mos74a] and [Acz77] for surveys on the results and methods established by then.

In the 1980's, fixed-point logics in the modern form arose independently in various areas of computer science, e.g. in finite model theory with the introduction of logics like least and inflationary fixed-point logic (see [EF99]), in database theory with query languages such as Datalog and its various extensions (see [AHV95]), or in computer-aided verification with specification languages like LTL and CTL or the modal μ-calculus (see [CGP99] and [AN01]). The main evolution over the cases studied in the 1970's was the introduction of explicit fixed-point constructions. For instance, least fixed-point logic (LFP) has an operator $[\mathbf{lfp}_{R,\overline{x}}\, \varphi](\overline{x})$ to form the least fixed point of φ, where φ is a formula in LFP positive in R. In particular, φ can again contain fixed-point operators.

Due to the differences in scope and requirements relevant to the various areas, a huge variety of fixed-point logics has evolved. As there are different types of fixed-point operators – they can be monotone, inflationary, or even more general, they can be deterministic or not – and a variety of different logics to which they can be attached to – such as first-order, modal, or Horn-clause logic – there are many ways in which fixed-point logics can be tailored to a particular area of application. We proceed with a brief introduction to the three areas of theoretical computer science relevant to this thesis.

Finite Model Theory

When studying the model theory of finite structures, it soon becomes clear that first-order logic loses the features like compactness and the Löwenheim-Skolem property that constitute its distinguished role in infinite model theory. For one, every finite model can be axiomatised up to isomorphism by a single sentence of FO. On the other hand, many interesting *classes* of finite structures cannot be defined in first-order logic. In this sense, FO is too strong and too weak at the same time. As a consequence, in finite model theory logics like second-order logic or finite variable fragments of infinitary logic have played a prominent role. Another type of logics important to the development of finite model theory are logics incorporating some kind of fixed-point construct, primarily due to their relevance in descriptive complexity theory. See [EF99], [Grä03], and [Imm98] for a detailed introduction to finite model theory and descriptive complexity.

The subject of descriptive complexity theory is to classify computational problems in terms of the descriptive resources required for their specification. Although the equivalence between monadic second-order logic (MSO) and tree or word automata established by Büchi, Thatcher, Wright, and Doner [Büc60, TW68, Don70] can be seen as first results in this direction, the work that initiated this line of research is Fagin's theorem from 1974.

Theorem. [Fag74] A class of finite structures is definable in existential second-order logic (Σ^1_1) if, and only if, it is decidable in NP.

In this sense, Σ^1_1 provides a logical characterisation of the complexity class NP. Such results are usually referred to as *capturing results,* i.e. Σ^1_1 *captures* NP. Results of this kind are interesting because they allow the combination of the methods used in each of the individual areas and to gain a deeper inside into the structure of problems definable in the logic and the complexity class.

Similar to Fagin's theorem, capturing results on ordered structures have been established for all major complexity classes using variants of fixed-point logics. In particular, Immerman [Imm86] and Vardi [Var82] showed that, on finite ordered structures, *least fixed-point logic* (LFP) captures polynomial time computations, in the sense that a class of finite ordered structures is decidable in polynomial time if, and only if, it is definable in least fixed-point logic. Other complexity classes such as polynomial or logarithmic space can also be characterised in this way, using different fixed-point constructs. For instance, PSPACE can be captured by *partial fixed-point logic* (PFP) – the extension of first-order logic by partial fixed points – (see [AV89]) and NLOGSPACE by *transitive-closure logic* (TC) – the extension of FO by an operator that forms the transitive closure of a definable graph (see [Imm88, Imm87a]). These results raised the hope that a separation of complexity classes could be achieved using methods from finite model theory. Many of the so called capturing results rely on the availability of an ordering on the structures. In particular, it is easily seen that on classes of unordered structures, least fixed-point logic falls short of expressing all polynomial time computable properties. It is therefore remarkable that a separation of PSPACE and PTIME would also follow from a separation of LFP and PFP on classes of arbitrary finite structures. This was shown by Abiteboul and Vianu in [AV91]. See also [Daw93, DLW95].

Theorem. [AV91] On arbitrary classes of finite structures, PFP equals LFP if, and only if, PSPACE equals PTIME.

Considerable efforts have been spent on exploring the properties of the various fixed-point logics with the ultimate goal of providing a separation of the corresponding complexity classes. On arbitrary classes of finite structures, separations of some of the fixed-point logics have been achieved. In particular, it has been shown by Immerman [Imm81] that transitive-closure logic is strictly contained in a logic called *stratified fixed-point logic* (SFP) or, equivalently, stratified DATALOG. A simpler proof of this fact, that makes use of the game trees considered by Dahlhaus [Dah87], can be found in [Gro97]. Game trees were also used by Kolaitis [Kol91] to show that stratified fixed-point logic is strictly contained in least fixed-point logic on the class of finite structures. He also showed that the strict inclusion extends to infinite structures. However, the methods used rely on classes of unordered structures and the proofs do not extend to classes of ordered finite structures.

As noted above, least and inflationary fixed-point inductions have been studied since the 1970's, and already then the question of whether inflationary fixed-point inductions are more expressive than least fixed-point inductions was raised. Partial results in the direction of establishing the equivalence between inflationary and least fixed-point inductions on classes of structures equipped with a coding function were achieved in [HK75a, HK75b, Mos74b, HM74]. In 1986, Gurevich and Shelah [GS86] proved that least and inflationary fixed-point logic are equivalent on classes of finite structures. However, the question of whether in general LFP equals IFP was still open. For partial fixed-point logic, all that is known is that it contains IFP and, as PSPACE strictly contains NLOGSPACE, TC is strictly contained in PFP too.

So far we presented results showing that complexity classes like PTIME and Pspace can be captured by fixed-point logics. However, for the non-deterministic complexity classes between deterministic polynomial time and space, no such characterisation by fixed-point logics had been given. This gap was bridged by Abiteboul, Vianu, and Vardi [AVV97] with the introduction of the non-deterministic fixed-point logic NFP. It was shown that on ordered structures, NFP captures NP. They also introduced a logic based on alternating fixed points and proved that this logic captures alternating polynomial time and thus Pspace on finite ordered structures. For these logics, results similar to the aforementioned Abiteboul-Vianu theorem have been established. In particular, a separation of any of them from IFP or PFP on arbitrary classes of finite structures would show the corresponding complexity classes apart.

In both logics, the non-determinism is on the formulae which are used to define the induction stages and not on the choice of tuples included into the fixed-point relation. Logics following the second approach, i.e. where the choice is made on the tuples that are to be included into the fixed point, were considered by Arvind and Biswas [AB87], and Gire and Hoang [GH98]. See also [DR03] for a detailed study of these logics. The logic C-IFP introduced by Gire and Hoang [GH98] was designed towards capturing polynomial time computations. The idea is to restrict the choice of elements so that the logic's data complexity is in Ptime. However, in the general form considered in [DR03], it still captures NP.

A different line along which the expressive power of the various fixed-point logics can be explored is to study their bounded arity fragments. The bounded arity fragments of a fixed-point logic $\mathcal{L}$ are obtained by allowing only fixed-point variables of arity at most n, for some $n < \omega$. The hierarchy naturally obtained in this way within $\mathcal{L}$ is called the *arity hierarchy* of $\mathcal{L}$. It has been shown by Grohe [Gro96, Gro94] that on arbitrary finite structures, the arity hierarchies of all fixed-point logics mentioned so far, except for the non-deterministic logics capturing NP, are strict. In fact, he showed that there are properties of finite structures definable in the arity k fragment of transitive-closure logic that are not definable in the arity $(k-1)$ fragment of partial fixed-point logic.

On the other hand, it was shown by Imhof [Imh96b, Imh96a] that on ordered structures, all of transitive-closure logic (TC) is contained in the monadic fragment of PFP, i.e. the fragment where all fixed-point variables are of arity one. Further, he proved that, again on ordered structures, *deterministic* transitive-closure logic (DTC) is contained in the monadic fragment of IFP. Finally, Grohe showed that DTC is contained in the ternary fragment of LFP. It follows, that establishing the strictness of the arity hierarchy for LFP or IFP would yield a separation of Logspace and Ptime. Conversely, showing its collapse would separate Ptime from Pspace, as the arity hierarchy for PFP is strict, even on classes of ordered structures (see, again, [Imh96a]).

Computer-Aided Verification

The main issue of computer-aided verification is to provide formal methods for the validation of dynamic processes as those occurring e.g. in hard- or software systems. To this end, the processes are modelled as transition systems, which are directed, node and edge labelled graphs. Each individual node in the system represents a particular state of the process, and an edge between two nodes represents the transition between the two states of the process. Logics, now, play the role of specification languages that are used to formalise properties of the processes to be validated. There are various ways in which modelling a process by a transition system can be done. As a result, there usually is more than one transition system describing the behaviour of a single hard- or software system. Obviously, specifications should not distinguish between different models of the same process. This is formalised by the notion of bisimulation (see Definition 9.4) which captures a behavioural equivalence between processes. As already first-order logic is undecidable and has PSPACE-complete model checking, fixed-point logics based on full FO are of limited use for verification purposes. Instead, logics based on modal logic are used to define specification languages.

The most prominent fixed-point language studied in the area of verification is the *modal μ-calculus* (L_μ). Introduced by Kozen [Koz83] in 1983, L_μ extends basic modal logic by least fixed-point operators in the same way as LFP extends FO. In particular, if $\varphi(X)$ is a formula in L_μ, positive in X, then $\mu X.\varphi$ is also a formula in L_μ which defines the least fixed point of the operator naturally induced by φ on a transition system. The importance of the μ-calculus is based on its good balance between expressive power and complexity. In particular, it enjoys a number of properties, e.g. decidability, crucial to verification purposes. See [BS01] for an introductory paper and [AN01] for a detailed study of the μ-calculus.

It was shown by Kozen and Parikh [KP84] that the μ-calculus is decidable and later, in 1988, Kozen proved the finite model property [Koz88]. However, the complexity of the decision procedure was still non-elementary. A first elementary decision procedure for L_μ was given by Streett and Emerson in [SE89]. Later the problem was shown to be EXPTIME-complete by Emerson and Jutla [EJ88].

Further, it has been shown that model checking for L_μ is in NP$\cap$co-NP, even in UP$\cap$co-UP [Jur98]. However, the only lower bound that was proven is PTIME-hardness and it has been conjectured that the model checking problem is in PTIME. The (deterministic) algorithms available so far are all exponential in the number of alternations between least fixed-point operators and negation symbols occurring in the formula. Thus, showing L_μ model checking to be in PTIME could trivially be achieved by showing that alternations in the formulae can be avoided. However, this hope is dashed by results of Bradfield who proved that the alternation hierarchy for L_μ is strict. See [Bra98a, Bra98b] for details.

Other results link modal logics like ML and L_μ to first-order based logics. Van Benthem proved in his thesis [vB76] that the properties of transitions systems definable in ML are precisely the properties which are invariant under bisimulations and definable in FO. Later, Rosen [Ros97] established the finite model theory variant of the result. In this line of research, Janin and Walukiewicz showed a close correspondence between the modal μ-calculus and monadic second-order logic (MSO). They proved that the bisimulation invariant properties of transition systems definable in MSO are precisely the properties definable in L_μ. Whether this remains true in the restriction to finite structures is still open.

Constraint Databases

The relational database model introduced by Codd [Cod70] has become the standard model for current database systems. However, in recent years new application areas have passed beyond the capabilities of this model. Applications in data integration or the management of data on the web with the need to integrate huge amounts of data from different sources that may not obey to strict structural rules have led to the development of models for semi-structured data. Among these, the XML database model is the most prominent.

The growing interest in spatial databases and geographical information systems with their need to store geometrical figures, naturally viewed as infinite sets of points, has led to the introduction of new database models capable of meeting the requirements posed by such applications. One such model is the constraint database model introduced by Kanellakis, Kuper, and Revesz [KKR95, KKR90].

Constraint databases are relational extensions of arbitrary, potentially infinite structures $\mathfrak{A}$, called *context structure,* such that all database relations are definable by quantifier-free first-order formulae over $\mathfrak{A}$. The formulae defining the relations are called finite representations. The actual content of a constraint database relation is manipulated by manipulating the formula representing it. For instance, spatial data may be processed in a constraint database by choosing the ordered field of reals $\mathfrak{A} := (\mathbb{R}, <, +, \cdot)$ as context structure. The relations that are finitely representable over this structure are precisely the semi-algebraic sets.

Since their introduction, a huge body of results and methods exploring the power and limitations of constraint databases and query languages has been established. Considerable efforts have been spent on proving collapse results for first-order queries on constraint databases over so-called o-minimal context structures, e.g. the ordered field of reals. Different kinds of collapse results have been investigated. As constraint databases are essentially infinite structures, the quantifiers present in first-order queries naturally range over the infinite universe. In various cases, however, the expressive power of first-order logic does not decrease if quantification is restricted to the active domain – the set of all elements contained in some database relation – even if this domain should be finite. In such cases, we speak of a *natural-active collapse.* Another type of collapse results deals with queries invariant under certain transformations of the database, which is usually referred to as *genericity.* See [PVV94] and [KV00] for more details on generic queries.

A typical example of a generic collapse result is that over certain ordered context structures, every first-order query that is invariant under order preserving permutations, is equivalent to a query which only uses the database relations and the order symbol, i.e. makes no use of other built-in predicates like addition. This is called the *order-generic collapse.* For more details on collapse results see [BST99, BST97, BST96, BDLW96, BDLW98] and [BL00] and references therein. See also [Sch01, LS01], and in particular [Sch02], which, among many other things, gives a survey of collapse results on constraint databases.

When applied to spatial databases, one usually considers either constraint databases storing semi-algebraic sets, called *polynomial constraint databases*, or semi-linear sets, known as *linear constraint databases.* Polynomial constraint databases allow the storage of spatial information in a natural way.

An important application for such databases are geographical information systems (GIS) which are used to store geographic information, i.e. information about objects on or below the earth's surface. Typically, such a system contains *spatial information*, i.e. relations over the reals, in form of a map with additional non-spatial or *thematic information* about the objects on the map. The thematic information often comes from a finite or at least countable domain, like the set of words over an alphabet. For instance, local administrations provide town maps which list the historical usage of the various sites in the town. Potential purchasers of properties can then use these maps to assess the probability of having the site polluted with residual waste, e.g. oil pollution caused by former industrial use.

Geographical information systems have numerous applications in geology, environmental sciences, and geography and are becoming more and more important in these areas. As the (practical) complexity of algorithms manipulating semi-algebraic sets is too high and the accuracy achieved by linear approximations to such kinds of data is usually sufficient, geographical information systems often use semi-linear sets to store spatial information.

Essentially, the evaluation of first-order queries in the polynomial model consists of quantifier-elimination in the theory of ordered real fields, for which a non-deterministic exponential-time lower bound has been proved. On the other hand, query evaluation on linear constraint databases can be done efficiently.

It is known that first-order logic lacks the power to define queries relying on recursion. A prominent example is connectivity which is not expressible in FO on almost all interesting classes of structures. This lack of expressive power shows up on polynomial as well as linear constraint databases. In finite model theory, a standard method to solve this problem is to consider transitive closure or least fixed-point logic. Although such logics have successfully been used in the context of dense-order constraint databases (see [KKR90, GS97, GK99]), they are less suitable for the linear database model, as they are neither closed nor decidable on this class of databases (see [KPSV96]). Logics extending FO by fixed-point constructs can be found in [GK97] and [GK00]. Decidability and closure of these languages was achieved by including certain stop conditions for the fixed-point induction. The language considered in [GK97] has PTIME data-complexity and provides an example for a tractable query language. To the contrary, the language considered by Geerts and Kuijpers in [GK00] provides an example for an expressively complete language (see also [Gee01]). Another expressive complete query language is introduced by Gyssens, Van den Bussche, and Van Gucht in [GdBG97]. A detailed study of linear constraint databases and queries can be found in [KLP00, Chapter 9].

Besides queries based on recursion there are other important queries that are not first-order definable. In the context of linear databases, one such query is the convex closure query. It is easily seen that any extension of first-order logic that allows the definition of convex closures for arbitrary sets results in a query language in which full multiplication becomes definable, and which is therefore not closed. Vandeurzen et al. [VGG98, Van99] defined a query language called PFOL which extends FO by a limited amount of multiplication. They proved that in this language, the convex closure query becomes definable for finite sets of points.

Vandeurzen also showed that there are PFOL-queries that define an encoding and decoding of linear constraint databases by finite point sets. This result allows the transfer of methods from finite databases to linear constraint databases and has been utilised frequently in subsequent research. It will also play a significant role in our investigations in Part III.

Contributions of this Thesis

In this thesis we study fixed-point logics in three different areas of computer science. The first part is concerned with fixed-point extensions of first-order logic, with the focus primarily on infinite structures. The second part deals with fixed-point logics as used for verification purposes, i.e. logics based on modal logic. Finally, in Part III we consider query languages for constraint databases based on fixed-point constructs.

Part I: Fixed-Point Extensions of First-Order Logic

In the first part we consider fixed-point extensions of first-order logic. As mentioned above, fixed-point logics have played a crucial role in the development of finite model theory and even more so in descriptive complexity where they are used to provide logical characterisations of most major complexity classes. In Chapter 6, we follow this line of research by introducing the *choice fixed-point logic* (CFP) that allows the non-deterministic choice of tuples with formulae of the form $[\mathbf{cfp}^{\overline{c}}_{R,\overline{x}}\,\varphi(R,\overline{x},\overline{c})]$. The semantics is similar to the standard semantics for IFP, with the difference that at each stage a tuple of elements is chosen non-deterministically for the variables $\overline{c}$. The fixed point defined by such a formula consists of the union of all fixed points reached by some sequence of choices made at the individual stages. It is easily seen that the positive fragment of this logic, i.e. the fragment where no fixed-point operator occurs in the scope of a negation symbol, captures NP on arbitrary classes of finite structures and is therefore equivalent to the other non-deterministic fixed-point logics informally discussed above. Consequently, the full logic captures the polynomial-time hierarchy.

As noted above, Imhof [Imh96a] proved that on ordered structures deterministic transitive closure logic (DTC) is contained in the monadic fragment of IFP. Similarly, we will show containment of (non-deterministic) transitive-closure logic (TC) in the monadic fragment of CFP. On unordered structures, one additional binary fixed-point operator is needed. We can even strengthen the result by showing that the translation of TC into the monadic fragment of CFP can be done so that the resulting formulae only need two universal and no existential quantifiers.

In the preceding paragraphs, we discussed a variety of fixed-point logics, e.g. logics based on least, inflationary, and partial fixed points. All these logics can be studied on finite as well as infinite structures – with the exception of partial fixed points. The reason why partial fixed-point logic does not extend to infinite structure is that the common semantics given to this logic has no rule to define limit stages of the fixed-point induction. As the sequence of stages induced by a partial fixed point is not necessarily increasing, there is no straightforward way to define such a rule. A second limitation of the common semantics is that it does not generalise to extensions of other logics than FO, e.g. to modal logic. We give several examples for this in Chapter 7 and 12.

In Chapter 7, an alternative semantics for partial fixed-point inductions is defined that overcomes the first, and arguably also the second limitation. By giving an explicit rule to define limit stages, we guarantee that PFP with this semantics is well-defined on infinite structures. We show that on finite structures, partial fixed-point logic with the new and the old semantics are equivalent. Further, it is shown that PFP with the new semantics contains IFP via the natural translation of formulae. Even more so, we will be able to demonstrate a property on infinite structures that is definable in PFP but not in IFP. Thus, IFP is strictly contained in PFP.

To generalise the partial fixed-point semantics so that it can also be affixed to other logics than FO, we give a new interpretation to what the result of a sequence of stages defined by a partial fixed-point induction shall be. We demonstrate its usefulness in Chapter 7, and study to some depth the partial fixed-point extension of modal logic in Chapter 12.

The main contribution of this thesis can be found in Chapter 8 where we address the problem of whether least and inflationary fixed-point logic are equivalent. As explained above, the question was first raised in the 1970's in the context of monotone and non-monotone inductions over first-order formulae, and has been open since then. A positive solution of this question in the restriction to finite structures has been given by Gurevich and Shelah in [GS86]. In Chapter 8 we give a positive answer to the general problem by showing that for every IFP-formula there is a formula in LFP such that both are equivalent on all structures.

Part II: Fixed-Point Extensions of Modal Logic

In Part II, we consider fixed-point extensions of modal logic. As mentioned above, such logics are widely used in the area of verification. The modal μ-calculus, the extension of modal logic by least fixed-points, is a very well-studied logic in this field. However, so far no fixed-point constructs more powerful than least fixed points have been considered in the context of modal logics. Therefore, we start our investigations in this area by considering the straightforward extension of modal logic by inflationary fixed points. We call this logic the *modal iteration calculus* (MIC). Contrary to the results established in Chapter 8, where it is proved that least and inflationary fixed points in the context of first-order logic are equivalent, we will see that in the context of modal logic, inflationary fixed points are far more expressive and complex and various properties of L_μ fail for MIC. In particular, we show that MIC has infinity axioms, i.e. the finite model property fails. Further, we show that the satisfiability problem for MIC is undecidable, in fact not even in the arithmetical hierarchy, whereas it is EXPTIME-complete for L_μ. Regarding the model checking complexity, we show that it is PSPACE-complete for MIC, whereas it is in UP$\cap$CO-UP for L_μ, and potentially even in PTIME.

We also investigate the type of languages definable in MIC and show that there are languages that are not context-free but MIC-definable. As every language definable in L_μ is regular, this gives another separation of the two logics. Finally, for inflationary fixed points simultaneous inductions are more powerful than inductions over a single formula. This also differs from the μ-calculus, where simultaneous fixed points can be eliminated in favour of nested inductions. Clearly, these results disqualify MIC as a logic for verification purposes.

In Chapter 12 we study partial fixed-point extensions of modal logic. We first consider the common partial fixed-point semantics and demonstrate its failure in this context. We continue by studying the alternative semantics defined in Chapter 7. The resulting logic is called the *modal partial iteration calculus* (MPC). We show that the unary trace equivalence problem, which is known to be CO-NP-complete, is definable in MPC. As this problem is shown not to be definable in MIC (see Section 14.4), it follows that MPC is a strict extension of the modal iteration calculus.

So far we have considered a variety of modal fixed-point logics. There are various techniques that may be used to compare their expressive power. Showing that one logic is at least as expressive as another can be done by giving an explicit translation of formulae of the first to formulae of the second. Establishing separations between logics is often more involved. Usually, this requires identifying a property expressible in one logic, and showing that it is not expressible in the other. Many specialised techniques have been developed for such proofs of inexpressibility. One may consider diagonalisation arguments similar to the one we use in Chapter 7 to separate PFP and IFP. Such methods have also been used by Bradfield to establish that increasing alternations of least and greatest fixed points in the μ-calculus yield greater expressive power [Bra98a]. Another important method are variants of Ehrenfeucht-Fraïssé games, for instance bisimulation games. One can also relate logics to finite automata, allowing for the use of methods such as the pumping lemma.

In Chapter 13 and 14, an alternative complexity measure for modal properties of finite structures, called *labelling index*, is introduced. We will use this measure to analyse the expressive power of various fixed-point logics, in particular the modal logics introduced in Part II.

The notion of labelling index generalises the concept of the automaticity of languages, introduced by Shallit and Breitbart in [SB96]. The idea is to classify languages not in terms of the automata model or type of grammar needed to accept the whole language, but in terms of the growth rates of automata accepting the fragments of the language of words up to a fixed length.

We aim at extending the concept of automaticity from words to modal properties of arbitrary transition systems. This extension consists of two somewhat independent parts. First, we introduce a measure of size or complexity of transition systems, called their *rank*, that generalises the length of a word. We then introduce an automata-like device that we call *labelling system*. It is shown that every class of transition systems can be defined up to a fixed rank by a labelling system.

For every class $\mathcal{C}$ of transition systems, the function that takes every $n \in \omega$ to the size of the smallest labelling system, that accepts the subclass of $\mathcal{C}$ of structures whose rank is at most n, is called the *labelling index* of $\mathcal{C}$.

By deriving a number of separation results from it, we demonstrate that this is a useful measure for the complexity of classes of finite transition systems. In particular, we will obtain separation results for the modal logics introduced so far.

Further, we use the framework of labelling indices to analyse the relationship between the monadic fragments of first-order based fixed-point logics and the corresponding modal logics. As noted above, for least fixed points, Janin and Walukiewicz [JW96] were able to show that precisely the bisimulation invariant MSO-definable properties of transition systems are definable in L_μ. Further, the bisimulation invariant properties of transition systems that are definable in MSO and the monadic fragment of LFP coincide. This proves that L_μ is also the bisimulation invariant fragment of monadic LFP. As ML is the bisimulation invariant fragment of FO [vB76, Ros97], it is conceivable that this correspondence between the monadic fragments of first-order fixed-point logics and the corresponding modal logics can be extended to other types of fixed points. This hope is dashed by results we obtain in Section 13.4 where we utilise the labelling index to show that there are bisimulation invariant properties definable in monadic IFP that are not definable in MIC.

We also consider the relationship between the labelling index of a class $\mathcal{C}$ and conventional time and space based notions of its complexity. Finally, we determine the labelling index of the trace equivalence problem over specific classes of structures and derive results about its expressibility in modal logics.

As it turns out, the framework of labelling systems as described above is smoother when applied to classes of acyclic structures than when cycles are allowed. Therefore, we split the presentation in two parts and in Chapter 13 introduce the concept on the class of acyclic finite structures, i.e. those bisimilar to finite trees, and generalise it to classes of arbitrary structures in Chapter 14.

Part III: Fixed-Point Query Languages for Constraint Databases

In the third part of this thesis, we consider query languages for constraint databases. Among the various possible context structures, the structure $(\mathbb{R}, <, +)$ of the ordered real group is the most important. We have already noted that first-order logic lacks the power to define a variety of interesting queries. One such query is *convex closure*. A logic capable of defining convex closure cannot be closed on the class of linear constraint databases (see Theorem 16.4). *Closure* here means that the result of a query on a linear constraint database has to be semi-linear again. As closure is an important property of any constraint query language, we start our investigation with the definition of a query language that extends first-order logic by a restricted form of convex hull operator that retains the closure property (see Section 16.3.1). It turns out that the expressive power of this language coincides with the expressive power of the query language PFOL introduced by Vandeurzen et al. [Van99, VGG98]. As a result, the queries that encode and decode a semi-linear set by a finite set of points, shown to be expressible in PFOL, can also be expressed in the query language that we call *FO(conv)*.

In Chapter 17 and 18, we study query languages based on fixed-point logics. As we will see, fixed-point logics can be used for defining highly expressive but also tractable query languages. As an example of the former, we show that transitive closure logic (TC) is expressively complete on the class of linear constraint databases in the sense that every partially computable query on such databases is definable in TC. However, care has to be taken on what the partially computable query defined by a formula in TC is. As on finite databases the result of a TC-formula is always computable, the precise method used to compute it is irrelevant for the query defined by this formula. On linear constraint databases, however, it is easily seen that full arithmetic is TC-interpretable in a linear constraint database. Thus, the result of a TC-query is not necessarily computable. Therefore, the partially computable query defined by such a formula depends on the precise operational semantics, i.e. evaluation method, given to TC – with some evaluation schemata the result of a given query might be computable whereas other schemata may fail to do so. Therefore, we first define a suitable operational semantics for TC (see Chapter 17) for which we are then able to prove that precisely the partially computable queries on linear constraint databases are TC-definable.

Finally, in Chapter 18, we show that fixed-point logics can also be used to define tractable, yet expressive, query languages. We demonstrate this by defining two languages based on least fixed points. The first extends the query language *FO(conv)* defined in Section 16.3.1 by a least fixed-point operator. The fixed point induction is not defined over sets of real numbers, but over the regions in a decomposition of the input structure. For practical applications, this means that the input map is decomposed into meaningful parts, e.g. streets, houses, parks, and so on, and then the fixed point induction runs over sets of these regions. In the theoretical framework developed below, we take an arrangement of the input structure as basis for this logic. Arrangements of semi-linear sets are presented in Section 16.2.2. It is shown that the resulting query language has polynomial time data complexity and, moreover, is expressive enough to define all polynomial time computable queries. In this sense, the language, called *RegLFP(conv)*, captures PTIME on the class of linear constraint databases.

We also introduce a second query language based on a least fixed-point construct. This time, the fixed-point induction is not defined in terms of a decomposition of the input. Instead, the fixed point is defined by two formulae. One formula is used to define a finite set of points, and the other defines an ordinary least fixed-point induction, with the restriction that only tuples from the set defined by the other formula are included into the fixed point. This way, closure and decidability is retained. It is shown, that this logic also has polynomial time data complexity and captures PTIME on the class of linear constraint databases. Thus, the two logics introduced in Chapter 18 are actually equivalent.

Chapter 2

Preliminaries

In this chapter, we present the notation used in the sequel. We assume familiarity with standard notions of mathematical logic and complexity theory. An introduction to mathematical logic can be found in [EFT94]. Our notation follows [Hod97], which, besides being a text book on model theory, also serves as an excellent introduction to mathematical logic. For an introduction to complexity theory we refer to [Pap94]. We also assume familiarity with ordinal numbers and transfinite inductions as it can be found in any text book on set theory. See e.g. [Jec97] or [Mos94].

We use small letters $x, y, z, \dots$, possibly subscripted, to denote first-order variables and capital letters $R, Q, X, Y, Z, \dots$ for second-order variables. Tuples of variables are denoted as $\overline{x}$. We often use tuples without being specific about their length, which will then always be irrelevant or clear from the context. If A is a set, we also write $\overline{a} \in A^k$ and in this case k is assumed to be the arity of $\overline{a}$.

Ordinal numbers will always be denoted by Greeks letters $\alpha, \beta, \xi, \lambda, \dots$, where λ usually stands for a limit ordinal. We write Ord for the class of all ordinals. For finite ordinals, i.e. integers, we use small letters like n, m, i, j, c and k, where n and m usually denote variable, large integers, e.g. the size of a finite structure, i and j are indices, and k and c are fixed, constant elements, e.g. the arity of a tuple or relation.

If A is a set, then $|A|$ denotes its cardinality and $|A|^+$ the least infinite cardinal strictly greater than $|A|$. Further, for any set $B \subseteq A$, B^c denotes its complement in A.

Usually, structures are denoted by German letters $\mathfrak{A}, \mathfrak{B}, \dots$ with universes $A, B, \dots$ respectively. If φ is a formula, we write $\varphi(\overline{x})$ to express that the variables $\overline{x}$ occur among the free variables of φ. Note that φ may have additional free variable besides the variables in $\overline{x}$. For any tuple $\overline{a}$ from A we write $\mathfrak{A} \models \varphi(\overline{a})$ to express that φ is satisfied by the interpretation $(\mathfrak{A}, \sigma)$ that assigns to each variable x_i in $\overline{x} := x_1, \dots, x_k$ the element a_i from $\overline{a} := a_1, \dots, a_k$. We write $\varphi^{\mathfrak{A}}$, and sometimes also $\varphi(\mathfrak{A})$, for the set

$\{\overline{a} \in A^k : \mathfrak{A} \models \varphi(\overline{a})\}$. Similarly, if $\overline{t}$ is a tuple of terms, then $\overline{t}^{\mathfrak{A}}$ denotes the interpretation of $\overline{t}$ in $\mathfrak{A}$.

When writing formulae, we will often use an informal notation. In general, the policy has been taken to present formulae as simple and understandable as possible. Whenever a choice had to be made between syntactical correctness and readability, the latter has been chosen. However, it will always be clear that there is an equivalent formula that is syntactically correct.

We now list some notation that will frequently be used in the sequel. We often write $\forall\overline{x} < \overline{z}\,\varphi$ instead of $\forall\overline{x}(\overline{x} < \overline{z} \rightarrow \varphi)$. Similarly, if R is a second-order variable, we use expressions like $\forall\overline{x} \in R$ with the obvious semantics. We also write $R = \varnothing$ or $R = Q$, where R and Q are second-order variables, instead of $\forall\overline{x}\neg R\overline{x}$ and $\forall\overline{x}(R\overline{x} \leftrightarrow Q\overline{x})$ respectively. As a final bit of notation, if φ and $\psi(\overline{x})$ are formulae and R is a second-order variable, then $\varphi(R\overline{u}/\psi(\overline{u}))$ denotes the formula obtained from φ by replacing every occurrence of an atom $R\overline{u}$ in φ by the formula $\psi(\overline{u})$, where $\overline{u}$ is any tuple of terms and $\psi(\overline{u})$ stands for the formula ψ with the variables in $\overline{x}$ being replaced by the corresponding terms in $\overline{u}$.

At various places in this thesis we will use *word structures* $\mathfrak{W}_w$ that encode words of an alphabet as a structure.

2.1 Definition. *Let Σ be a non-empty alphabet. For every word $w \in \Sigma^*$ of length n, define the word structure*

$$\mathfrak{W}_w := (\{0, \ldots, n-1\}, <, \mathit{succ}, \mathit{min}, \mathit{max}, (P_a)_{a\in\Sigma}),$$

such that for each $a \in \Sigma$, the i-th letter w_i of w is a if, and only if, $i \in P_a^{\mathfrak{W}_w}$. Further, succ and $<$ denote the natural successor and order relation on the universe and min and max the minimal and maximal element.

Depending on the context, we sometimes consider word structures without the successor relation and the constants min and max.

Part I

Fixed-Point Extensions of First-Order Logic

Chapter 3

Least and Inflationary Fixed-Point Logic

In this chapter, we introduce two fundamental fixed-point logics – *least* and *inflationary fixed-point logic* – and present basic results about their expressive power and model theory.

3.1 Definitions by Monotone Inductions

Consider the usual definition of regular expressions over an alphabet Σ.

3.1 Definition. *Let Σ be an alphabet and $\Sigma' := \Sigma \mathbin{\dot\cup} \{), (, \cdot, ^*, +\}$. The class of regular expressions over Σ is defined as the least set* $\mathrm{Reg}_\Sigma \subseteq (\Sigma')^*$ *such that*

- $\Sigma \subseteq \mathrm{Reg}_\Sigma$ *and*
- *with any $u, v \in \mathrm{Reg}_\Sigma$, Reg_Σ also contains $(u + v), (u \cdot v)$ and $(u)^*$.*

Definitions of this kind occur frequently in mathematics. Among the numerous other examples are the Fisher-Ladner closure of formulae, the transitive closure of graphs, sub-groups induced by sets of elements, the definition of first-order logic, and many more.

On a more abstract level, one can read the definition above as defining Reg_Σ as the least set closed under the operator $F_{\mathit{Reg}} : \mathrm{Pow}((\Sigma')^*) \to \mathrm{Pow}((\Sigma')^*)$, which takes any set $X \subseteq (\Sigma')^*$ to the set containing all elements from Σ and also all $(u)^*, (u+v), (u \cdot v)$ for any pair $u, v \in X$. Similar operators can be given for many other definitions of this form.

The important property that these operators share is that for all sets X, Y such that $X \subseteq Y$ also $F(X) \subseteq F(Y)$. Operators satisfying this property are called *monotone*.

3.2 Definition. *Let A be a set and $F : \mathrm{Pow}(A) \to \mathrm{Pow}(A)$ be a function. F is called* monotone *if for all $X \subseteq Y \subseteq A$, $F(X) \subseteq F(Y)$. A* fixed point P of F *is any set $P \subseteq A$ such that $F(P) = P$. A* least fixed point of F *is a fixed point that is contained in any other fixed point of F.*

In the light of this definition, we can rephrase the example above and define the set of regular expressions as the least set *Reg* closed under the operator F_{Reg}, i.e. the least fixed point of F_{Reg}. The importance of the class of monotone operators is based on the fact that they always have a unique least fixed point. This is a special property of monotone operators as in general, functions $f : \mathrm{Pow}(A) \to \mathrm{Pow}(A)$ do not need to have any fixed points, let alone least fixed points.

However, monotone operators do have least fixed points and these can be characterised in various ways.

3.3 Theorem (Knaster and Tarski). *Let A be a set. Every monotone function $F : \mathrm{Pow}(A) \to \mathrm{Pow}(A)$ has a least and a greatest fixed point, written as* **lfp**(F) *and* **gfp**(F)*, which can be defined as*

$$\mathbf{lfp}(F) := \bigcap\{X \subseteq A : F(X) = X\} = \bigcap\{X \subseteq A : F(X) \subseteq X\},$$

and

$$\mathbf{gfp}(F) := \bigcup\{X \subseteq A : F(X) = X\} = \bigcup\{X \subseteq A : F(X) \supseteq X\}.$$

The proof of this theorem is standard and therefore omitted. A consequence of the Knaster-Tarski theorem is, that definitions such as the definition of regular expressions given above, define a unique set.

There is another common way of defining regular expressions, namely by saying that the set Reg_Σ of regular expressions over an alphabet Σ is inductively build up from the set of letters in Σ by iteratively applying the following rules:

- If $u \in Reg_\Sigma$, then add the word $(u)^*$ to Reg_Σ.
- If $u, v \in Reg_\Sigma$, then add $(u + v)$ and $(u \cdot v)$ to Reg_Σ.

In general, this kind of definition starts with the empty set and then iteratively applies an operator F until no further elements are added. The result of such a process is called the *inductive fixed point of F*.

3.4 Definition. *Let A be a set and $F : \mathrm{Pow}(A) \to \mathrm{Pow}(A)$ be a monotone operator. Consider the sequence $(X^\alpha)_{\alpha \in \mathrm{Ord}}$ of sets $X^\alpha \subseteq A$ defined as*

$$\begin{aligned} X^0 &:= \varnothing \\ X^{\alpha+1} &:= F(X^\alpha) \\ X^\lambda &:= \bigcup_{\xi<\lambda} X^\xi \qquad \textit{for limit ordinals } \lambda. \end{aligned}$$

As F is monotone, this sequence of sets is increasing, i.e. for all $\alpha \in \mathrm{Ord}$, $X^\alpha \subseteq X^{\alpha+1}$, and therefore reaches a fixed point X^∞, with $X^\infty := X^\alpha$ for the least ordinal α such that $X^\alpha = X^{\alpha+1}$. The fixed point X^∞ is called the inductive fixed point *of F.*

Often, these two kinds of definitions are used interchangeably. The following theorem due to Knaster and Tarski proves that they are equivalent indeed, i.e. any least fixed point of a monotone operator can inductively be built up by a sequence of sets as above. Again, the proof of the theorem is omitted.

3.5 Theorem (Knaster-Tarski). *Let A be a set. For every monotone operator $F : \mathrm{Pow}(A) \rightarrow \mathrm{Pow}(A)$, the least and the inductive fixed point coincide.*

Similarly, the greatest fixed point of a monotone operator can also be defined inductively by the following sequence of sets:

$$\begin{aligned} X^0 &:= A \\ X^{\alpha+1} &:= F(X^\alpha) \\ X^\lambda &:= \bigcap_{\xi<\lambda} X^\xi \qquad \text{for limit ordinals } \lambda. \end{aligned}$$

Least and greatest fixed points are dual to each other. For every operator F define the dual operator $F^d : X \longmapsto F(X^c)^c$. If F is monotone, then F^d is also monotone and we have that

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

3.2 Elementary Inductive Definitions

In this section we continue the study of definitions by monotone inductions for such operators F that are first-order definable.

Let τ be a signature and $\mathfrak{A} := (A, \tau)$ be a τ-structure. Consider a first-order formula $\varphi(R, \overline{x})$ with k free variables $\overline{x}$ and a free relation symbol R not occurring in τ. On the structure $\mathfrak{A}$, the formula φ induces an operator

$$\begin{aligned} F_\varphi : \ \mathrm{Pow}(A^k) &\longrightarrow \mathrm{Pow}(A^k) \\ R &\longmapsto \{\overline{a} : (\mathfrak{A}, R) \models \varphi[\overline{a}]\}. \end{aligned}$$

In general, an operator induced by an arbitrary formula is not necessarily monotone and it also need not have any fixed points. For instance, the formula $\varphi(R, x) := \neg\exists y\, Ry$ defines on any structure $\mathfrak{A}$ the operator F_φ taking the empty set to the universe A of $\mathfrak{A}$ and all other sets to the empty set. Thus, this operator has no fixed points.

But even in the case where fixed points exist, there need not to be a least fixed point and even if there is a least fixed point, it is not necessarily reachable by an inductive sequence as in Definition 3.4.

However, if the formula is chosen such that the operator becomes monotone, then there are always least and greatest fixed points. Unfortunately, whether a first-order formula defines a monotone operator is undecidable. One possible way to avoid this, is to restrict the class of admissible formulae even further, namely to formulae which are positive in the free second-order variable.

A simple induction on the structure of the formulae shows, that if $\varphi(R, \overline{x})$ is positive in the second-order variable R, then the operator induced by φ is monotone in R.

3.6 Definition. *Let $\mathfrak{A} := (A, \tau)$ be a structure and $\varphi(R, \overline{x})$ be a first-order formula positive in the free k-ary second-order variable $R \notin \tau$. Let $\overline{x}$ be a free k-tuple of first-order variables and let $\overline{z}$ list all free first-order variables of φ other than those in $\overline{x}$. For any interpretation $\overline{b}$ of the variables $\overline{z}$ we define by transfinite induction for every ordinal α the* stage R^α of the induction on *φ as follows.*

$$\begin{array}{rcll} R^0 & := & \varnothing & \\ R^{\alpha+1} & := & \{\overline{a} : (\mathfrak{A}, R^\alpha) \models \varphi(\overline{a})\} & \\ R^\lambda & := & \bigcup_{\xi<\lambda} R^\xi & \text{for limit ordinals } \lambda. \end{array}$$

We also write φ^α for the α-th stage of the induction on φ. The closure ordinal $\mathrm{cl}(\varphi)$ of φ on $\mathfrak{A}$ *is the least ordinal α such that $\varphi^\alpha = \varphi^{\alpha+1}$. Further, define the* closure ordinal of $\mathfrak{A}$ *as the least upper bound of all closure ordinals of positive first-order formulae.*

A set $P \subseteq A^k$ is positive elementary inductively definable in $\mathfrak{A}$*, or simply* inductive in $\mathfrak{A}$*, if there is a first-order formula $\varphi(R, \overline{x})$, positive in the free k-ary relation symbol R, and a tuple $\overline{b}$ of elements from A interpreting the free variables in φ other than $\overline{x}$, such that P is the fixed point reached by the induction on φ in $(\mathfrak{A}, \overline{b})$. The elements in $\overline{b}$ are called the* parameters *of the induction.*

Finally, P is co-inductive *if its complement in A is inductive and it is* hyperelementary *if it is both, inductive and co-inductive.*

We agree on the following notation that will be used in various places later on.

Notation. Let $(R^\alpha)_{\alpha\in\mathrm{Ord}}$ be an increasing sequence of sets. For any $\alpha \in \mathrm{Ord}$, define $R^{<\alpha} := \bigcup_{\xi<\alpha} R^\xi$. If the sequence $(R^\alpha)_{\alpha\in\mathrm{Ord}}$ is the sequence of stages induced by a formula φ on a structure $\mathfrak{A}$, we also write $\varphi^{<\alpha}$ for $R^{<\alpha}$. □

As noted in the introduction, the concept of fixed-point logics dates back to the study of inductive definitions in generalised recursion theory. The first systematic study of inductive definitions on abstract structures, that is on structures other than the arithmetic, can be found in Moschovakis' book *Elementary Induction on Abstract Structures* [Mos74a] from 1974. See also [Acz77].

We continue the example from the previous section and show that the set of regular expressions is inductive.

3.7 Example. *Let Σ be an alphabet and let Γ be the disjoint union of Σ and $\{), (, \cdot, +, ^*\}$. Consider the structure $\mathfrak{A} := (\Gamma^*, \Sigma, \cdot, c_(, c_), c_., c_+, c_*)$ over the signature $\tau_{\text{reg}} := \{\Sigma, \cdot, c_(, c_), c_., c_+, c_*\}$, where $\Sigma^{\mathfrak{A}}$ is interpreted by the set of letters in Σ, $\cdot^{\mathfrak{A}}$ is a binary function interpreted as the concatenation of words, and, for $a \in \Gamma \backslash \Sigma$, c_a is a constant interpreted by the symbol a, e.g. $c_*^{\mathfrak{A}}$ is interpreted as the symbol *. Then the set $\text{Reg}_\Sigma \subseteq \Gamma^*$ is obtained as the least fixed point of the formula $\varphi(R, x)$ defined as*

$$\begin{array}{rcl}\varphi(R,x) & := & \Sigma x \vee \exists u \exists v\, (Ru \wedge Rv \wedge (x = c_(\cdot u \cdot c_) \cdot c_* \vee \\ & & x = c_(\cdot u \cdot c_. \cdot v \cdot c_) \vee x = c_(\cdot u \cdot c_+ \cdot v \cdot c_)).\end{array}$$

Note that, independent of the cardinality of Σ, the fixed point of φ on $\mathfrak{A}$ is reached at stage ω. This is due to the fact that φ is existential and will be proved below.

We proceed by collecting some simple facts about the notions introduced so far.

3.8 Proposition. *Let $\mathfrak{A} := (A, \tau)$ be a structure.*

(i) *For every formula $\varphi(R, \overline{x})$ positive in R, $\text{cl}(\varphi) < |A|^+$.*

(ii) *Further, if φ is existential, then $\text{cl}(\varphi) \leq \omega$.*

(iii) $\text{cl}(\mathfrak{A}) \leq |A|^+$.

Proof. Parts *(i)* and *(iii)* are trivial. Towards establishing Part *(ii)*, suppose there is an existential positive formula $\varphi(R, \overline{x})$ such that $\text{cl}(\varphi) > \omega$. Without loss of generality, we assume that φ is in prenex normal form, i.e. $\varphi(\overline{x}) := \exists z_1 \ldots \exists z_n \psi(\overline{x}, \overline{z})$. As $\text{cl}(\varphi) > \omega$, there exists a tuple $\overline{b}$ in $\varphi^{\omega+1} \backslash \varphi^\omega$. Thus, $(\mathfrak{A}, R^\omega) \models \varphi(\overline{b})$ and therefore there are elements $a_1 \ldots a_n$ such that $(\mathfrak{A}, R^\omega) \models \psi(\overline{b}, a_1, \ldots, a_n)$. As $R^\omega := \bigcup_{\xi<\omega} R^\xi$, there is a stage $\alpha < \omega$ such that $a_i \in R^\alpha$ for all i. But then, $(\mathfrak{A}, R^\alpha) \models \psi(\overline{b}, a_1, \ldots, a_n)$ and therefore $\overline{b} \in R^{\alpha+1}$ contradicting the assumption that $\overline{b} \notin R^\omega$. □

We now present one of the most fundamental tools in the theory of inductive definitions: the stage comparison relations. Introduced by Moschovakis [Mos74a], they provide the central method to compare inductive definitions.

3.9 Definition (Stage Comparison Relations). *Let $\varphi(R, \overline{x})$ be a formula positive in R and let $\overline{a}$ be a tuple of elements from A. The* rank $|\overline{a}|_\varphi$ of $\overline{a}$ with respect to φ *is defined as the least ordinal α such that $\overline{a} \in \varphi^\alpha$. If there is no such ordinal, the rank of $\overline{a}$ is ∞.*

The stage comparison relations $\leq_\varphi$ and $\prec_\varphi$ of φ *are defined as*

$$\overline{x} \leq_\varphi \overline{y} \iff \overline{x}, \overline{y} \in \varphi^\infty \text{ and } |\overline{x}|_\varphi \leq |\overline{y}|_\varphi,$$

and

$$\overline{x} \prec_\varphi \overline{y} \iff \overline{x} \in \varphi^\infty \text{ and } |\overline{x}|_\varphi < |\overline{y}|_\varphi,$$

where we allow $|\overline{y}|_\varphi$ to be ∞.

The proof of the following lemma follows right from the definition.

3.10 Lemma. *For all $\overline{a}$,*

$$\begin{aligned} \overline{a} \in \varphi^\infty \quad & \text{if, and only if,} \quad \overline{a} \leq_\varphi \overline{a} \\ & \text{if, and only if,} \quad (\mathfrak{A}, \{\overline{u} : \overline{u} \prec_\varphi \overline{a}\}) \models \varphi[\overline{a}]. \end{aligned}$$

Thus, the fixed point of a formula φ can be recovered from its stage comparison relations. The following theorem, due to Moschovakis, shows that these relations are themselves inductive.

3.11 Theorem (Stage Comparison Theorem). *Let $\varphi(R, \overline{x})$ be a first-order formula positive in R. Then $\leq_\varphi$ and $\prec_\varphi$ can be obtained as the fixed point of positive inductions on first-order formulae.*

A sketch of the proof will be presented in Section 3.3, where the corresponding and more general theorem is established for least fixed-point logic.

There is a second definition of induction stages that is common in the literature. Here the stages R^α are inductively defined by the rule

$$R^\alpha := \{\overline{a} : (\mathfrak{A}, R^{<\alpha}) \models \varphi(\overline{a})\}.$$

Obviously, the fixed points of the induction sequences defined by either rule are identical and therefore one can use either to define inductive sets. Which one to choose is a matter of style and preference.

The main difference between the two definitions is that in the definition of stages as in 3.6, all elements enter the fixed point at successor stages, i.e. there are no elements whose rank is a limit ordinal. In the alternative definition, however, there might be such elements. In particular, according to this definition, Stage 0 is not just the empty set but already contains the elements that are in Stage 1 of our definition. More general, stage α in the alternative definition corresponds to stage $\alpha + 1$ according to Definition 3.6.

The alternative definition is sometimes more convenient when proving results about fixed point inductions, as the rank of the elements coincides with their height in the well-ordering defined by $\prec_\varphi$. However, we will stick to Definition 3.6, as it seems to be more common in the literature. The only exception to this is Chapter 8, where the alternative rule is applied.

3.3 Least and Monotone Fixed-Point Logic

In the previous section, we studied inductive definitions—relations defined by iteratively applying a first-order formula. In computer science, logics are often considered in the tradition of query or programming languages. Here, one is interested in nesting inductive definitions, i.e. to be able to use the fixed point built up by a formula to define another fixed point. This gives rise to the first of our inductive logics, the *monotone fixed-point logic.*

3.12 Definition (Monotone Fixed-Point Logic). Monotone fixed-point logic (MFP) *is defined as the extension of first-order logic by the following formula building rule. If $\varphi(R,\overline{x})$ is a formula with free first-order variables $\overline{x} := x_1,\dots,x_k$ and a free second-order variable R of arity k such that the corresponding operator F_φ is monotone on all structures, then*

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

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

Let $\mathfrak{A}$ be a structure providing an interpretation of the free variables of φ except for $\overline{x}$. Then for any tuple $\overline{t}$ of terms, $\mathfrak{A} \models [\mathbf{lfp}_{R,\overline{x}}\,\varphi](\overline{t})$ if, and only if, $\overline{t}^{\mathfrak{A}} \in \mathbf{lfp}(F_\varphi)$.

As explained above, the least fixed point of any monotone operator F always exists. Therefore the semantics of the monotone fixed-point logic is well defined. However, the property of a formula to define an operator monotone on all structures is undecidable and therefore the logic has an undecidable syntax.

To avoid this, one considers syntactical restrictions of MFP which guarantee monotonicity of the corresponding operators. The most important such restriction is to allow only formulae in the fixed-point rule, which are positive in the relation variable R. This leads to the definition of *least fixed-point logic.*

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

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

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

Let $\mathfrak{A}$ be a structure providing an interpretation of the free variables of φ except for $\overline{x}$. Then for any tuple $\overline{t}$ of terms, $\mathfrak{A} \models [\mathbf{lfp}_{R,\overline{x}}\, \varphi](\overline{t})$ if, and only if, $\overline{t}^{\mathfrak{A}} \in \mathbf{lfp}(F_\varphi)$.

Clearly, LFP $\subseteq$ MFP. Further, the theorem by Knaster and Tarski applies to least and monotone fixed-point logic. Thus the fixed points can also be defined by an inductive sequence of stages. Finally, the notion of closure ordinals of formulae and structures can easily be adopted to this framework too.

A fragment of LFP that is of particular interest to various areas of computer science is monadic least fixed-point logic.

3.14 Definition. Monadic least fixed-point logic (M-LFP) *is defined as the fragment of* LFP *where all fixed-point variables are monadic, i.e. of arity one.*

We proceed by presenting several examples that demonstrate the use of least fixed-point logic.

3.15 Example.
- *Let $\mathfrak{G} := (V, E)$ be a graph. The transitive closure*[1] *of E is defined by the formula $[\mathbf{lfp}_{R,x,y}\, Exy \vee \exists z(Rxz \wedge Rzy)](x, y)$. More general, if $\varphi(\overline{x}, \overline{y})$ is a formula and $\mathfrak{A}$ a structure, the transitive closure of the relation $\varphi^{\mathfrak{A}}$ defined by φ on $\mathfrak{A}$ is defined by*

$$[\mathbf{lfp}_{R,\overline{x},\overline{y}}\, \varphi(\overline{x}, \overline{y}) \vee \exists \overline{z}(R(\overline{x}, \overline{z}) \wedge R\overline{z}\overline{y})](\overline{u}, \overline{v}).$$

- *Let $\mathfrak{A} := (A, <)$ be an ordered structure. The formula*

$$[\mathbf{lfp}_{R,x} \forall y(y < x \rightarrow Ry)](x)$$

  defines the well-founded part of $<$. The individual stages R^α contain all elements of height less than α.

- *Finally, we show that every regular language is definable in* LFP*. Let $\mathcal{A} := (Q, \Sigma, q_0, \delta, F)$ be a deterministic finite automaton with a finite set Q of states, a set $F \subseteq Q$ of final states, and the transition function δ. As Q is finite, we can quantify over a variable x_q for each $q \in Q$ such that all these variables denote distinct elements.*[2] *Therefore, we use notation like $\exists q \in F\, \varphi(q)$ to express that φ is satisfied by one of the variables x_q denoting a final state.*

[1]The transitive closure of a binary relation is formally defined in Definition 4.1.

[2]Note that his requires structures of size at least $|Q|$. However, there are only finitely many words of length less than $|Q|$ and thus the set of words of length less than $|Q|$ which are accepted by $\mathcal{A}$ is definable by a first-order formula.

Thus prepared, it is easily seen that the formula

$$\exists q_f \in F\ [\mathbf{lfp}_{R,x,q} \begin{pmatrix} (x = \min \wedge \bigvee_{a\in\Sigma}(P_a x \wedge q = \delta(q_0, a)) \\ \vee(\exists y \exists q' \operatorname{succ}(y) = x \wedge Ryq' \wedge \\ \bigvee_{a\in\Sigma}(P_a x \wedge q = \delta(a, q')) \end{pmatrix}](\max, q_f)$$

is satisfied by a word structure $\mathfrak{W}_w := (W, <, \min, \max, \operatorname{succ}, (Q_a)_{a\in\Sigma})$ *if, and only if,* w *is accepted by* $\mathcal{A}$.

In Example 3.17 below, we will see that the regular languages are even definable in monadic LFP.

3.3.1 Simultaneous Inductions

We now introduce the concept of simultaneous inductions. As the name suggests, simultaneous inductions allow us to build up simultaneously the fixed points of more than one formula. As we will see, for most fixed-point logics we consider here, this does not increase the expressive power and thus, strictly speaking, is not necessary. However, it often helps to present formulae in a more readable way and makes them easier to understand. Therefore, we will make extensive use of simultaneous formulae in the sequel. The definition of simultaneous inductions is presented in terms of the inductive definition of stages leading to the least fixed point and not in terms of least fixed points directly. As we have seen, the two concepts are equivalent and choosing one is merely a matter of style and preference.

3.16 Definition (Simultaneous least fixed-point logic). *Let* $R_1, \dots, R_k$ *be relation symbols of arities* r_i*, respectively. Simultaneous formulae are formulae of the form* $\psi(\overline{x}) := [\mathbf{lfp}\ R_i : S](\overline{x})$*, where*

$$S := \begin{cases} R_1\overline{x}_1 & \leftarrow & \varphi_1(R_1, \dots, R_k, \overline{x}_1) \\ & \vdots & \\ R_k\overline{x}_k & \leftarrow & \varphi_k(R_1, \dots, R_k, \overline{x}_k) \end{cases}$$

is a system of LFP*-formulae positive in all variables* R_i*. On a structure* $\mathfrak{A}$*, a formula* φ_i *in* S *induces an operator*

$$\begin{array}{rlcl} F_{\varphi_i} : & \operatorname{Pow}(A^{r_1}) \times \cdots \times \operatorname{Pow}(A^{r_k}) & \longrightarrow & \operatorname{Pow}(A^{r_i}) \\ & (R_1, \dots, R_k) & \longmapsto & \{\overline{a} : (\mathfrak{A}, R_1, \dots, R_k) \models \varphi_i[\overline{a}]\}. \end{array}$$

The stages S^α *of an induction on such a system* S *of formulae are* k*-tuples of sets* $(R_1^\alpha, \dots, R_k^\alpha)$ *defined as*

$$\begin{array}{rcll} R_i^0 & := & \varnothing & \\ R_i^{\alpha+1} & := & F_{\varphi_i}(R_1^\alpha, \dots, R_k^\alpha) & \\ R_i^\lambda & := & \bigcup_{\xi<\lambda} R^\xi & \text{for limit ordinals } \lambda. \end{array}$$

For every structure $\mathfrak{A} := (A, \tau)$ and any tuple $\overline{a}$ from A, $\mathfrak{A} \models \psi[\overline{a}]$ if, and only if, $\overline{a} \in R_i^\infty$, where R_i^∞ denotes the i-th component of the simultaneous fixed point of S.

Let S-LFP *denote the class of* LFP*-formulae with simultaneous inductions.*

The following example demonstrates the use of simultaneous fixed-point formulae.

3.17 Example. *Consider again the simulation of deterministic finite automata by formulae in* LFP *as presented in Example 3.15. We show that this simulation can also be done in simultaneous* monadic *least fixed-point logic. Let $\mathfrak{A} := (Q, \Sigma, q_0, \delta, F)$ be a deterministic finite automaton and consider the formula $\varphi := [\mathbf{lfp}\ Q_f : S](\max)$, where S is defined as the system containing a rule*

$$\begin{array}{lll} Q_q x & \leftarrow & (x = \min \wedge \bigvee_{a\in\Sigma, q=\delta(q_0,a)} P_a x) \vee \\ & & (\exists y\, \mathrm{succ}(y) = x \wedge \bigvee_{q=\delta(q',a)} (Q_{q'} y \wedge P_a x)) \end{array}$$

for every state $q \in Q$ and the additional rule $Q_f x \leftarrow \bigvee_{q\in F} Q_q x$. It is easily seen that on every word structure $\mathfrak{W}_w := (W, <, \min, \max, \mathrm{succ}, (P_a)_{a\in\Sigma})$ the fixed point Q_q^∞ of the simultaneous induction on S contains precisely the positions in the word w to which the automaton $\mathcal{A}$ assigns the state q. Thus, $\mathfrak{W}_w \models \varphi$ if, and only if, w is accepted by $\mathcal{A}$.

We show now that allowing simultaneous fixed points does not increase the expressive power of LFP, i.e. S-LFP = LFP.

3.18 Lemma. *For any system S of formulae in* LFP*, positive in their free fixed-point variables, $[\mathbf{lfp}\ R_i : S](\overline{t})$ is equivalent to a formula in* LFP *(without simultaneous inductions).*

Proof. Let

$$S := \begin{cases} R_1\overline{x}_1 & \leftarrow \quad \varphi_1(R_1, \dots, R_k, \overline{x}_1) \\ & \quad \vdots \\ R_{k-1}\overline{x}_{k-1} & \leftarrow \quad \varphi_{k-1}(R_1, \dots, R_k, \overline{x}_{k-1}) \\ R_k\overline{x}_k & \leftarrow \quad \varphi_k(R_1, \dots, R_k, \overline{x}_k) \end{cases}$$

be a system of formulae in LFP. Then $[\mathbf{lfp}\ R_1 : S]$ is equivalent to the formula $[\mathbf{lfp}\ R_1 : T]$, where

$$T := \begin{cases} R_1\overline{x}_1 & \leftarrow \quad \varphi_1(R_1, \dots, R_{k-1}, R_k\overline{u}/[\mathbf{lfp}_{R_k,\overline{x}_k}\ \varphi_k](\overline{u}), \overline{x}_1) \\ & \quad \vdots \\ R_{k-1}\overline{x}_{k-1} & \leftarrow \quad \varphi_{k-1}(R_1, \dots, R_{k-1}, R_k\overline{u}/[\mathbf{lfp}_{R_k,\overline{x}_k}\ \varphi_k](\overline{u}), \overline{x}_k). \end{cases}$$

Thus, the new system T is obtained from S by removing the rule for R_k and substituting in the formulae φ_1 to φ_{k-1} every occurrence of an atom $R\overline{u}$ by the least fixed point of φ_k. The correctness of this construction – sometimes referred to as the *Bekič-principle* – follows from the monotonicity of the involved operators. We omit the proof of this result and refer to [AN01, Lemma 1.4.2 on Page 27] instead. □

The next theorem follows immediately by induction on the structure of the formulae using the preceding lemma.

3.19 Theorem. *Simultaneous least fixed-point logic and least fixed-point logic are equivalent, i.e.* S-LFP = LFP.

Note that the translation of simultaneous LFP-formulae to simple formulae does not increase the arity of the involved fixed-point variables nor does it increase the number of alternations between fixed-point operators and negation. It does, however, affect the nesting depth of the fixed-point operators. As a consequence, the formula given in Example 3.17 above is equivalent to a formula in M-LFP without simultaneous inductions. Thus, every regular language is definable in M-LFP and therefore MSO – the monadic fragment of second-order logic – and M-LFP are equivalent on words.

In Section 3.4 we present a different proof of the theorem where the nesting depth of the formulae does not increase. The price we pay is that the arity of the involved fixed-point variables increases. Therefore, for logics like the modal μ-calculus, defined in Chapter 10, which only allow monadic fixed-point inductions, we have to use the method here to eliminate simultaneous inductions.

3.3.2 Alternation and Nesting in Least Fixed-Point Logic

In this section, we aim at identifying potential sources of complexity for least fixed-point formulae. The first is the *nesting depth*, which measures the number of fixed-point operators in an LFP-formula nested inside each other.

The second is the *alternation depth*. The alternation depth is not based on the number of nested fixed points, but on the number of alternations between negation symbols and fixed-point operators.

It turns out, that for least fixed-point logic, not nesting but alternation of fixed points is the source of expressive power and evaluation complexity.

3.20 Theorem (Transitivity Theorem). *Let $\varphi(R, Q, \overline{x})$ and $\psi(R, Q, \overline{y})$ be first-order formulae positive in R and Q such that no free first-order variable of ψ is bound in $\chi := [\mathbf{lfp}_{R,\overline{x}}\,\varphi(Q\overline{u}/[\mathbf{lfp}_{Q,\overline{y}}\psi](\overline{u}))](\overline{x})$. Then χ is equivalent to a formula with only one application of an* **lfp**-*operator.*

See [Mos74a, Theorem 1C.3] for a proof of the theorem. The presentation given here follows [EF99, Lemma 8.2.6 on Page 182]. The next corollary follows by induction on the number of fixed-point operators.

3.21 Corollary. *Every formula $\varphi \in \text{LFP}$ in which all fixed-point operators occur only positively is equivalent to a formula with only one application of a fixed-point operator.*

We show now that using fixed points negatively, i.e. allowing alternations between negation symbols and fixed-point operators, does increase the expressive power.

There are various definitions of the alternation hierarchy for LFP in the literature. Our presentation is based on the definition given in [Niw86]. See also [Bra98a] and references therein.

3.22 Definition (Defining formulae). *Let $\varphi \in \text{LFP}$ be a formula in* LFP *and let $\varphi_1, \dots, \varphi_k$ be the sub-formulae of φ of the form $[\mathbf{lfp}_{R_i,\overline{x}}\psi_i](\overline{t}_i)$. φ_i is called the* defining formula of R_i in φ. *We usually omit the reference to φ when it is clear from the context.*

We now define a partial order, the so-called *dependency order* $\sqsubseteq_\varphi$, on the fixed-point variables in a formula $\varphi \in \text{LFP}$ such that $X \sqsubseteq_\varphi Y$ if the induction on Y depends on the current stage of the induction on X.

3.23 Definition (Dependency order). *Let $\varphi \in \text{LFP}$ be a formula such that no fixed-point variable is bound twice in it and let $R_1, \dots, R_k$ be the fixed-point variables occurring in φ. For all i, let φ_i be the defining formula of R_i.* R_i depends directly on R_j in φ, *in terms $R_j \sqsubseteq^1_\varphi R_i$, if R_j occurs free in φ_i. Define $\sqsubseteq_\varphi$ as the transitive closure of $\sqsubseteq^1_\varphi$ and say* R_i depends on R_j in φ *just in case that $R_j \sqsubseteq_\varphi R_i$.*

Based on this dependency relation for the fixed point variables in a formula, we can now define the alternation and nesting-depth hierarchy of LFP.

3.24 Definition (Alternation and nesting-depth hierarchy). *Let $\varphi \in$* LFP *be a formula such that no fixed-point variable is bound twice in it and let $R_1, \dots, R_k$ be the fixed-point variables occurring in φ.*

(*i*) *The* nesting-depth of φ *is defined as the maximal cardinality of a subset of $\{R_1, \dots, R_k\}$ linearly ordered by $\sqsubseteq_\varphi$.*

(*ii*) *The* alternation-depth of φ *is defined as the maximal cardinality of a subset $\mathcal{M}$ of $\{R_1, \dots, R_k\}$, linearly ordered by $\sqsubseteq_\varphi$, such that, in addition, for all $R_i, R_j \in \mathcal{M}$ if R_i is a direct predecessor of R_j in $\mathcal{M}$ with respect to $\sqsubseteq_\varphi$ then the defining formula φ_j of R_j occurs negative in the defining formula φ_i of R_i.*

The n-th level of the alternation hierarchy $(\text{LFP}^a_n)_{n\in\omega}$ *of* LFP *is defined as the class of formulae in* LFP *of alternation-depth at most n. Analogously, the n-th level of the* nesting depth hierarchy $(\text{LFP}^d_n)_{n\in\omega}$ *of* LFP *consists of all formulae of* LFP *with nesting-depth at most n. A formula $\varphi \in \text{LFP}$ is called* alternation free *if it is contained in* LFP^a_1.

Note that the definition of nesting and alternation depth does not simply count the number of nested fixed points or the number of alternations on a purely syntactical basis. It also takes into account, whether the nested fixed points really depend on each other.

Obviously, the nesting-depth hierarchy is finer than the alternation hierarchy in the sense that a formula with alternation depth n also has nesting-depth at least n. Using simple diagonalisation arguments like the one in Chapter 7 below, it can be shown that in general the nesting-depth hierarchy is strict, i.e. there is no constant $k < \omega$ such that every LFP formula is equivalent to a formula with only k nested fixed points.

On the other hand, Theorem 3.20 implies that for LFP the nesting-depth hierarchy collapses to the alternation hierarchy. An immediate consequence of this is, that in general the alternation hierarchy is strict for LFP. (See [Mos74a, Chapter5] but also [Bra98a] for proofs of this.)

3.25 Theorem. *The alternation hierarchy for* LFP *is strict.*

The proof of this theorem uses a diagonalisation argument which relies on structures being infinite. And indeed, it has been shown by Immerman [Imm86] that on finite structures, the nesting depth and therefore also the alternation hierarchy collapses for LFP. A proof of the theorem can also be found in Section 8.1.

3.26 Theorem. *On finite structures, every formula in* LFP *is equivalent to a formula with at most one occurrence of an* **lfp**-*operator.*

3.3.3 Comparing the Stages

We close the section by stating the LFP version of the Stage Comparison Theorem 3.11.

3.27 Theorem (Stage Comparison Theorem for LFP). *Let $\varphi(R, \overline{x})$ be a* LFP*-formula positive in R. Then the stage comparison relations $\leq_\varphi$ and $\prec_\varphi$ are definable in* LFP.

Consider the system S of formulae defined as

$$S := \begin{cases} \overline{x} \leq \overline{y} & \leftarrow \quad \varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{y}) \wedge \varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{y}) \\ \overline{x} \prec \overline{y} & \leftarrow \quad \overline{x} \leq \overline{x} \wedge \neg\varphi(\overline{y}, R\overline{u}/\neg(\overline{x} \prec \overline{u} \vee \overline{x} \leq \overline{u})), \end{cases}$$

where $\varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{y})$ is obtained from φ by replacing each atom of the form $R\overline{u}$ by $\overline{u} \prec \overline{y}$. We claim that on any structure $\mathfrak{A}$, the simultaneous fixed point $(\leq^\infty, \prec^\infty)$ of the induction of S on $\mathfrak{A}$ contains precisely the stage comparison relations $\leq_\varphi$ and $\prec_\varphi$ on $\mathfrak{A}$.

The proof of this is rather technical and therefore omitted. Note that the result also follows from Theorem 8.7 below.

Least fixed-point logic extends first-order logic by an operator to define the least fixed point of operators induced by positive formulae. Clearly, this concept is not limited to first-order logic and other logical formalisms can be similarly extended by least fixed-point constructs. We will see another example of this in Chapter 10, where we consider the modal μ-calculus, a fixed point logic based on modal logic.

3.4 Inflationary Fixed-Point Logic

Least fixed-point logic is a powerful and important logic. However, the restriction to positive formulae allowed inside the fixed-point operators limits the logic to monotone inductions. To overcome this restriction, several alternative fixed-point concepts have been introduced. In this section, we consider the *inflationary fixed-point logic*, which extends FO by inflationary fixed points in the same way as LFP extends FO by least fixed points. This logic is not limited to monotone operators and in some sense is the simplest non-monotone fixed-point logic. We will consider even stronger logics in later chapters.

3.28 Definition. *Let A be a set and $F : \mathrm{Pow}(A) \to \mathrm{Pow}(A)$ be a function. F is* inflationary *if for all $X \subseteq A$, $X \subseteq F(X)$. F is* inductive *if the sequence X^α inductively defined by*

$$\begin{array}{rcll} X^0 & := & \varnothing & \\ X^{\alpha+1} & := & F(X^\alpha) & \\ X^\lambda & := & \bigcup_{\xi<\lambda} X^\xi & \quad \textit{for limit ordinals } \lambda \end{array}$$

is increasing, i.e. for all α, $X^\alpha \subseteq X^{\alpha+1}$. Obviously, any inflationary function F is inductive and on any set A, the sequence $(X^\alpha)_{\alpha\in\mathrm{Ord}}$ must reach a fixed point $X^\infty := X^\alpha$ for the least α such that $X^\alpha = X^{\alpha+1}$. The inflationary fixed point of *F is defined as the fixed point X^∞ of the sequence of sets induced by F.*

Clearly, any monotone or inflationary function is inductive but there are inductive functions which are neither monotone nor inflationary. Further, monotone functions need not to be inflationary and conversely, inflationary functions are not necessarily monotone. The following example demonstrates this.

3.29 Example. *Consider $A := \omega$, the set of finite ordinals.*

(i) The function F_1 taking the empty set to $\{0\}$, each singleton $\{n\}$ to $\{n+1\}$, and all other sets to $\varnothing$ is inductive but neither monotone nor inflationary.

(*ii*) *The function F_2 taking any set $X \subseteq A$ to the empty set is obviously monotone but not inflationary.*

(*iii*) *The function F_3 taking $\{0\}$ to $\{0, 1, 2\}$, $\{0, 1\}$ to $\{0, 1\}$, and all other sets to itself is inflationary but not monotone.*

(*iv*) *Finally, the function F_4 taking any set $X \neq \varnothing$ to $\varnothing$ and $\varnothing$ to $\{0\}$ is neither monotone, nor inflationary, nor inductive.*

As the example demonstrates, monotone, inflationary, and inductive functions are different concepts. However, when it comes to fixed-point logics based on these concepts, the difference between inductive and inflationary operators vanishes.

3.30 Definition (Inflationary Fixed-Point Logic). Inflationary fixed-point logic (IFP) *is defined as the extension of first-order logic by the following formula building rule. If $\varphi(R, \overline{x})$ is a formula with free first-order variables $\overline{x} := x_1, \dots, x_k$ and a free second-order variable R of arity k, then*

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

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

Let $\mathfrak{A}$ be a structure with universe A providing an interpretation of the free variables of φ except for $\overline{x}$. Obviously, the operator I_φ defined as

$$I_\varphi(R) := R \cup F_\varphi(R) = \{\overline{a} \in A^k : \overline{a} \in R \text{ or } (\mathfrak{A}, R) \models \varphi[\overline{a}]\}$$

is inflationary and thus has an inflationary fixed point R^∞. For any tuple $\overline{t}$ of terms, $\mathfrak{A} \models [\mathbf{ifp}_{R,\overline{x}}\, \varphi](\overline{t})$ if, and only if, $\overline{t}^{\mathfrak{A}} \in R^\infty$.

Again, the monadic fragment of IFP is of particular interest.

3.31 Definition (M-IFP). Monadic inflationary fixed-point logic (M-IFP) *is defined as the fragment of* IFP *where all fixed-point variables are monadic, i.e. of arity one.*

A question that naturally arises is whether we indeed have defined something new, i.e. whether LFP and IFP are different. Obviously, LFP $\subseteq$ IFP as the least fixed point of a monotone operator is also the inductive and hence the inflationary fixed point. In Chapter 8 we will see that the converse also holds, i.e. that the two logics are actually equivalent. Showing this, however, requires some more effort.

In spite of LFP and IFP being equivalent, least and inflationary inductions have rather different properties and inflationary inductions are generally more expressive. We demonstrate this by an example.

3.32 Example. *Let $\Sigma := \{a, b, d\}$ be an alphabet and consider the class $\mathcal{W}$ of word structures $\mathfrak{W}_w := (W, <, P_a, P_b, P_d)$ such that $w \in (a \cup b)^* d (a \cup b)^*$. The class $\mathcal{C} \subseteq \mathcal{W}$ of structures representing words $w := vdv$, for some $v \in (a \cup b)^*$, is definable in* M-IFP *but not in* M-LFP.

Clearly, the language represented by $\mathcal{C}$ is not regular and thus not definable in MSO. *As* M-LFP $\subseteq$ MSO*, it follows that $\mathcal{C}$ is not* M-LFP*-definable. However, it is defined in* M-IFP *by the formula $\varphi := \forall x \neq d\, [\mathbf{ifp}_{R,x}\, \psi](x)$, with*

$$\psi := \begin{array}{c} \exists x_1 \notin R \wedge \forall y < x_1\, Ry \wedge \exists x_2 \notin R \wedge \forall y (d < y < x_2 \rightarrow Ry) \wedge \\ P_a x_1 \leftrightarrow P_a x_2 \wedge P_b x_1 \leftrightarrow P_b x_2 \wedge x = x_1 \vee x = x_2), \end{array}$$

where, to simplify notation, d is used as a constant that denotes the unique letter labelled by d.

On a word $w = w_1 d w_2$ the formula simultaneously scans the sub words w_1 and w_2 from the left to the right. At each stage, the first line of ψ defines the two positions x_1, x_2 in w_1 and w_2 respectively, which have not yet been read but such that all positions to the left of x_1, x_2 have already been considered.

The second part of ψ ensures that x_1 and x_2 are added to R if, and only if, they are labelled by the same letter.

Obviously, for every word $w = w_1 d w_2$ such that $\mathfrak{W}_w \in \mathcal{C}$, the formula eventually adds all positions except d to the fixed point and thus $\mathfrak{W}_w \models \varphi$. Conversely, if $\mathfrak{W}_w \models \varphi$, then w must be of the form vdv and thus $\mathfrak{W}_w \in \mathcal{C}$.

Note how the formula uses the conjunction $x_1 \notin R \wedge \forall y < x_1\, Ry$ to say that x_1 is not yet contained in the fixed point but will enter in the next step. This is a general technique for inflationary fixed-point logic that will be applied at various places in the sequel. In general, if $\varphi(R, \overline{x})$ is a formula defining a fixed-point induction, then for any given stage R^α, the formula $\neg R\overline{x} \wedge \varphi(R, \overline{x})$ defines the tuples $\overline{x} \in R^{\alpha+1} - R^\alpha$. In a limited form, this gives the formula access to the individual stages of the fixed-point induction, something that is impossible in least fixed-point logic.

Besides this more technical advantage of inflationary inductions, it turns out that it is often easier to formalise properties in IFP than in LFP. The main advantage of LFP, however, is that it allows a simpler evaluation. In particular for least fixed-point formulae with a limited number of alternations between fixed-point operators and negation the (practical) evaluation complexity is much better than for IFP.

We will see numerous further examples showing that least and inflationary fixed point inductions have very different properties. And indeed, if the fixed-point inductions are not considered in the context of full first-order logic but on restricted variants of FO, for instance modal logic, then inflationary fixed points turn out to be far more expressive than least fixed points. This will further be explored in Part II.

Simultaneous fixed points. As with least fixed-point logic, we also allow simultaneous inductions in IFP, and again these do not increase the expressive power. The proof, however, is completely different as for LFP. As the formulae used to build up the fixed points are no longer monotone, simultaneous fixed points cannot be pushed inside each other. Examples for the failure of the Bekič-principle for inflationary inductions can easily be found. See also Section 11.4.

3.33 Theorem. *Simultaneous inflationary fixed-point logic* (S-IFP) *and inflationary fixed-point logic* (IFP) *have the same expressive power.*

Proof. The proof is by induction on the structure of formulae in S-IFP. In the main step, let $\varphi := [\mathbf{ifp}\ R_i : S](\overline{t})$ be a formula, where

$$S := \begin{cases} R_1\overline{x}_1 & \leftarrow & \varphi_1(R_1, \dots, R_k, \overline{x}_1) \\ & \vdots & \\ R_k\overline{x}_k & \leftarrow & \varphi_k(R_1, \dots, R_k, \overline{x}_k) \end{cases}$$

is a system of formulae in IFP. Without loss of generality, we assume that all R_i are of the same arity r and all $\overline{x}_i$ are of the form $x_1, \dots, x_r$. Let $\overline{x}$ be a $(r+1)$-tuple of variables and R be a $r+1$-ary relation symbol not occurring in any of the φ_i.

The idea is that instead of having k distinct fixed-point variables $R_1, \dots,$ R_k we only take one, of enhanced arity, and use markers $1, \dots, k$ in its last component so that at each stage α, the projection of R^α on those tuples, where the last component is i, is just the stage R_i^α. For this, consider the formula $\vartheta := \exists 1 \dots \exists k\ (\bigwedge_{1 \le i \ne j \le k} i \ne j) \land [\mathbf{ifp}_{R,\overline{x}}\, \psi(R, \overline{x})](\overline{t}')$, where $\overline{t}' := \overline{t}i$ and

$$\psi(R, \overline{x}) := \bigvee_{i=1}^{k} x_{r+1} = i \land \varphi_i(\overline{x}, R_j u_1 \dots u_r / R u_1 \dots u_r j).$$

Let $\mathfrak{A}$ be a structure with at least k elements. We claim that for all $i \in \{1, \dots, k\}$, all $\overline{a} \in A^r$, and all $\alpha \in \mathrm{Ord}$,

$$\overline{a} \in (S^\alpha)_i \iff (\overline{a}, i) \in R^\alpha.$$

The claim is easily proved by induction on α. Thus, φ and ϑ are equivalent on structures with at least k elements. On structures with less than k elements, the theorem is trivial as the fixed point defined by φ is already definable in first-order logic. □

Alternation and Nesting in Inflationary Fixed-Point Logic. In Section 3.3.2 we studied the impact of alternation and nesting of fixed points on the expressive power of LFP. As we have seen, nested positive least fixed points can always be eliminated in favour of a single fixed point. On the other hand, alternation between negation and fixed points can not be eliminated. We show now, that the situation for IFP is rather different.

3.34 Theorem. *Every* IFP*-formula is equivalent to a formula where negation occurs only in front of atoms. In particular, the alternation hierarchy*[3] *for* IFP *collapses.*

Proof. The theorem is proved by induction on the structure of the formulae. For the case of **ifp**-operators, note that a formula $\neg[\mathbf{ifp}_{R,\overline{x}}\,\varphi](\overline{t})$ is equivalent to the simultaneous fixed point $[\mathbf{ifp}\ Q : S](\overline{t})$ of the system

$$S := \begin{cases} R\overline{x} & \leftarrow \varphi(R, \overline{x}) \\ Q\overline{x} & \leftarrow \forall \overline{y}\,(\varphi(R, \overline{y}) \rightarrow R\overline{y}) \wedge \neg R\overline{x}. \end{cases}$$

On structures with at least two elements this is equivalent to the inflationary fixed point of a single formula whereas on structures with only one element, IFP collapses to FO anyway and the theorem is trivial. □

As with LFP, simple diagonalisation arguments as used in Chapter 7 show that the nesting depth hierarchy for IFP does not collapse in general.

3.35 Proposition. *The nesting-depth hierarchy for* IFP *is strict.*

In Chapter 8, we will establish a tight correspondence between the LFP-alternation hierarchy and the IFP nesting-depth hierarchy by showing that for all structures $\mathfrak{A}$ either both hierarchies collapse on $\mathfrak{A}$ or both are strict. See Theorem 8.8 and Corollary 8.9 for a proof.

However, the normal form proved for LFP on finite structures, also holds true for IFP. See [Imm86] for a proof of this theorem, which can also be found in Section 8.1.

3.36 Theorem. *On finite structures, every formula in* IFP *is equivalent to a formula with at most one fixed-point operator.*

Inflationary Stage Comparison. We close the section by stating the stage comparison theorem for IFP. Recall the definition of the relations $\prec_\varphi$ and $\leq_\varphi$ for positive formulae φ (see Definition 3.9). Clearly, the same definition also applies for formulae φ which are not required to be positive in their fixed-point variables. In this case, the induction stages refer to the inflationary instead of the least fixed-point induction.

We have already seen how the stage comparison relations of a least fixed-point induction can be defined in LFP. We now prove the analogous theorem for inflationary fixed-point logic. As it turns out, defining the stage comparison relations for an inflationary induction is even simpler than the analogous definition for LFP.

[3] The definition of the alternation and nesting-depth hierarchy for IFP is analogous to Definition 3.24.

3.37 Theorem. *For every formula $\varphi(R, \overline{x}) \in$ IFP the stage comparison relations $\prec_\varphi$ and $\leq_\varphi$ are definable in* IFP.

Proof. W.l.o.g. we assume that φ is of the form $R\overline{x} \vee \varphi'$. We claim that the relations $\leq_\varphi$ and $\prec_\varphi$ can be obtained as the simultaneous fixed point of the following system S of formulae:

$$S := \begin{cases} \overline{x} \leq \overline{y} & \longleftarrow \quad \varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{y}) \wedge \varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{y}) \\ \overline{x} \prec \overline{y} & \longleftarrow \quad \varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{x}) \wedge \neg\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x}). \end{cases}$$

Here, $\varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{y})$ means that every occurrence of an atom $R\overline{u}$ in φ, for some tuple of terms $\overline{u}$, is replaced by the new atom $\overline{u} \prec \overline{y}$.

For every ordinal α, let $\leq^\alpha$ and $\prec^\alpha$ denote the relations $\leq$ and $\prec$ at stage α of the induction on S and let $\leq^{<\alpha}$ and $\prec^{<\alpha}$ be the union of all stages less than α, i.e. $\leq^{<\alpha} = \bigcup_{\beta<\alpha} \leq^\beta$ and $\prec^{<\alpha} = \bigcup_{\beta<\alpha} \prec^\beta$. We prove by induction that for all α and all pairs $(\overline{a}, \overline{b})$,

- $(\overline{a}, \overline{b}) \in \leq^\alpha$ if, and only if, $|\overline{b}|_\varphi \leq \alpha$ and $|\overline{a}|_\varphi \leq |\overline{b}|_\varphi$ and
- $(\overline{a}, \overline{b}) \in \prec^\alpha$ if, and only if, $|\overline{a}|_\varphi \leq \alpha$ and $|\overline{a}|_\varphi < |\overline{b}|_\varphi$.

From this, the theorem follows immediately. Let α be an ordinal and suppose that for all $\beta < \alpha$ the claim has been proved, i.e. $(\overline{a}, \overline{b}) \in \leq^{<\alpha}$ if, and only if, $|\overline{a}|_\varphi \leq |\overline{b}|_\varphi < \alpha$ and likewise for $\prec^{<\alpha}$.

Suppose $\overline{b}$ is a tuple of elements of rank $\xi \leq \alpha$. Then, the set $\{\overline{u} : \overline{u} \prec^{<\alpha} \overline{b}\}$ contains precisely the elements of rank less than ξ. Thus, $\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{y})$ is satisfied by $\overline{b}$. Set $\overline{y} := \overline{b}$. A tuple $\overline{a}$ satisfies $\varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{y})$ if, and only if, the rank of $\overline{a}$ is at most ξ and therefore $|\overline{a}|_\varphi \leq |\overline{b}|_\varphi$.

On the other hand, if the rank of $\overline{b}$ is greater than α, then $\{\overline{u} : \overline{u} \prec^{<\alpha} \overline{b}\}$ is just $\varphi^{<\alpha}$ and therefore $\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{y})$ is not satisfied by $\overline{b}$. This proves the induction hypothesis for the first item above.

For $\prec$, let $\overline{a}$ be a tuple of elements of rank $\xi \leq \alpha$. Again, $\{\overline{u} : \overline{u} \prec^{<\alpha} \overline{a}\}$ contains all elements of rank less than ξ and therefore $\varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{x})$ is satisfied by $\overline{a}$. Obviously, if we set $\overline{x} := \overline{a}$, then $\neg\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x})$ is satisfied by those tuples $\overline{b}$ whose rank is greater than ξ and therefore greater than the rank of $\overline{a}$. Finally, if $\overline{a}$ is a tuple of rank greater than α, it does not satisfy $\varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{x})$. This proves the second item above and, with it, the claim.

Thus, the stage comparison relations $\leq_\varphi$ and $\prec_\varphi$ are defined by the IFP-formulae $[\textbf{ifp} \leq : S](\overline{x}, \overline{y})$ and $[\textbf{ifp} \prec : S](\overline{x}, \overline{y})$ respectively. $\square$

Note that the formula defining the non-strict stage comparison relation $\leq$ depends on the fixed-point variable $\prec$ and therefore on the induction on $\prec$. However, the second formula, defining $\prec$, does not use the variable $\leq$. Therefore, the strict relation $\prec$ can be defined by the inflationary fixed-point of the second formula without any need to simultaneously define $\leq$. This will be used in Chapter 8 below.

Right from the definition, the syntax of IFP is much more liberal than the syntax of LFP and as we have seen above, the two logics have rather different properties in terms of alternation and nesting depth but also in various other aspects. Nevertheless, we will show below that they are actually equivalent. In this sense, IFP is the simplest fixed-point logic which extends LFP towards non-monotone inductions. We discuss various more expressive logics below.

3.5 Inductive Fixed Points and Second-Order Logic

In this section we compare least and inflationary fixed-point logic with second-order logic. A first hint how to link LFP with SO was already given in Theorem 3.3 above. There, the least fixed point of a monotone operator F on a set A was defined as

$$\mathbf{lfp}(F) := \bigcap\{X \subseteq A : F(X) = X\} = \bigcap\{X \subseteq A : F(X) \subseteq X\}.$$

It follows immediately, that LFP $\subseteq$ SO, as every formula $\varphi \in$ LFP can be converted into an equivalent SO-formula φ' by iteratively replacing subformulae of the kind $[\mathbf{lfp}_{R,\overline{x}}\,\psi](\overline{u})$ by $\forall R\,((\forall\overline{x}\psi(R,\overline{x}) \rightarrow R\overline{x}) \rightarrow R\overline{u})$.

3.38 Proposition. *Every formula $\varphi \in$ LFP is equivalent to a formula ψ in second-order logic. Further, if k is the maximal arity of a fixed-point variable in φ, then ψ can be chosen so that all second-order variables in ψ are at most k-ary. In particular, monadic least fixed-point logic* (M-LFP) *is contained in monadic second-order logic* (MSO).

For inflationary fixed points there is no such second-order characterisation and indeed it can be shown that monadic inflationary fixed-point logic (M-IFP) is not contained in MSO. This follows immediately from Example 3.32, where we have seen that the language $w \cdot d \cdot w \subseteq (a \cup b)^*d(a \cup b)^*$ is definable in M-IFP. As this is not a regular language it is not definable in MSO.

Using results from descriptive and computational complexity theory, it can easily be shown that in the restriction to finite structures, IFP $\subseteq \Delta^1_0$, i.e. every least fixed-point formula is equivalent over finite structures to both, an universal and an existential second-order formula. For this, note that every class of structures definable in IFP is in PTIME and thus in NP as well as CO-NP (see Chapter 5). As every class of structures decidable in NP is definable in Σ^1_1 it follows that IFP $\subseteq \Sigma^1_1$ and likewise for CO-NP and Π^1_1.

The result can also be established by a direct argument, without the help of descriptive complexity theory. See [DG02] for details.

On infinite structures, the Σ^1_1 bound fails and we are only able to prove the following result. See again [DG02] for details.

3.39 Theorem. *Every formula in* IFP *is equivalent to a formula in* Δ^1_2*, i.e. equivalent to both, a formula in* Σ^1_2 *and* Π^1_2*.*

In general, the result cannot be improved any further as IFP is not contained in Δ^1_1: On the structure $(\omega, <)$ every Π^1_1 formula is equivalent to a formula in LFP (see [Mos74a]). As LFP is closed under negation, this implies that every formula in $\Sigma^1_1 \cup \Pi^1_1$ is equivalent to a formula in LFP. Finally, $\Sigma^1_1 \neq \Pi^1_1$ and therefore $\Delta^1_1 \subsetneq \Sigma^1_1$ and IFP $\not\subseteq \Delta^1_1$ (see [Kle55]).

Combining this with the following theorem establishing the Löwenheim-Skolem-property for IFP, we are able to give a precise bound for the complexity of the satisfiability problem for IFP, i.e. the complexity of the problem to decide for a given IFP-formula whether it is satisfiable. The following theorem is well known in the theory of fixed-point logics and its proof is therefore omitted. See [Flu95] and also [Grä02].

3.40 Theorem (Löwenheim-Skolem Property). *Inflationary fixed-point logic has the Löwenheim-Skolem Property, i.e. every satisfiable* IFP*-formula has a finite or countable model.*

The following corollary follows immediately.

3.41 Corollary. *The satisfiability problem* Sat(IFP) *for* IFP *is in* Σ^1_2*, the second level of the analytical hierarchy. Further,* Sat(IFP) *is not in* Δ^1_1 *and therefore not in* Σ^1_1*, i.e. it is not in the first level of the analytical hierarchy. In particular,* Sat(IFP) *is not arithmetical.*

Proof. Let φ be a formula in IFP. By Theorem 3.39, it is equivalent to a formula ψ in Σ^1_2 and, by Theorem 3.40 above, if φ is satisfiable then it has a countable model. Thus, there is a sentence $\chi \in \Sigma^1_2$ saying that there is a countable structure which is a model of ψ. Clearly, χ is true in $(\mathbb{N}, <, +, \cdot)$ if, and only if, φ is satisfiable.

Towards the lower bound, we have seen that IFP is not contained in Δ^1_1 and consequently neither in Σ^1_1 nor in Π^1_1. A direct proof showing that Sat(IFP) is not arithmetical can be found in Section 11.1 below. $\square$

In Chapter 8, we will show that LFP and IFP are equivalent. Thus the complexity bounds for Sat(IFP) apply to LFP also.

Chapter 4

Other Fixed-Point Logics

4.1 Fragments of Least Fixed-Point Logic

For many practical applications, the fixed-point logics introduced so far are already much too powerful and complex. Therefore, a variety of logics have been introduced which are weaker than LFP in terms of expressive power and complexity. We now present some of these logics, which will be used later.

4.1.1 Transitive Closure Logics

In many applications, inductive processes are used to find paths in graphs. Examples are navigation systems, trying to find connections between cities, or routing systems, that find paths through a network. Applications like this are quite common and essentially the complexity classes LOGSPACE and NLOGSPACE consist of problems of this kind. It is therefore natural to study an extension of first-order logic by operators so that such problems become definable. A closer look at the anatomy of the examples shows, that the essential operation common to all these problems is to form the transitive closure of a graph.

4.1 Definition (Transitive Closure). *Let A be a set and let for some k, $E \subseteq A^k \times A^k$ be a binary relation on k-tuples of elements from A. The* transitive closure TC(E) *of E is defined as the least set* $\mathrm{TC}(E) \subseteq A^k \times A^k$ *such that*

- $E \subseteq \mathrm{TC}(E)$ *and*
- *for any $\overline{a}, \overline{b}, \overline{c} \in A^k$, if $(\overline{a}, \overline{b}) \in \mathrm{TC}(E)$ and $(\overline{b}, \overline{c}) \in \mathrm{TC}(E)$ then also $(\overline{a}, \overline{c}) \in \mathrm{TC}(E)$.*

The deterministic transitive closure ($\mathrm{DTC}(E)$) *of E is defined as the transitive closure of the relation*

$$E_d := \{(\overline{a}, \overline{b}) \in E : \textit{ for all } \overline{c} \textit{ such that } (\overline{a}, \overline{c}) \in E, \overline{b} = \overline{c}\}.$$

We now introduce *(deterministic) transitive closure logic* ((D)TC), obtained from FO by adding operators for defining the (deterministic) transitive closure of definable graphs.

4.2 Definition (TC and DTC). Transitive closure-logic (TC) *is defined as the extension of first-order logic by the following formula building rule. If $\overline{x}$ and $\overline{y}$ are tuples of variables of the same arity and $\varphi(\overline{x},\overline{y})$ is a formula such that $\overline{x},\overline{y}$ are among its free variables, then $\psi := [\mathbf{tc}_{\overline{x},\overline{y}}\,\varphi](\overline{t},\overline{t}')$ is also a formula, where $\overline{t},\overline{t}'$ are tuples of terms of the same arity as $\overline{x}$ and $\overline{y}$. The free variables of ψ are the free variables in $\overline{t}$ and $\overline{t}'$ and the variables free in φ other than $\overline{x}$ and $\overline{y}$.*

The semantics of TC *is defined inductively. Let $\psi(\overline{x},\overline{y}) := [\mathbf{tc}_{\overline{x},\overline{y}}\,\varphi](\overline{x},\overline{y})$ be a formula in* TC *and let k be the arity of $\overline{x}$. On any structure $\mathfrak{A}$ providing an interpretation of the free variables of φ except for $\overline{x}$ and $\overline{y}$, φ defines a binary relation $\varphi^{\mathfrak{A}} := \{(\overline{a},\overline{b}) : \mathfrak{A} \models \varphi(\overline{a},\overline{b})\}$ on k-tuples of elements. Now, $\mathfrak{A} \models \psi(\overline{a},\overline{b})$ if, and only if, $(\overline{a},\overline{b}) \in TC(\varphi^{\mathfrak{A}})$.*

Analogously, the deterministic transitive-closure logic (DTC) *is defined as the extension of* FO *by formulae of the form $[\mathbf{dtc}_{\overline{x},\overline{y}}\varphi](\overline{t},\overline{t}')$ defining the deterministic transitive-closure of the graph defined by φ.*

Clearly, every formula $[\mathbf{dtc}_{\overline{x},\overline{y}}\varphi(\overline{x},\overline{y})](\overline{t},\overline{t}')$ is equivalent to the TC-formula

$$[\mathbf{tc}_{\overline{x},\overline{y}}\,\varphi(\overline{x},\overline{y}) \wedge \forall\overline{z}(\varphi(\overline{x},\overline{z}) \rightarrow \overline{z} = \overline{y})](\overline{t},\overline{t}')$$

Thus, DTC $\subseteq$ TC. In Example 3.15, we have seen that the transitive closure of a relation defined by a formula $\varphi(\overline{x},\overline{y})$ is definable in LFP by

$$[\mathbf{lfp}_{R,\overline{x},\overline{y}}\,\varphi(\overline{x},\overline{y}) \vee \exists\overline{z}(R\overline{x}\overline{z} \wedge R\overline{z}\overline{y})](\overline{u},\overline{v}).^{1}$$

It immediately follows that TC $\subseteq$ LFP. Examples of TC-formulae can be found in Chapter 17, where we show that TC is expressive complete on the class of linear constraint databases.

4.1.2 Existential and Stratified Fixed-Point Logic

In this section we consider a hierarchy of logics inside LFP that originates from database theory. The study of fixed-point logics in database theory has concentrated on fixed-point extensions of conjunctive queries, which correspond, in the notation of first-order logic, to existential positive first-order formulae. Positive, here, means that no negation at all occurs in the formulae.

One of the best known fixed-point logics in this context is the query language DATALOG, which extends conjunctive queries by least fixed-points.

[1]Without proof we remark that the converse is also true, i.e. every formula in LFP of the form $[\mathbf{lfp}_{R,\overline{x}}\varphi_0(\overline{x}) \vee \exists\overline{z} \in R\varphi_1(\overline{x},\overline{z})](\overline{x})$, where R does not occur in φ_0 and φ_1, is equivalent to a formula in TC. See e.g. [EF99, Proposition 8.7.1, page 235-236].

There are various definitions of DATALOG depending on whether or not they allow the negation of input predicates (the relations present in the structure). Here, we consider the variant where the negation of relation symbols from the signature of the structure is allowed. To keep consistent with the notation used for the other fixed-point logics, we do not present the syntax of DATALOG as common in database theory but consider the corresponding first-order based fixed-point logic, the *existential fixed-point logic.*

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

The following proposition lists some simple properties of existential least fixed-point logic. The simple proofs are omitted. See [Gro97] for more on E-LFP. Further information can also be found in [EF99].

4.4 Proposition.

- *Existential least fixed-point logic is closed under extensions, i.e. if* $\mathfrak{A} \subseteq \mathfrak{B}$ *is a substructure of* $\mathfrak{B}$ *and* $\varphi \in$ E-LFP *a sentence such that* $\mathfrak{A} \models \varphi$*, then also* $\mathfrak{B} \models \varphi$.
- *The closure ordinal of an* E-LFP*-formula is at most* ω.
- FO $\not\subseteq$ E-LFP.

As the proposition shows, E-LFP does not contain first-order logic. As both, DTC and TC do contain FO, it follows immediately that TC $\not\subseteq$ E-LFP. The converse, i.e. E-LFP $\not\subseteq$ TC has been proved by Immerman [Imm81].

4.5 Proposition. *In terms of expressive power,* TC *and* E-LFP *are incomparable, i.e. neither* TC $\subseteq$ E-LFP *nor* E-LFP $\subseteq$ TC.

We now introduce a logic that extends E-LFP such that again all of first-order logic becomes definable. The logic – *stratified fixed-point logic* – also has an analogue in database theory, namely stratified DATALOG.

Stratified DATALOG has been defined as an extension of DATALOG to allow some restricted form of negation. The idea is that once a DATALOG-formula is completely evaluated, the relation defined by it may be used in other DATALOG-programs just like an ordinary relation from the structure. In particular, it can be used negatively without the need to ever recompute it in the evaluation of the formula using it.

Therefore, a formula or program in stratified DATALOG is built up in stages, the strata, where in each stratum, the formula may use – positively and negatively – the fixed-point relations built up in the lower but not the variables defined in higher strata.

As with DATALOG, we do not consider stratified DATALOG directly but present a first-order based fixed-point logic equivalent to it.

4.6 Definition (Stratified fixed-point logic). *Define* SFP_1 *as existential fixed-point logic and* SFP_{i+1} *as the class of existential fixed-point formulae which may use literals of the form* $\psi(\overline{x})$*, where, for some* $l \leq i$*,* ψ *is a* SFP_l *formula without free fixed-point variables. Note that these formulae may occur under the scope of negation symbols.*

Stratified fixed-point logic *is defined as the union* $\mathrm{SFP} := \bigcup_{i \in \omega} \mathrm{SFP}_i$ *of all* SFP_i*.*

We now list some properties of stratified fixed-point logic.

4.7 Proposition.

- *On all structures, the closure ordinal of every* SFP*-formula is at most* ω*.*
- $\mathrm{TC} \subsetneq \mathrm{SFP}$*.*
- $\mathrm{SFP} \subsetneq \mathrm{LFP}$*.*

Clearly, SFP is contained in LFP and it has been shown by Kolaitis [Kol91] that this inclusion is strict. Further, we already mentioned that there are properties definable in E-LFP but not in TC. Thus we get the following relations between $\mathrm{TC}, \mathrm{SFP}$, and LFP.

4.8 Proposition. $\mathrm{TC} \subsetneq \mathrm{SFP} \subsetneq \mathrm{LFP}$ *and also* $\text{E-LFP} \subsetneq \mathrm{SFP} \subsetneq \mathrm{LFP}$*. Further,* E-LFP *and* TC *are incomparable.*

With $\mathrm{TC}, \text{E-LFP}$, and SFP, we introduced several fixed-point logics that bridged the gap between first-order and least fixed-point logic. In the next section, we go beyond LFP and consider a logic whose expressive power presumably exceeds that of LFP and IFP.

4.2 Partial Fixed-Point Logic

In this section, we introduce *partial fixed-point logic (*PFP*)*, which in some sense is the most general fixed-point extension of first-order logic. The syntax of PFP is defined as for inflationary fixed-point logic, except that we write **pfp** for the fixed-point operator.

4.1 Definition (Syntax of Partial Fixed-Point Logic). Partial fixed-point logic (PFP) *is defined as the extension of first-order logic by the following formula building rule. If* $\varphi(R, \overline{x})$ *is a formula with free first-order variables* $\overline{x} := x_1, \ldots, x_k$ *and a free second-order variable* R *of arity* k*, then*

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

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

Partial fixed-point logic originates from finite model theory where it has been introduced as a logic capable of defining properties beyond PTIME. And indeed, it has been shown that PFP captures PSPACE on finite ordered structures. (See Chapter 5 for details.) However, one effect of the exclusive focus on finite models is that the common semantics of PFP is limited to finite structures and does not extend to arbitrary structures. As we are also interested in the study of fixed-point logics on potentially infinite structures, we will define an alternative semantics for PFP in Section 7.1. This semantics will be equivalent to the standard semantics on finite structures but does generalise to infinite structures. For now, we present the standard definition of the partial fixed-point semantics as it is common in finite model theory.

4.9 Definition (Finite Model Semantics). *Let $\psi := [\mathbf{pfp}_{R,\overline{x}}\,\varphi](\overline{t})$ be a formula and let $\mathfrak{A}$ be a finite structure with universe A, providing an interpretation of the free variables of φ other than $\overline{x}$. Consider the following sequence of stages induced by φ on $\mathfrak{A}$.*

$$\begin{array}{rcl} R^0 & := & \varnothing \\ R^{\alpha+1} & := & F_\varphi(R^\alpha) \end{array}$$

As there are no restrictions on φ, this sequence may not reach a fixed point. In this case, ψ is equivalent on $\mathfrak{A}$ to false. Otherwise, i.e. if the sequence becomes stationary and reaches a fixed point R^∞, then for any tuple $\overline{a} \in A$,

$$\mathfrak{A} \models [\mathbf{pfp}_{R,\overline{x}}\,\varphi](\overline{a}) \text{ if, and only if, } \overline{a} \in R^\infty.$$

The following example demonstrates the potential power of partial fixed-point logic.

4.10 Example. *Consider the formula φ over the signature $\tau := \{<\}$ defined as*

$$\varphi(x) := [\mathbf{pfp}_{R,x} \begin{array}{l} \forall x\, Rx \lor (\exists y\, \neg Ry \land \forall z < y\, Rz \ \land \\ (x = y \lor (x > y \land Rx))) \end{array}](x).$$

Let $\mathfrak{A} := (A, <)$ be a finite ordered structure and let $n := |A|$ be its cardinality. Any subset $B \subseteq A$ can naturally be interpreted as the binary encoding of an integer between 0 and 2^n–1. In the sequence of stages, φ defines an enumeration of all binary strings of length at most n. At the end of the induction, R contains the encoding of the number 2^n–1, i.e. $R^\infty = A$. Thus, on any finite ordering $\mathfrak{A}$, the formula φ simply defines the universe of $\mathfrak{A}$ and is equivalent to true. But before that, it runs through an exponential number of stages.

As in inflationary fixed-point logic, we allow simultaneous inductions and again these can always be eliminated in favour of simple inductions.

4.11 Theorem. *Simultaneous partial fixed-point logic and partial fixed-point logic have the same expressive power.*

The negation normal form proved for IFP in Theorem 3.34 holds true for PFP also. Thus, the alternation hierarchy collapses for PFP.

4.12 Theorem. *Every* PFP*-formula is equivalent to a formula where negation occurs only in front of atoms.*

Further, similar to LFP and IFP, every partial fixed-point formula is equivalent on finite structures to a formula with only one fixed-point operator. See [EF99] for a proof of this theorem.

4.13 Theorem. *Every formula* $\psi \in$ PFP *is equivalent to a formula with only one application of a fixed-point operator.*

Clearly, on finite structures, every IFP formula is equivalent to a formula in PFP. As the example demonstrates, there are partial fixed-point inductions of exponential length, whereas every inflationary fixed-point induction must reach its fixed point after a polynomial number of stages. This makes partial fixed-point logic potentially more powerful than inflationary fixed-point logic. However, although the fixed point of a partial fixed-point induction might need an exponential number of stages to be reached, it is still of polynomial size. Therefore, it might be possible to define the same fixed point with a number of stages polynomial in the size of the structure. As we will see below, the question whether IFP = PFP is closely related to a major open problem in complexity theory, namely whether PTIME equals PSPACE.

The potential power of PFP is based on the fact that elements contained in some stage of the fixed point induction need not to be in later stages also. A consequence of this is that there no longer is a unique stage at which an element enters the fixed point. Therefore, for partial fixed-point inductions, the stage comparison method does not apply, depriving us of one of the most fundamental tools for analysing fixed point inductions.

Chapter 5

Descriptive Complexity

In the preceding chapters, we have seen a variety of fixed-point logics and established a number of results about their relationship in terms of expressive power. However, so far we have said nothing about their complexity, i.e. the complexity of evaluating a formula from a particular logic.

There are various different questions that arise in connection with logics and complexity, e.g. whether two formulae of a logic are equivalent, whether one implies the other, or whether a formula is satisfiable, i.e. has a model. All these questions can be reduced to each other wherefore usually only the satisfiability problem is considered.

5.1 Definition. *The* satisfiability problem (Sat($\mathcal{L}$)) *for a logic $\mathcal{L}$ is defined as the problem of deciding for a given formula $\varphi \in \mathcal{L}$ whether φ has a model.*

For instance, Corollary 3.41 shows that the satisfiability problem for inflationary fixed-point logic is in the second level of the analytical hierarchy but not in the first. Another important question is the complexity of evaluating a given formula φ in a structure $\mathfrak{A}$. This is referred to as the *evaluation* or *model checking problem* for the logic $\mathcal{L}$.

Whereas the satisfiability problem makes sense for finite as well as infinite structures, the evaluation problem is a problem of finite model theory. We can measure its complexity in three different ways.

5.2 Definition. *Let $\mathcal{L}$ be a logic. The* evaluation *or* model checking problem for *$\mathcal{L}$ is defined as the problem of deciding for a given finite structure $\mathfrak{A}$ and a sentence $\varphi \in \mathcal{L}$ whether $\mathfrak{A} \models \varphi$.*

- *The* data complexity *of this problem is defined as the amount of resources (e.g. time, space, or number of processors) needed to decide this problem for a fixed formula φ. In particular, the complexity is measured only in the size of the structure.*
- *The* expression complexity *of the problem is defined as the amount of resources needed to decide the problem for a fixed structure $\mathfrak{A}$. The*

complexity is measured only in the size or structural complexity of the formula.

- *Finally, if both, the structure and the formula is part of the input, we speak of the* combined complexity *of the problem. Here, the complexity is measured both in the size of the structure and the formula.*

In a slight abuse of notation, we speak about the *evaluation problem* of a logic $\mathcal{L}$ whenever we consider its *data complexity* and call it *model checking problem* when referring to *combined complexity.*

When considering data complexity, we fix a formula and let the structures vary. This corresponds to the complexity analysis of programs or algorithms, where for a fixed program, i.e. Turing-machine, the complexity is measured in terms of the size of the input. This leads to results like the algorithm being a polynomial time algorithm meaning that the class of words accepted by the Turing-machine can be decided in time polynomial in the input size.

It also makes sense to ask the other way round: Given a class of structures whose membership problem is in a particular complexity class, say, in polynomial time, can it be defined in the given logic. Or is it even true that *all* problems decidable in the complexity class can be defined in this logic.

In a situation like this, where a logic $\mathcal{L}$ can express all problems decidable in a complexity class $\mathcal{C}$ and where further the evaluation problem of this logic is in $\mathcal{C}$ itself, we say that $\mathcal{L}$ *captures* $\mathcal{C}$. This is a desirable situation as it means that the logic provides a logical characterisation of the problems in $\mathcal{C}$. However, there is a subtle point worth mentioning. When we talk about Turing-machines accepting an input, we always think about the input given as a word on the machine's input tape. On the other hand, when speaking about models of a formula, we usually do not think about words but about more complex structures like graphs, databases and so on. The crucial difference is that the representation of a structure by a word on the input tape implicitly defines an ordering on the structure, even if the structure itself is unordered. Thus, the Turing-machine always works on (representations of) ordered structures, whereas the logic has to deal with unordered structures also.

This is a major obstacle when proving capturing results for certain logics. For instance, it has been shown by Immerman and Vardi [Imm86, Var82] that least fixed-point logic captures PTIME on the class of finite ordered structures. However, no such capturing result is known for PTIME on the class of all finite structures.

Encoding the input on a Turing-tape will also be the main problem we have to deal with in Chapter 18, where we prove capturing results on classes of constraint databases, a model for infinite databases.

In the next section, we consider the complexity of the evaluation problem for formulae in the various logics we considered so far. In the section thereafter, we mention several capturing results known for these logics. None of the results is novel to this thesis. Therefore, we only sketch the proofs or give no proofs at all. For more details and full proofs we refer to [EF99], [Imm98] and references therein.

5.1 Evaluation Complexity of Fixed-Point Formulae

In this section, we consider the evaluation complexity of fixed-point logics. We start the analysis with first-order logic. Clearly, if the formula and thus the number of quantifiers is fixed, the evaluation of a first-order formula can be done in LOGSPACE: We just need one pointer to the structure for each of the variables in the formula. As the number of quantifiers is fixed, this results in a constant number of pointers and therefore the evaluation can be done in logarithmic space.[1]

Now consider a formula $\psi(\overline{x}) := [\mathbf{lfp}_{R,\overline{x}}\, \varphi(R, \overline{x})](\overline{x})$, where R is a k-ary relation variable. To evaluate this formula, we repeatedly have to evaluate φ on the different stages of the induction. Now, if $\mathfrak{A}$ is a finite structure of size n, then there are no more than n^k stages. Therefore, if the evaluation complexity of φ is bound by a polynomial n^l, the whole cost of evaluating ψ is bound by $n^k \cdot n^l = n^{k+l}$. By induction on the number of fixed-point operators, we get that every formula in LFP or IFP can be evaluated in polynomially many steps. This establishes the following theorem.

5.3 Theorem. *The evaluation problem of least and inflationary fixed-point logic is in* PTIME.

Obviously, the complexity bounds hold also true for the fragments of LFP we mentioned in Section 4.1. However, for TC and DTC we can actually prove better bounds.

Consider a formula $\psi := [\mathbf{tc}_{\overline{x},\overline{y}}\, \varphi(\overline{x}, \overline{y})](\overline{t}, \overline{s})$. Evaluating such a formula boils down to finding a path from $\overline{t}$ to $\overline{s}$ in the graph defined by φ. This, of course, can be done in non-deterministic logarithmic space, provided that φ can be evaluated in NLOGSPACE also. By induction on the number of **tc**-operators it follows, that the evaluation complexity of TC-formulae is in NLOGSPACE. Consequently, if the graph defined by φ is deterministic and φ can be evaluated in LOGSPACE, then the evaluation problem for $[\mathbf{dtc}_{\overline{x},\overline{y}}\, \varphi]$ is in LOGSPACE also.

[1]Actually, the data complexity of FO is even less, namely in AC^0 – the class of problems that can be solved by a family of circuits of constant depth and arbitrary fan-in. AC^0 is a complexity class at the bottom end of the hierarchy of complexity classes and is strictly contained in LOGSPACE.

5.4 Theorem. *The evaluation problem for* TC *is in* NLogspace. *For* DTC, *the evaluation problem is in* Logspace.

Finally, we consider partial fixed-point logic. In PFP, there is no polynomial bound on the number of stages a formula can take to define its fixed point. However, each individual stage is still of polynomial size and can therefore be stored within polynomial space. This yields the following result.

5.5 Theorem. *The evaluation problem for* PFP *is in* Pspace.

We now turn towards establishing lower bounds for the complexity of the evaluation problem of the logics considered above.

5.2 Logics Capturing Complexity Classes

In the previous section, we established upper bounds for the evaluation complexity of various fixed-point logics. It can easily be shown that these complexity bounds are strict, e.g. there are formulae in LFP whose evaluation problem is Ptime-complete. However, we aim at even stronger results, namely that not only there are formulae in LFP whose evaluation problem is complete for Ptime, but that for every class of finite ordered structures, which can be decided in polynomial time, there is a formula in LFP defining it. In a situation like this, we say that the logic (LFP) *captures* the complexity class (Ptime) on the class of finite ordered structures. We will see, that for all logics we mentioned above, such capturing results can be established.

5.6 Definition. *A logic $\mathcal{L}$* captures *a complexity class $\mathcal{C}$ on a class $\mathcal{S}$ of structures if for every vocabulary τ*

- *and every sentence $\varphi \in \mathcal{L}[\tau]$, the problem of deciding for a structure $\mathfrak{A} \in \mathcal{S}$ whether $\mathfrak{A} \models \varphi$ is in $\mathcal{C}$ and*
- *for every class $\mathcal{K} \subseteq \mathcal{S}$ of τ-structures whose membership problem is in $\mathcal{C}$ there is a sentence in $\mathcal{L}[\tau]$ such that for all τ-structures $\mathfrak{A} \in \mathcal{S}$*

$$\mathfrak{A} \in \mathcal{K} \textit{ if, and only if, } \mathfrak{A} \models \varphi.$$

It has been shown that for the class $\mathcal{S}$ of finite ordered structures all major complexity classes can be captured by natural extensions of first-order logic.

5.7 Theorem. *On the class of finite ordered structures,*

- DTC *captures* Logspace, *[Imm87b]*
- TC *captures* NLogspace, *[Imm88, Imm87b]*

- LFP *and* IFP *capture* PTIME, *[Imm86, Var82]*
- Σ^1_1, *i.e. the existential fragment of second order logic, captures* NP *[Fag74], and*
- PFP *captures* PSPACE *[AV89].*

The various capturing results can all be proved along the same line by simulating the run of a Turing-machine on a structure. We only sketch the proofs here. See [EF99] and [Imm98] for full proofs. An adaptation of the method can be found in Chapter 18, where we establish a capturing result for PTIME on the class of constraint databases.

As mentioned above, Turing-machines do not directly work on structures but on their encoding on an input tape. Therefore, to simulate the run of a Turing-machine on a structure, we need three separate formulae: The first defines an encoding of the input structure by a word. The second formula defines the start configuration of the Turing-machine on this encoding. Finally, the third formula defines the transition relation between two configurations. This done, one can use the fixed-point construct present in the logic, i.e. the transitive-closure or fixed-point operator, to simulate the sequence of configurations the Turing-machine takes on (the encoding of) the structure. In the case of Σ^1_1 one existentially quantifies this sequence of configurations.

The different computational power present in the various fixed-point constructs determines the class of problems for which the run of a Turing-machine can be simulated.

An immediate consequence of the theorem above is, that one way of separating the complexity classes mentioned there, is to separate the corresponding logics on finite ordered structures. However, a number of properties of finite ordered structures make them hard to attack by standard techniques used to prove non-expressibility results.

It would therefore be interesting if a separation of complexity classes could also be derived from a separation of logics on classes of arbitrary rather than ordered finite structures. Abiteboul and Vianu showed that for polynomial time and space, this can indeed be done. They proved that a separation of IFP and PFP on arbitrary finite structures would be sufficient to separate PTIME and PSPACE.

5.8 Theorem. *Inflationary and partial fixed-point logic are equivalent on finite structures if, and only if,* PTIME *equals* PSPACE.

For a proof of this theorem see [AV91]. See also [DLW95, Daw93].

Chapter 6

Fixed-Point Logics with Choice

We have already seen a variety of fixed-point logics with various degrees of expressiveness. In terms of complexity and expressive power on finite ordered structures, these logics provide precise characterisations of complexity classes such as polynomial time or space and deterministic or non-deterministic logarithmic space. However, there is still a huge gap between PTIME and PSPACE with important complexity classes such as NP or the polynomial-time hierarchy. In this section, we close this gap by defining a fixed-point logic whose fragments in terms of alternation depth capture the levels of the polynomial-time hierarchy. In particular, the positive fragment of the logic captures NP.

There are different attempts to capture the non-deterministic behaviour of NP by logics. The first and most important logic is Σ^1_1, the existential fragment of second-order logic. By Fagin's theorem, we know that Σ^1_1 captures NP on the class of all finite structures. However, Σ^1_1 is a second-order logic and does not fit well into the uniform framework of fixed-point logics. In particular, methods such as stage comparison and other methods unique to fixed-point logics do not apply to Σ^1_1. Therefore, attempts have been made to capture NP by fixed-point logics. In general, there are two different lines along which such logics can be defined. The first is the so-called *formula non-determinism* in contrast to what is called *data non-determinism.*

Fixed-point logics based on formula non-determinism have been considered by Abiteboul, Vardi, and Vianu in [AVV97]. Here, the inflationary fixed-point is defined by two formulae. At each stage, a non-deterministic choice is made on the formula to be used for defining the next stage. The resulting relation is defined as the union of all fixed points reached by the various possible choices. The fragment of this logic, where only positive applications of fixed-point operators are allowed, is shown to capture NP on finite ordered structures. Similarly, a logic called *alternating fixed-point logic*

(AFP) is defined which chooses the formulae not only on a non-deterministic basis but uses alternating choice. It is shown that this logic captures alternating polynomial time, and thus PSPACE. See also [DG02] for more details on these logics.

In contrast to the formula non-determinism, where the choice is made on the formulae used to define the stages, in *data non-determinism* there is only one formula that defines the fixed point but the choice is made on the elements to be added during the fixed-point induction. There are two different approaches towards fixed-point logics implementing data non-determinism. The first is the logic NIO introduced by Arvind and Biswas in [AB87]. The syntax of NIO is as of IFP except that the fixed-point operator is written as **nio**. Semantically, at each stage of the fixed-point induction induced by a formula $[\mathbf{nio}_{R,\overline{x}}\,\varphi]$, exactly one tuple of the set of tuples satisfying φ is chosen and added to the fixed point.

The second logic of this kind is C-IFP, where the fixed point is defined by an induction on two formulae φ and ψ. At each stage, the second formula defines a set of which non-deterministically one tuple is chosen. Based on this choice, the first formula defines the next stage of the induction.

The power of both logics lies in the non-deterministic choice of tuples. In this way, a distinction between otherwise indistinguishable elements, for instance automorphic elements, is possible. In particular, this can be used to define a linear order on an unordered structure. It can also be used to guess arbitrary sets or to model the non-deterministic choice made by a non-deterministic Turing-machine. This is used to show that both logics capture NP, even on the class of arbitrary, potentially unordered, finite structures. See [DR03] for precise definitions and various further results on the two logics.

Both logics have in common, that they resolve the non-determinism of nested formulae only at the outermost level, i.e. on top of the syntax tree. This makes it possible to allow closure under negation while staying inside NP. In fact, it has been shown in [DR03] that negated fixed points can be eliminated in these logics.

6.1 Choice Fixed-Point Logic

In this section we introduce another fixed-point logic utilising data non-determinism. Contrary to the two logics mentioned above, the non-determinism here is resolved at each fixed-point operator. Therefore, the fragment of the logic capturing NP is closed under negation if, and only if, NP = CO-NP.

6.1 Note. *In this chapter we only consider finite structures. Most of the results extend to infinite structures but if unbounded choice on infinite structures is allowed, effects that are not topic of this work have to be considered.*

We now give precise definitions and present some straightforward results.

6.2 Definition (Syntax of CFP). *The logic* CFP *is defined as the extension of first-order logic by the following formula building rule. If $\varphi(R, \overline{x}, \overline{c})$ is a formula with a free l-ary second-order variable R, a free l-tuple of first-order variables $\overline{x}$, and a tuple of free variables $\overline{c}$ distinct from $\overline{x}$, then $\psi := [\mathbf{cfp}^{\overline{c}}_{R,\overline{x}}\varphi](\overline{t})$ is also a formula in* CFP*, where $\overline{t}$ is a l-tuple of terms. Note that the tuples $\overline{x}$ and $\overline{c}$ might differ in their length.*

The semantics of the formulae is defined inductively, with the semantics of the **cfp**-rule being as follows. Essentially, $[\mathbf{cfp}\ \varphi]$ defines the inflationary fixed-point of φ, but, in addition, at each step an arbitrary tuple of elements is chosen for $\overline{c}$. This makes the logic non-deterministic and – as we will see – substantially increases its expressive power.

6.3 Definition (Semantics of CFP). *Let $\psi := [\mathbf{cfp}^{\overline{c}}_{R,\overline{x}}\varphi(R, \overline{x}, \overline{c})](\overline{t})$ be a formula in* CFP *and let $\mathfrak{A}$ be a structure with universe A. Further, let $\nu = (\overline{a}_\alpha)_{\alpha\in\mathrm{Ord}}, \overline{a}_\alpha \in A^k$, be a sequence of tuples from A of the same length as $\overline{c}$. We inductively define the ν-*relativised fixed point of *φ as follows:*

$$\begin{array}{lll} R^0_\nu & := & \varnothing \\ R^{\alpha+1}_\nu & := & R^\alpha_\nu \cup \{\overline{a} : (\mathfrak{A}, R^\alpha_\nu) \models \varphi[\overline{a}, \overline{a}_{\alpha+1}]\} \\ R^\lambda_\nu & := & \bigcup_{\xi<\lambda} R^\xi_\nu, \qquad \textit{for limit ordinals } \lambda \end{array}$$

*For the least ordinal α such that $R^\alpha_\nu = R^{\alpha+1}_\nu$, we call R^α_ν the ν-*relativised fixed point of *φ and denote it as $F^\infty_\varphi(\nu)$. The* inflationary fixed-point with choice, in terms *F^∞_φ*, of *φ is defined as*

$$F^\infty_\varphi := \{\overline{a} : \begin{array}{l} \textit{there is a sequence } \nu := (\overline{a}_\alpha)_{\alpha\in\mathrm{Ord}}, \overline{a}_\alpha \in A^k, \\ \textit{such that } \overline{a} \in F^\infty_\varphi(\nu) \end{array}\}.$$

Now, $\mathfrak{A} \models [\mathbf{cfp}^{\overline{c}}_{R,\overline{x}}\varphi(R, \overline{x}, \overline{c})](\overline{t})$ if, and only if, $\overline{t} \in F^\infty_\varphi$.

We present some examples to get acquainted with CFP.

6.4 Example.

- *Let $\mathfrak{A}$ be a finite structure with universe A. Consider the formula φ defined as*

$$\exists x\, [\mathbf{cfp}^{c_1,c_2}_{R,x,y} \begin{array}{l} (c_1 \neq c_2 \wedge \text{“}c_1 \notin R \wedge c_2 \notin R\text{”} \wedge x = c_1 \wedge y = c_2) \vee \\ (\forall x \exists y (Rxy \vee Ryx) \wedge x = y) \end{array}](x, x).$$

  where “$c_1 \notin R \wedge c_2 \notin R$” is an abbreviation for $\neg\exists z(Rc_1z \vee Rzc_1 \vee Rc_2z \vee Rzc_2)$ stating that neither c_1 nor c_2 occur in some tuple in R.

  We claim that $\mathfrak{A} \models \varphi$ if, and only if, $|A|$ is even. For this, consider the sequence of stages induced by φ for some choice of elements. At each stage, the two elements chosen for c_1 and c_2 are added to R, provided that they are different and have not been chosen before. This process continues until no pair of elements is left to be chosen. If $|A|$ is even, then R contains all elements and thus the conjunct $\forall x \exists y (Rxy \vee Ryx)$

in the second line becomes true and all pairs (x, x) are added to R. Otherwise, i.e. if $|A|$ is odd, the induction stops. Thus, independent of the choice of elements, a pair (x, x) occurs in the fixed point if, and only if, $|A|$ is even.

- *The second example demonstrates how an ordering of a structure $\mathfrak{A}$ can be defined in* CFP*. For this, let $\mathcal{C}$ be a class of ordered structures defined by a sentence φ in* IFP*. We claim that the class*

  $$\mathcal{C}^{\leq} := \{\mathfrak{A} : \text{ there is a linear order } \leq \text{ on } \mathfrak{A} \text{ and } (\mathfrak{A}, \leq) \in \mathcal{C}\}$$

  is definable in CFP *by the (simultaneous) formula*

  $$\exists x\,[\mathbf{cfp}^c\ Q : \begin{array}{lcl} Rxy & \leftarrow & \neg Rcc \wedge y = c \wedge (Rxx \vee x = c) \\ Qx & \leftarrow & \forall x\, Rxx \wedge \varphi(u \leq v / Ruv) \end{array}](x).$$

  Clearly, the formula is equivalent to a formula without simultaneous inductions. Consider the sequence of stages induced by the formula for a particular choice of elements. Provided that the element chosen for c at a particular stage had not been chosen before, the rule $\neg Rcc \wedge y = c \wedge (Rxx \vee x = c)$ adds all pairs (x, c) to R such that either x occurs in R – and is thus smaller than c in the ordering built up in R – or $x = c$. In this way, a linear order is defined in R. As soon as the order is total, i.e. all x occur in R and $\forall x\, Rxx$ becomes true, Q becomes non-empty if, and only if, the formula φ evaluates to true, where all references in φ to the ordering $\leq$ are replaced by references to R.

As the example demonstrates, the class Even of structures of even cardinality is definable in CFP. As it is known not to be definable in PFP, it follows that on arbitrary structures, CFP is not contained in PFP.

6.5 Corollary. CFP $\not\subseteq$ PFP.

In the definition of CFP, we allowed the tuple $\overline{c}$ to be of arbitrary length. Another natural way of defining the logic is to restrict $\overline{c}$ to a single variable. That is, at each stage not a tuple but only a single element is chosen. We call this logic CFP$'$ and show that both ways of defining CFP are equivalent.

6.6 Lemma. *On finite structures,* CFP *and* CFP$'$ *are equally expressive.*

Proof. Clearly, every CFP$'$-formula is equivalent to a formula in CFP. The converse is proved by induction on the structure of the formula. As Example 6.4 above demonstrates, a linear order on the structure is definable in CFP$'$ and therefore we can restrict attention to ordered structures. The only interesting case are formulae $[\mathbf{cfp}^{\overline{c}}_{R,\overline{x}}\varphi](\overline{t})$, where φ is already a CFP$'$-formula. We simulate the choice of a k-tuple $\overline{c}$ by a sequence of k steps of a CFP$'$-formula. In each step, one of the components of $\overline{c}$ is guessed and stored in the fixed-point relation.

W.l.o.g. we assume that $|\mathfrak{A}| \geq k$. Let l be the arity of $\overline{x}$ and let P be a $(2l{+}1)$-ary fixed-point variable. We define a formula $\psi(P, \overline{x}, \overline{s}, i)$ such that ψ defines the following sequence of stages P^α. The induction on P consists of blocks, each containing $k+1$ stages. In each block, one individual stage of the induction on φ is simulated. The tuple $\overline{s}$ contains the number of the stage in the induction on φ and is invariant within each block. In the first k stages $1 \leq i \leq k$ of each block, tuples $(c, \overline{x}, \overline{s}, i)$ are added to P where c is the element chosen in this stage and $\overline{x}$ are arbitrary. elements. Thus, after k stages a k-tuple $\overline{c}$ has been chosen and stored in the first components of the tuples added in this block. In the last step, all $(\overline{x}, \overline{s}, (k+1))$ are added to P such that $\overline{x}$ satisfies $\varphi(\overline{x}, \overline{c})$ where all atoms $R\overline{u}$ are replaced by $\exists \overline{v} < \overline{s} R(\overline{u}, \overline{v}, (k+1)))$.

The formula $\chi := \exists \overline{s}\, [\mathbf{cfp}^c_{P,\overline{x},\overline{s},i}\, \psi](\overline{t}, \overline{s}, k+1)$ implements this idea, where ψ is defined as

$$\begin{aligned}
&P = \varnothing \wedge \overline{s} = \overline{0} \wedge i = 0 \wedge \textstyle\bigwedge_{j=0}^{k} x_j = c \vee \\[1ex]
&P \neq \varnothing \wedge \text{“}\exists \overline{s}' \text{ maximal in } P\text{”} \wedge \text{“}\exists i' \text{ maximal in } P\_\overline{s}'\_\text{”} \wedge \\
&\qquad\qquad i' < k \rightarrow (i = i' + 1 \wedge \overline{s} = \overline{s}' \wedge x_0 = c) \wedge \\
&\qquad\qquad i' = k \rightarrow (i = k+1 \wedge \overline{s}' = \overline{s} \wedge \exists \overline{c} \textstyle\bigwedge_{j=1}^{k} (\exists \overline{x}\, P\overline{x}\overline{s}j \wedge c_j = x_0) \wedge \\
&\qquad\qquad\qquad \varphi(x, \overline{c}, R\overline{u}/\exists \overline{v} < \overline{s}' P(\overline{u}, \overline{v}, (k+1)))) \wedge \\
&\qquad\quad i' = k+1 \rightarrow (\overline{s} = \overline{s}' + 1 \wedge i = 0 \wedge x_0 = c).
\end{aligned}$$

Here, “$\exists \overline{s}'$ maximal in P” means that $\overline{s}'$ is the greatest tuple such that there are $\overline{x}$ and i with $(\overline{x}, \overline{s}', i) \in R$. Analogously, $\exists i'$“maximal in $P\_\overline{s}'\_$” means that i' is the supremum of all i such that $(\overline{x}, \overline{s}', i) \in R$ for some $\overline{x}$.

A straightforward induction on the stages shows that for all stages α of the induction on φ, if $\overline{s}$ is the α-th tuple in the ordering, then for all $\overline{a}$, $\overline{a} \in R^\alpha$ if, and only if, there is some $\overline{v} \leq \overline{s}$ such that $(\overline{a}, \overline{v}, (k{+}1)) \in P^{\alpha\cdot(k+1)}$. This finishes the proof of the lemma. $\square$

As the previous lemma shows, in terms of expressive power, it makes no difference whether we allow a single element or a whole tuple to be chosen at each step.

6.1.1 Simplifying the First-Order Quantifier Structure

We show next that first-order quantifiers can be eliminated in CFP using the choice construct. Obviously, every formula $\exists x \varphi(x, \overline{y}) \in$ CFP is equivalent to $[\mathbf{cfp}^c_R \varphi(c, \overline{y})](\overline{y})$, where R is a 0-ary relation symbol. As universal quantifiers can be eliminated using negation and existential quantification, the following corollary follows immediately.

6.7 Corollary. *Every formula in* CFP *is equivalent to a quantifier-free formula.*

However, we are mainly interested in fragments of CFP which limit the number of alternations between fixed-point operators and negation. Thus, eliminating universal quantification by introducing further negations is not satisfactory.

On ordered structures where the order is given by a successor function with constants n and m for the minimal and maximal element, universal quantification can easily be eliminated using that $\forall x \varphi(x, \overline{y})$ is equivalent to

$$[\mathbf{cfp}^{c}_{R,x}(x = \mathrm{n} \land \varphi(x,\overline{y})) \lor (Rc \land x = \mathrm{succ}(c) \land \varphi(x,\overline{y}))](\mathrm{m}). \tag{6.1}$$

On unordered structures, or structures where the ordering is not given by a successor function, we can still define an ordering as in Example 6.4. However, the construction used universal quantification and also defined an order and not a successor relation. Whereas the formula given there can easily be modified so that it defines the successor and not an order relation, we will not be able to eliminate all universal quantifiers. However, for many applications, showing that there is a small constant k such every formula in a logic is equivalent to one with less than k quantifiers is as good as showing that every formula is equivalent to a quantifier-free formula.

For CFP, such a constant k indeed exists, in fact: $k = 2$. For this, let $\varphi(\overline{x})$ be a formula in CFP. We shall convert φ into an equivalent formula with only two universal quantifiers. To increase readability, we present the following formulae as simultaneous fixed-point formulae but they can easily be transformed into simple formulae using one additional existential quantifier. Consider the formula

$$\psi := [\mathbf{cfp}\ R : T](\overline{x})$$

where T is a system of formulae defined as

$$T := \begin{cases} Sxy & \leftarrow \quad \neg Scc \land y = c \land (\varphi_{\max}(x) \lor x = c)) \\ R\overline{x} & \leftarrow \quad \exists n \exists m\, \varphi_{\mathrm{comp}}(n,m) \land \varphi'(\overline{x}). \end{cases}$$

Here $\varphi'(\overline{x})$ is the formula obtained from φ by first eliminating all universal quantifiers in φ as indicated by the formula (6.1) above (using a successor function *succ*) and then replacing every atom $(succ(u) = v)$ by $Suv \land u \neq v$. The auxiliary formulae $\varphi_{\max}(x)$ and φ_{comp} are defined as follows: $\varphi_{\max}(x)$ is defined as $\varphi_{\max} := (Sxx \land \forall y \neg Sxy)$ and is true for x at a particular stage of the induction on S if x is the maximal element in the fragment of the successor relation built up in S so far. Finally, the formula

$$\varphi_{\mathrm{comp}}(n,m) := \forall x(Sxx \land (x \neq n \to \neg Sxn) \land (x \neq m \to \neg Smx))$$

tests whether the successor relation in S is complete and in this case defines n and m to be the minimal and maximal element of this relation.

The formula ψ simultaneously builds up the fixed points of two formulae in the relations S and R. In S, a successor relation is defined as follows. At each stage, it is checked whether c is already contained in S and in case that not, the pairs (c, c) and (x, c) are added to S, where x is the maximal element in the relation build up so far. The addition of (c, c) is necessary as it allows to check whether an element x does not occur in S by using the atom $\neg Sxx$ instead of involving an additional universal quantifier.

For R, the formula φ_{comp} first checks whether the fixed point of S has been reached. In this case, n and m are defined to be the minimal and maximal element in S and the formula φ' is evaluated. Thus, the fixed point of R contains all elements that satisfy φ'. As φ' and φ are equivalent, R^∞ contains the elements satisfying φ. Note that in φ' every atom $(succ(u) = v)$ is replaced by $Suv \wedge u \neq v$. The conjunct $\neg u = v$ is necessary as S was constructed to be reflexive.

By construction of φ', the formula ψ only contains two universal quantifiers. Combining this with the elimination of existential quantifiers as indicated above, the next theorem follows immediately.

6.8 Theorem. *Every formula in* CFP *is equivalent to a formula with only two universal and no existential quantifiers. Further, if φ is a formula with k alternations between* **cfp***-operators and negation such that the outermost fixed point is positive, then φ is equivalent to a formula with the same alternation structure using only two quantifiers, both universal.*

Note, however, that with each eliminated quantifier we introduce another fixed-point operator. So, complexity-wise we gain nothing. However, when proving that a certain property is not definable in CFP, it might be helpful to know that we can do with only two quantifiers.

6.1.2 Choice Fixed-Point and Second-Order Logic

We now study the connection between CFP and second-order logic. In particular, we show that there is a close correspondence between the levels of the alternation hierarchy for CFP – see Definition 3.24 – and the levels of the quantifier-alternation hierarchy for second-order logic.

6.9 Lemma. Σ^1_1 *is equivalent to the positive fragment of* CFP*, i.e. the class of formulae in* CFP *where no fixed-point operator occurs negatively.*

Proof. The proof of both directions is by induction on the structure of the formulae. Towards the forth direction, let X be a k-ary relation symbol and let $\psi := \exists X \varphi(X)$ be a formula such that φ is in CFP. We claim that ψ is equivalent to the formula

$$\varphi(\varnothing) \vee \exists \overline{x} \exists y\, x_1 \neq y \wedge [\mathbf{cfp}^{\overline{c}}_{R,\overline{x},y} \begin{array}{l} (\overline{c}c_1 \notin X \wedge \overline{x} = \overline{c} \wedge y = x_1) \vee \\ (\overline{c}c_1 \in X \wedge \varphi(X\overline{u}/R\overline{u}u_1)) \end{array}](\overline{x}, y),$$

where $\overline{c} := c_1, \ldots, c_k$ is a tuple of variables and $\overline{c}c_1$ denotes the tuple $c_1, \ldots, c_k, c_1$. Here, $\varphi(\varnothing)$ denotes the formula obtained from φ by replacing every atom $X\overline{u}$ by *false*.

The first disjunct, $\varphi(\varnothing)$, becomes true if φ holds for X being empty. The induction induced by the second disjunct can be split into two parts. First, a non-empty set X is guessed and built up inductively. This is done by adding at each stage a tuple $(\overline{c}, c_1)$ to R. As soon as a tuple is chosen for $\overline{c}$ a second time, the first part of the induction is finished and the set X built up in R is used to check whether φ is satisfied.

If φ holds for this set, all possible tuples are added to the fixed point. Otherwise, nothing is added. In either case, no further tuples are added in successive stages and the fixed point is reached. Thus, a tuple $(\overline{x}, y)$ with $y \neq x_1$ occurs in the fixed point if, and only if, φ holds for some $X \neq \varnothing$.

For the converse, let $\psi := [\mathbf{cfp}^{\overline{c}}_{R,\overline{x}}\varphi](\overline{x})$ be a formula such that φ is in Σ^1_1. Let $\overline{c}$ be l-ary and $\overline{x}$ be k-ary. To see that ψ is equivalent to a formula in Σ^1_1, let O, C, S be relation symbols, where O and S are $2k$-ary and C is $(k + l)$-ary. Consider the formula ϑ informally defined as

$$\begin{array}{ll}
\exists O & \text{“}O \text{ is a linear order on } k\text{-tuples”} \wedge \\
\exists C & \text{“}C \text{ is a function from } k\text{-tuples to } l\text{-tuples”} \wedge \\
\exists S & (\forall\overline{x} S\overline{0}\overline{x} \leftrightarrow \exists\overline{c}\, C\overline{0}\overline{c} \wedge \varphi(\overline{x}, \overline{c}, \varnothing)) \wedge \forall\overline{s}\forall\overline{s}'(\overline{s} = \overline{s}' + 1 \rightarrow \\
& (\forall\overline{x}(S\overline{s}\overline{x} \leftrightarrow (X\overline{s}'\overline{x} \vee \exists\overline{c}\, C\overline{s}\overline{c} \wedge \varphi(\overline{x}, \overline{c}, R\overline{u}/X\overline{s}'\overline{u}))))).
\end{array}$$

On any finite structure $\mathfrak{A}$ of size n, the induction on φ must reach its fixed point after at most n^k steps. Thus, the number of each stage can be identified with a k-tuple of elements.

The formula ϑ first defines in O an order on the k-tuples. The functional relation C then associates with each stage α, coded by a k-tuple $\overline{s}$, the l-tuple of elements chosen for $\overline{c}$ at this stage. Finally, the $2k$-ary relation S stores for each number $\overline{s}$, corresponding to a stage α, the elements contained in the α-th stage of the induction on φ assuming the choice of elements made in C. From this, the fixed point of φ can easily be defined. $\square$

The following corollaries follow immediately.

6.10 Corollary. CFP *and* SO *are equivalent.*

Let CFP_k denote the k-th level of the alternation hierarchy for CFP and let CFP^+_k be the class of formulae in CFP_k where the outermost fixed points occur positive. Let CFP^-_k denote the class of CFP-formulae φ such that $\neg\varphi \in \mathrm{CFP}^+_k$, i.e. where the outermost fixed points are negative.

6.11 Corollary.

- CFP^+_k *captures* Σ^1_k*, the k-th level of the polynomial-time hierarchy.*
- CFP^-_k *captures* Π^1_k*, the universal k-th level of the polynomial-time hierarchy.*

- *In this sense,* CFP *captures* PH *on the class of finite structures. In particular,* CFP_1^+ *captures* NP *on arbitrary finite structures and* CFP_1^- *captures* CO-NP.

6.1.3 Arity-Restricted CFP and Transitive-Closure Logic

We now consider fragments of CFP_1^+ based on the arity of the involved fixed-point operators. In particular, we show that every formula in transitive-closure logic is equivalent to a formula in monadic CFP_1^+. Let $\mathrm{M\text{-}CFP}_1^+$ denote this fragment of CFP_1^+, i.e. the class of formulae in CFP_1^+ where all fixed-point variables are unary.

6.12 Theorem. *On finite ordered structures, every formula in* TC *is equivalent to a formula in* $\mathrm{M\text{-}CFP}_1$.

Proof. It is known that on finite ordered structures, every TC formula is equivalent to a formula $\chi := [\mathbf{tc}_{\overline{x},\overline{y}}\,\varphi(\overline{x},\overline{y})](\overline{u},\overline{v})$ with first-order kernel φ. (See e.g. [Imm98] and [EF99].) We claim that χ is equivalent to a formula in $\mathrm{M\text{-}CFP}_1^+$.

The proof is an adaptation of Imhof's proof that on finite ordered structures, DTC is contained in M-IFP (see [Imh96a]). We only sketch the idea here and note the differences between the two cases.

On any ordered structure $\mathfrak{A}$ with universe A of size n, a set $P \subseteq A$ can be seen as representing a binary string $p := p_0 \dots p_{n-1}$, where $p_i = 1$ if the i-th element of the ordering on $\mathfrak{A}$ is in P, and $p_i = 0$ otherwise. Further, every element of A has a bit representation that only uses $\log n$ bits. Thus, a tuple $\overline{a} := a_1 \dots a_k$ can be encoded by a binary string of length $k \cdot \log n$ and it can be represented by a set $P \subseteq A$, provided that $k \cdot \log n < n$. Similarly, any sequence of tuples $\overline{a}_1, \dots, \overline{a}_k$ can be represented by a set $P \subseteq A$, again provided that the representation does not use too many bits. Thus, if the tuples $\overline{a}_i$ are l-ary then a set P can represent a sequence of length $\lfloor \frac{n}{l \cdot k \cdot \log n} \rfloor$.

It has been shown by Imhof that there are formulae in M-IFP that code and decode a sequence in a relation, i.e. there are formulae that, given a tuple $\overline{a}$ define a set P representing the tuple, and given a set P representing a sequence $\overline{a}_1 \dots \overline{a}_k$ obtain $\overline{a}_k$ from it. Further, there is a formula in M-IFP that, given a set P representing $\overline{a}_1 \dots \overline{a}_k$ and given a tuple $\overline{b}$, defines the set representing the sequence $\overline{a}_1 \dots \overline{a}_k \overline{b}$. He used this to show that the deterministic path defined by a formula in DTC can be encoded by a set P, provided that the length of the path is bounded as above. As the length of a path defined by an arbitrary DTC-formula φ with two k-tuples of variables can be of length n^k, it may not be representable in one set. Therefore, Imhof used nested fixed points, so that the inner fixed point only defines fragments of the path of length $m := \lfloor \frac{n}{l \cdot k \cdot \log n} \rfloor$ and the next fixed point contains the sequence $\overline{a}_1 \dots \overline{a}_k$ of tuples, such that the distance of $\overline{a}_i$ and $\overline{a}_{i+1}$ on the path defined by φ is m. By iterating this construction he proved that φ

is equivalent to a formula in M-IFP. Further details can also be found in [Gro94] and [Gro96], where a similar construction is used to show that DTC is contained in binary LFP on ordered structures.

For TC the construction fails as there no longer is a single path that has to be computed. Instead, a TC formula $\psi := [\mathbf{tc}_{\overline{x},\overline{y}}\,\varphi(\overline{x},\overline{y})](\overline{x},\overline{y})$ computes a tree where every node corresponds to a tuple $\overline{x} \in A^k$ and has a successor for each tuple $\overline{y}$ such that $\mathfrak{A} \models \varphi[\overline{x},\overline{y}]$. The whole tree, of course, can no longer be represented by nested monadic fixed points.

In CFP, we do not have to store the whole tree in the fixed-point variables but we can non-deterministically guess a path in the tree and compute this. As the path is again polynomially bounded, it follows that it can be stored with nested fixed points as indicated above.

There is a subtle point to be made regarding the moment when the non-determinism of the fixed-point induction is resolved. We demonstrate this by two nested fixed points computing a path of length at most m^2. A first approach would be to let the outer fixed point evaluate the inner first. This done, it non-deterministically chooses a pair $(\overline{u},\overline{v})$ satisfying the inner fixed point formula, i.e. a pair connected by a path of length at most m. However, as the non-determinism in the inner fixed point is resolved immediately, i.e. the fixed point defined is the union of all strings representing a path of length m between two tuples, there would be no way to collect the right tuples from it. The fixed point might well be the whole universe as for all bit positions there might be a string representation of a path where this bit is set to 1. Thus, this approach fails.

However, if the outer fixed point first guesses a tuple $(\overline{u},\overline{v})$ and then presents it to the inner fixed point as parameter, the inner fixed point can non-deterministically check whether there is a path between the two of length at most m. Here, resolving the non-determinism is not critical, as the inner fixed point now consists of the union of all paths between $\overline{u}$ and $\overline{v}$. If the inner fixed point reports success, i.e. existence of a path between $\overline{u}$ and $\overline{v}$, in a unique way - as it was done frequently in the examples above - taking the union of all paths is not a problem. □

Combining this result with Theorem 6.8 above, we immediately get the following corollary.

6.13 Corollary. *On finite ordered structures where the order is given by a successor relation with constants for the minimal and maximal element, every formula in* TC *is equivalent to a quantifier-free formula in* M-CFP$_1$.

As noted in in Chapter 5, TC captures NLogspace on ordered structures. Further, by Corollary 6.11, CFP$_1$ captures NP. Thus, we immediately get the following result.

6.14 Corollary. *On finite ordered structures, if* M-CFP$_1$ $\neq$ CFP*, then* NLogspace $\neq$ NP.

Proof. Suppose M-CFP$_1 \neq$ CFP. Then there is a problem definable in CFP which is not definable in M-CFP$_1$. As CFP is contained in PH, there is a problem in PH which is not in NLogspace. But, as NLogspace is closed under complementation, if NLogspace = NP then also NLogspace = PH, a contradiction. □

6.2 Fixed-Point Logics with Alternating Choice

In the previous section, we considered an extension of inflationary fixed-point logic by non-deterministic choice of elements. Clearly, a similar construct can be added to PFP, resulting in a logic capturing Pspace on arbitrary structures. Further, instead of non-deterministic choice, we can also allow the alternating choice of elements. In this case, a fixed point induction is no longer a sequence of stages but a tree. The straightforward implementation of this idea leads to a logic (ACFP) that captures alternating polynomial time on arbitrary finite structures.

6.15 Theorem. ACFP *captures* APtime *and thus* Pspace *on arbitrary classes of finite structures.*

As APtime = Pspace, the extension of PFP by a choice construct and ACFP are equivalent and both capture Pspace.

Chapter 7

A General Semantics for Partial Fixed-Point Logic

In the preceding chapters we have seen a variety of fixed-point logics. All these logics can be studied on finite as well as infinite structures. The only exception to this rule is partial fixed-point logic. To see this, consider again the definition of stages of a partial fixed-point induction as in Definition 4.9.

$$\begin{aligned} R^0 &:= \varnothing \\ R^{\alpha+1} &:= F_\varphi(R^\alpha) = \{\overline{a} : (\mathfrak{A}, R^\alpha) \models \varphi[\overline{a}]\} \end{aligned}$$

Clearly, on an infinite structure $\mathfrak{A}$ the sequence of stages induced by φ may not become cyclic on the first ω stages, i.e. the fixed point of φ may not be reached after finitely many steps. What is missing is a rule for limit stages. Clearly, defining the limit stages as the union of all lower stages as it was done for IFP is not reasonable, as the sequence of stages induced by φ is not necessarily increasing (see Example 4.10).

In this chapter we aim at extending the semantics of PFP to infinite structures by giving an explicit rule for the definition of limit stages. We justify the choice of the rule by showing that a) on finite structures both definitions for PFP are equivalent and b) under the new semantics, PFP now contains IFP on all rather than just finite structures. In fact, we will even be able to show that partial fixed-point logic is strictly more expressive than inflationary fixed-point logic by exhibiting a property on infinite structures that is definable in PFP but not in IFP.

Besides the extension to infinite structures, there is another limitation of the standard PFP semantics. As noted in the introduction, there are several parameters along which fixed-point logics can be varied. One is the choice of the fixed-point construct. Another is the logic to which the fixed-point operators are affixed. All the fixed-point constructs considered so far can be attached to first-order but also to other logics like modal logic and these extensions give rise to well-defined and natural logics – again with the exception of partial fixed points. In particular, we will demonstrate below that on strings or trees, if the partial fixed point of a formula in modal logic is non-empty then the induction leading to it is actually inflationary (see Lemma 12.3). As a consequence, on trees or strings, inductions of exponential length are impossible. Clearly, this is unnatural for a partial fixed-point logic. Therefore, a second issue addressed by the new semantics is the generalisation to logical frameworks other than first-order logic.

The following example gives further evidence for the weakness of the common partial fixed-point semantics in connection with modal logic and also motivates the definition of the alternative semantics. Let MPC denote the straightforward extension of modal logic by a partial fixed-point operator. More details on modal logic and MPC can be found in Part II, in particular in Chapter 9 and 12.

Trace Equivalence and Modal Partial Fixed-Point Logic. Consider the following problem, known as the *unary trace- or language equivalence problem.* It is formally defined in Definition 12.9 below.

The input is a directed, rooted graph consisting of a number of disjoint subgraphs rooted at successors of a unique root v. The root is labelled by the proposition symbol w and is not reachable from any other node in the graph. In each of the subgraphs, some nodes are marked as final states – coloured by a colour f – whereas the other nodes are not coloured at all.

Two subgraphs rooted at successors of the root are *trace equivalent*, if, for each $n < \omega$, whenever in one of the graphs there is a path of length n from the root to a final state such a path also exists in the other graph.

We aim at showing that the class $\mathcal{C}$ of structures, such that all subgraphs rooted at successors of the root are trace equivalent, is definable in MPC. A simple idea to formalise the trace equivalence problem is as follows. Consider the formula ψ defined as

$$\psi := \mathbf{pfp}\ Z : \left\{ \begin{array}{lcl} X & \leftarrow & (f \wedge \neg Y) \vee \Diamond X \\ Y & \leftarrow & f \\ Z & \leftarrow & (w \wedge \Diamond X \wedge \Diamond \neg X) \vee Z. \end{array} \right.$$

In the first stage, X contains all final states, i.e. those labelled by f. In the successive stages, all elements are selected, which have a successor in X. Thus, the stage X^n contains precisely the elements from which there is a path of length $n-1$ to a final state. The variable Y is only used to ensure that the final states are added to X only once at the beginning, so that the induction is not started over and over again. Now, the root of the structure is included into Z if, for some n, in one subgraph there is a path of length n from its root to a final state but not in the other. Obviously, once the root is added to Z, it stays in forever. Thus, ψ is true at the root if, and only if, the subgraphs rooted at its successors are not trace equivalent.

However, if at least one of the sub-structures is cyclic, the induction on X never becomes stationary and thus, by definition, the fixed point is empty. To avoid this, we have to think about some way to ensure that the induction process becomes stationary, although the only information we are interested in, namely whether the root eventually occurs in Z, is independent of this.

This suggests a different way of defining partial fixed-point inductions. Consider the sequence of induction stages defined by ψ. Clearly, this sequence must eventually become cyclic. Now consider the set of elements that occur in all stages of this cycle and take this as the fixed point defined.[1] Applying this idea to the example above, we get that the fixed point of X becomes empty (unless there are self loops), the fixed point of Y contains all final states, and the fixed point of Z contains the root just in case there are two successors of it which are not trace equivalent. Thus, the formula $\neg\psi$ is true in $\mathcal{K}, v$ if, and only if, $\mathcal{K}, v \in \mathcal{C}$. This motivates an alternative semantics for partial fixed-point logic based on this idea.

7.1 An Alternative Semantics for Partial Fixed-Point Logic

We now give a formal definition of the general semantics for partial fixed-point logic.

7.1 Definition (General Semantics). *Let $\psi := [\mathbf{pfp}_{R,\overline{x}}\,\varphi](\overline{t})$ be a formula in* PFP *and let $\mathfrak{A}$ be a structure with universe A providing an interpretation of all free variables of φ that are not in $\overline{x}$. Consider the following sequence of stages induced by φ on $\mathfrak{A}$.*

$$\begin{aligned} R^0 &:= \varnothing \\ R^{\alpha+1} &:= F_\varphi(R^\alpha) = \{\overline{a} : (\mathfrak{A}, R^\alpha) \models \varphi[\overline{a}]\} \\ R^\lambda &:= \mathrm{final}((R^\alpha)_{\alpha<\lambda}) \qquad \textit{for limit ordinals } \lambda, \end{aligned}$$

where $\mathrm{final}((R^\alpha)_{\alpha<\lambda})$ *is defined as the set of tuples $\overline{a}$ such that there is some $\beta < \lambda$ and $\overline{a} \in R^\gamma$ for all $\beta < \gamma < \lambda$.*

Obviously, the sequence $(R^\alpha)_{\alpha\in\mathrm{Ord}}$ must eventually become cyclic. Let β_2 be minimal such that $R^{\beta_1} = R^{\beta_2}$ for some $\beta_1 < \beta_2$. Then, for any tuple $\overline{a} \in A$,

$$\mathfrak{A} \models [\mathbf{pfp}_{R,\overline{x}}\,\varphi](\overline{a}) \textit{ if, and only if, } \overline{a} \in R^\gamma \textit{ for all } \beta_1 \le \gamma < \beta_2.$$

[1]Note that this set is not necessarily a fixed point of the operator induced by the formula. Nevertheless, we use this terminology to keep consistent with the other fixed-point logics.

We also allow simultaneous inductions and again the same proof as for Theorem 3.33 shows that this does not increase the expressive power.

7.2 Theorem. *Under the general semantics, every formula in* PFP *with simultaneous inductions is equivalent to a formula without simultaneous inductions.*

According to the definition, the fixed point of a formula φ is defined as the set of elements which occur in every stage of the first cycle in the sequence of stages induced by φ. Note that this is not equivalent to saying that the fixed point consists of those elements $\overline{a}$ such that there is a stage β and $\overline{a}$ occurs in all stages greater than β. For example, consider the structure $\mathfrak{A} := (\{0,1,2,3\})$ and a formula defining an operator which takes $\varnothing \mapsto \{0,1\}$, $\{0,1\} \mapsto \{0,2\}$ and $\{0,2\} \mapsto \{0,1\}$. Further, it takes $\{0\} \mapsto \{2\}$ and $\{2\}$ to itself. The operator induces the following induction stages R^α: For all $0 < n < \omega$, $R^n = \{0,1\}$ if n is odd and $R^n = \{0,2\}$ if n is even. Thus, the partial fixed point according to Definition 7.1 is $\{0\}$. However, $R^\omega = \{0\}$ and for all $\alpha > \omega$, $R^\alpha = \{2\}$. Therefore, defining the fixed point as the set of elements which are contained in all stages greater than some β yields a different set than the partial fixed point as defined above. We could also have used this definition as the basis to define a semantics for PFP. But, this would imply that there are structures with four elements and formulae whose fixed point on these structures is reached only after infinitely many stages – a concept that is not very natural.

The semantics given in Definition 7.1 extends the common semantics in two ways, namely by introducing a rule for the limit stages of the fixed-point induction and by giving a different meaning to what the result obtained from the induction actually is. These two extensions are somewhat independent. In particular, we could just have considered the rule for the limit stages and define the fixed point in the usual way by saying that whenever the transfinite induction reaches a fixed point, we take this as the result of the induction and otherwise define it to be empty.

Conversely, we could have focused on the alternative way to define the fixed point and forget about the extension to transfinite inductions. It is easily seen that the proofs of Lemma 7.3 and Theorem 7.4, where we show that PFP with either of the two semantics is equivalent on finite structures, extend to transfinite inductions. Thus, whether we extend PFP by the rule for transfinite inductions and keep the old fixed-point definition or use the new fixed-point rule is irrelevant for the expressive power of the resulting logic. However, the new fixed-point rule also allows extensions of logics like modal logic and is therefore more general than the old definition.

We prove next, that in the restriction to finite structures, PFP with the semantics in Definition 4.9 and PFP with the semantics in Definition 7.1 are equivalent.

Notation. To distinguish between the two semantics, we denote PFP under the finite model semantics as PFP_{fin} and write the operator as $\mathbf{pfp}^f$. We write PFP_{gen} and $\mathbf{pfp}^g$ whenever we speak about the general semantics. Further, if φ is any formula in PFP, we write $\text{fin}(\varphi)$ to denote the formula under the finite model semantics and $\text{gen}(\varphi)$ for the general semantics. $\square$

We first prove a technical lemma that establishes the main step in the proof of the theorem below.

7.3 Lemma. *Let $\mathfrak{A}$ be a structure. For every formula $\varphi(R, \overline{x})$ in* PFP_{gen} *there is a formula* $\text{fixed-point}_\varphi(R, \overline{x})$ *such that for any stage R^α of the induction of φ on $\mathfrak{A}$ and all $\overline{a} \in A$, $(\mathfrak{A}, R^\alpha) \models \text{fixed-point}_\varphi[\overline{a}]$ if, and only if, there is a minimal ordinal γ such that there is some $\beta < \gamma$ with $R^\alpha = R^\gamma = R^\beta$ and $\overline{a} \in \bigcap_{\beta<\xi\leq\gamma} R^\xi$.*

Further, if $\mathfrak{A}$ is finite and φ is a formula in PFP_{fin}*, then fin*$(\text{fixed-point}_\varphi)$ *and gen*$(\text{fixed-point}_\varphi)$ *are equivalent.*

Proof. Without loss of generality we assume that for no $\alpha > 0$ the stage R^α of the induction on φ becomes empty. For, every formula $[\mathbf{pfp}_{R,\overline{x}}\ \varphi](\overline{x})$ is equivalent to $\exists 0 \exists 1\, 0 \neq 1 \wedge [\mathbf{pfp}_{R',\overline{x},t}\ (t = 0 \wedge \varphi) \vee (t = 1)](\overline{x}, 0)$.

Consider the formula

$$\textit{fixed-point}_\varphi(R, \overline{x}) := \textit{is-cycle} \wedge \textit{min-cycle} \wedge \textit{elem-on-cycle}(\overline{x})$$

where the sub-formulae *is-cycle*, *min-cycle*, and *elem-on-cycle*$(\overline{x})$ are defined as follows. The formula *is-cycle*$(R) := [\mathbf{pfp}\ Y : S](\overline{x})$, where

$$S := \begin{cases} X\overline{x} & \leftarrow \quad (X = \varnothing \wedge \varphi(R, \overline{x})) \vee (X \neq \varnothing \wedge X \neq R \wedge \varphi(X, \overline{x})) \vee \\ & \qquad (X \neq \varnothing \wedge X = R \wedge R\overline{x}) \\ Y & \leftarrow \quad Y \vee (Y = \varnothing \wedge X = R) \end{cases}$$

is true for an interpretation R^α of the relation variable R if there is some $\gamma > \alpha$ such that $R^\alpha = R^\gamma$, i.e. if R^α is contained in a cycle. For this, consider the induction stages induced by the system S. The relation X is initialised to the stage $R^{\alpha+1}$ and then runs through the stages R^β for $\beta > \alpha$. If it reaches a stage γ with $R^\gamma = R^\alpha$, the induction becomes stationary and for all $\xi > \gamma$, $R^\xi = R^\alpha$. In this case, the 0-ary relation Y becomes true. As the rule for Y is inflationary, it stays true forever. Thus the fixed point of S is reached and the formula *is-cycle*(R) becomes true. On the other hand, if there is no stage $\gamma > \alpha$ with $R^\gamma = R^\alpha$ then Y will never become true and *is-cycle*(R) evaluates to false.

Suppose *is-cycle* becomes true for a stage $R := R^\alpha$ and suppose that α is the least such stage. As shown above, this implies that R is part of a cycle, i.e. there is some stage $\gamma > \alpha$ with $R^\gamma = R^\alpha$. However, this does not rule out the possibility that there are stages β_1, β_2 with $\alpha < \beta_1 < \beta_2 < \gamma$ such that $R^{\beta_1} = R_2^\beta$ but $R^\alpha \neq R^{\beta_1}$. In this case, the fixed point of the induction on φ will not consist of the elements on the cycle between α and γ but of the elements on another cycle contained in it. Now consider the formula *min-cycle* $:= \neg[\mathbf{pfp}\ Y : T]$, with

$$T := \begin{cases} X & \leftarrow & (X = \varnothing \wedge \varphi(R, \overline{x})) \vee (X \neq \varnothing \wedge X \neq R \wedge \varphi(X, \overline{x})) \vee \\ & & (X \neq \varnothing \wedge X = R \wedge X\overline{x}) \\ Y & \leftarrow & Y \vee (X \neq R \wedge X \neq \varnothing \wedge [\mathbf{pfp}\ Z' : T']) \end{cases}$$

and

$$T' := \begin{cases} Z & \leftarrow & (Z = \varnothing \wedge \varphi(X, \overline{x})) \vee (Z \neq \varnothing \wedge \varphi(Z, \overline{x}) \wedge Z \neq X) \vee \\ & & (Z = R \wedge Z\overline{x}) \\ Z' & \leftarrow & Z = X \end{cases}$$

Suppose R is interpreted by a stage R^α such that there is some $\gamma > \alpha$ with $R^\gamma = R^\alpha$. We claim that *min-cycle* becomes true for this interpretation of R if there are no β_1, β_2 such that $\alpha < \beta_1 < \beta_2 < \gamma$. For this, consider the induction stages of the system T. The rule for X is the same as in the system S. Thus, X is initialised to $R^{\alpha+1}$, then runs through the stages R^γ with $\gamma > \alpha$. If $R^\gamma = R^\alpha$ for some γ then the induction becomes stationary and $X^\infty = R^\gamma$. Now consider the induction on the 0-ary relation Y. Y becomes true if at some stage ξ such that $X^\xi \neq \varnothing$ and $X^\xi \neq R^\alpha$ the formula $[\mathbf{pfp}\ Z' : T]$ becomes true. In the induction on T', Z is initialised to $X^{\xi+1}$ and then runs trough all stages $X^\beta, \beta > \xi$. As soon as Z equals R, the induction becomes stationary. Now, the relation Z' becomes true if at some stage $Z = X$. In this case, there are β_1, β_2 with $\alpha < \beta_1 < \beta_2 < \gamma$ such that $R^{\beta_1} = R^{\beta_2}$. Clearly, if there is no such cycle, then Z' stays false.

Thus, the variable Y becomes true in the induction on T if, and only if, there is a cycle between R^{β_1} and R^{β_2}. This proves the claim.

Finally, we define *elem-on-cycle*$(\overline{x}) := [\mathbf{pfp}\ Y : U](\overline{x})$ with

$$U := \begin{cases} X\overline{x} & \leftarrow & (X = \varnothing \wedge \varphi(R, \overline{x})) \vee (X \neq \varnothing \wedge X \neq R \wedge \varphi(X, \overline{x})) \vee \\ & & (X = R \wedge R\overline{x}) \vee \\ Y\overline{x} & \leftarrow & (Y = \varnothing \wedge R\overline{x}) \vee (Y \neq \varnothing \wedge Y\overline{x} \wedge X\overline{x}) \end{cases}$$

Suppose R is interpreted by a stage α such that *is-cycle* and *min-cycle* hold true for R^α. Let $\gamma > \alpha$ be the least ordinal greater than α such that $R^\gamma = R^\alpha$. We claim that *elem-on-cycle* becomes true for a tuple $\overline{a}$ if, and only if, $\overline{a} \in R^\xi$ for all $\alpha \leq \xi \leq \gamma$. Consider the induction stages induced by U. Again, the rule for X is as before, i.e. X is initialised to R^α, runs through all stages R^ξ with $\alpha < \xi < \gamma$ and becomes stationary once it has reached R^γ.

Now consider the induction on Y. At the beginning, Y is initialised to R^α. At each stage X goes through, the intersection between X and Y defines the next stage for Y. Thus at the end, Y contains all elements which are contained in any stage R^ξ with $\alpha < \xi < \gamma$.

This shows that if R is interpreted by R^γ for the least γ such that there is some $\alpha < \gamma$ with $R^\alpha = R^\gamma$, then the formula $\textit{fixed-point}_\varphi(R, \overline{x})$ defines all elements contained in any stage of the cycle. In all other cases, $\textit{fixed-point}_\varphi$ defines the empty set.

What is left to be shown is that on finite structures $\mathfrak{A}$, $\text{fin}(\textit{fixed-point}_\varphi)$ and $\textit{gen}(\textit{fixed-point}_\varphi)$ are equivalent. This is clear if R is interpreted by R^α as above, as then all inductions become stationary. If R is interpreted by some stage R^α such for no $\gamma > \alpha$, $R^\gamma = R^\alpha$, then the formula *is-cycle* becomes false under both semantics and both, $\text{fin}(\textit{fixed-point}_\varphi)$ and $\textit{gen}(\textit{fixed-point}_\varphi)$ define the empty set. □

We are now ready to prove the equivalence of the two partial fixed-point semantics defined above.

7.4 Theorem. *On finite structures,* PFP_{fin} *and* PFP_{gen} *are equivalent, i.e. every formula under the finite model semantics is equivalent to a formula under the general semantics and vice versa.*

Proof. The forth direction follows easily by induction on the structure of the formula. In the main step, let $\psi := [\mathbf{pfp}^f_{R,\overline{x}}\, \varphi(R, \overline{x})](\overline{t})$ be a formula in PFP_{fin}. It is equivalent to

$$\psi^g := [\mathbf{pfp}^g\ Q : \begin{array}{lcl} R\overline{x} & \leftarrow & \varphi^g(R, \overline{x}) \\ Q\overline{x} & \leftarrow & \forall \overline{x}(\varphi^g(R, \overline{x}) \leftrightarrow R\overline{x}) \wedge R\overline{x} \end{array}](\overline{t}),$$

where φ^g is a PFP_{gen}-formula equivalent to φ. By induction, such a formula always exists. Assume first that on a structure $\mathfrak{A}$ a fixed point of φ is reached at some stage α. In this case, $\forall \overline{x}(\varphi(R, \overline{x}) \leftrightarrow R\overline{x})$ becomes true at stage α and $Q^{\alpha+1} = R^\alpha$. Thus, the fixed point of Q is R^∞ and ψ and ψ^g are equivalent.

Now assume that the fixed point of φ does not exist. Then, at no stage, $\forall \overline{x}(\varphi^g(R, \overline{x}) \leftrightarrow R\overline{x})$ becomes true and ψ^g defines the empty set.

The other direction is also proved by induction on the structure of the formulae. In the main step, assume that $\psi := [\mathbf{pfp}^g_{R,\overline{x}}\, \varphi(R, \overline{x})](\overline{t})$ is a formula under the general semantics. By induction, φ is equivalent to a formula φ^f in PFP_{fin}. Then, ψ is equivalent to

$$\psi^f := [\mathbf{pfp}^f\ Q : \begin{array}{lcl} R\overline{x} & \leftarrow & (Q = \varnothing \wedge \varphi^f(R, \overline{x})) \\ Q\overline{x} & \leftarrow & Q\overline{x} \vee (Q = \varnothing \wedge \textit{fixed-point}_{(\varphi^f)}(R, \overline{x})) \end{array}](\overline{t}).$$

By Lemma 7.3, the formula $\textit{fixed-point}_{(\varphi^f)}(R)$ can be chosen from PFP_{fin}. Thus, as $\varphi^f \in \text{PFP}_{\text{fin}}$, we get that ψ^f is itself a formula in PFP_{fin}. To see that ψ and ψ^f are equivalent consider the sequence of stages induced by ψ^f. As long as Q is empty, R runs through all stages induced by φ^f. Now let β be the least ordinal such that there is some $\alpha > \beta$ with $R^\alpha = R^\beta$. Then, Lemma 7.3 implies that $\textit{fixed-point}_{(\varphi^f)}$ defines the set of tuples contained in all stages of the cycle. Thus, in the next stage, $Q^{\beta+1}$ contains the fixed point of φ. If this is not empty, then R becomes empty at the next stage, Q does not change anymore, and therefore the fixed point has been reached.

Should the fixed point of φ be empty, then Q remains empty. However, in this case, there is no stage γ such that $R^\gamma = R^{\gamma+1}$ and therefore ψ^f defines the empty set. □

The theorem allows us to transfer the results on $\mathrm{PFP}_{\mathrm{fin}}$ mentioned in Section 4.2 and Chapter 5 to $\mathrm{PFP}_{\mathrm{gen}}$, in particular the theorems by Abiteboul, Vianu, Immerman, and Vardi.

7.5 Corollary.

(*i*) $\mathrm{PFP}_{\mathrm{gen}}$ *captures* PSPACE *on ordered structures.*

(*ii*) $\mathrm{PFP}_{\mathrm{gen}}$ *and* IFP *are equivalent on finite structures if, and only if,* PTIME = PSPACE.

(*iii*) *Every* $\mathrm{PFP}_{\mathrm{gen}}$ *formula is equivalent on finite structures to a formula with only one application of a fixed-point operator.*

Proof. The corollary follows immediately from the fact that every formula in $\mathrm{PFP}_{\mathrm{fin}}$ is equivalent to a formula with only one fixed-point operator and that the translation to $\mathrm{PFP}_{\mathrm{gen}}$-formulae as presented in the proof of Theorem 7.4 does not increase the number of fixed-point operators. □

Using a diagonalisation argument as in Section 7.2 below, it is clear that for any fixed-point logic like LFP, IFP, or PFP, the nesting-depth hierarchy must in general be strict, i.e. allowing the nesting of fixed-point operators does strictly increase the expressive power. Thus, Part (*iii*) of the preceding corollary fails on infinite structures. We close the section by establishing a negation normal form for $\mathrm{PFP}_{\mathrm{gen}}$-formulae, i.e. the alternation between fixed points and negation does not provide more expressive power than just nesting fixed-points.

7.6 Theorem. *Every* $\mathrm{PFP}_{\mathrm{gen}}$ *formula is equivalent to one where negation occurs only in front of atoms.*

Proof. Let $\psi(\overline{t}) := \neg[\mathbf{pfp}_{R,\overline{x}}\varphi(R,\overline{x})](\overline{t})$ be a formula in PFP. Obviously, it is equivalent to the formula

$$\psi'(\overline{t}) := \exists 0 \neq 1 \ [\mathbf{pfp}\ Q : \begin{array}{lcl} P\overline{x}y & \leftarrow & y = 1 \vee (y = 0 \wedge [\mathbf{pfp}_{R,\overline{x}}\varphi](\overline{x})) \\ Q\overline{x} & \leftarrow & P \neq \varnothing \wedge \neg P\overline{x}0 \end{array}](\overline{t}),$$

where $0, 1$ are variables not occurring in φ. The theorem now follows immediately by induction on the structure of the formulae. □

7.2 Separating Partial and Inflationary Fixed-Point Logic

We now prove the main result of this chapter, the separation of $\mathrm{PFP}_{\mathrm{gen}}$ and IFP. As we are not considering the finite model semantics anymore, we simply write PFP and **pfp** instead of $\mathrm{PFP}_{\mathrm{gen}}$ and $\mathbf{pfp}^g$.

Clearly, every formula in IFP is equivalent to a formula in PFP. To demonstrate a property that is definable in PFP but not in IFP we show that the truth predicate for IFP-formulae – which clearly is not definable in IFP – can be defined in PFP on some classes of infinite structures.

7.2.1 Acceptable Structures, Coding, and Diagonalisation

First, we present a class of structures called *acceptable* (see [Mos74a, Chapter 5]). These structures are particularly well suited to be used with diagonalisation arguments.

7.7 Definition. *Let A be an infinite set. A* coding scheme on A *is a triple $(\mathcal{N}, \leq, <>)$, with $\mathcal{N} \subseteq A$, such that the structure $(\mathcal{N}, \leq)$ is isomorphic to $(\omega, \leq)$ and $<>$ is an injective map from $\bigcup_{n<\omega} A^n$ into A. The image a of $a_1, \dots, a_n$ under $<>$ is called the* code of $a_1, \dots, a_n$.

With each coding scheme we associate the following decoding relations:

(*i*) *$seq(x)$, which is true for x if, and only if, $x = \langle x_1, \dots, x_n\rangle$ is the code of some sequence $x_1, \dots, x_n$.*

(*ii*) *$lh(x) = n$, if x is the code of a sequence of length n.*

(*iii*) *$q(x, i) = x_i$, if $x = \langle x_1, \dots, x_n\rangle$ and $n \geq i$. We write $(x)_i = a$ for $seq(x) \wedge q(x, i) = a$.*

The numbers $0, 1, \dots$ refer to the corresponding elements in $\mathcal{N}$. For convenience, we write lh and q functional. But note that they are relations. In particular, $lh(x)$ and $q(x, i)$ are undefined if x does not code a sequence.

An elementary coding scheme $\mathcal{C}$ on a structure $\mathfrak{A}$ *is a coding scheme on its universe where the relations $\mathcal{N}, \leq$, seq, lh, and q are elementary, i.e., first-order definable.*

A structure $\mathfrak{A}$ admitting an elementary coding scheme is called acceptable. *We call $\mathfrak{A}$* quasi-acceptable *if there exists an acceptable expansion $\mathfrak{A}'$ of $\mathfrak{A}$ by a finite set of* PFP*-definable relations.*

Observe that quasi-acceptable structures are those which admit an PFP-definable coding scheme, i.e. one where the relations $<$, seq, lh, and q are PFP-definable. See [Mos74a, Chapter 5] for more on elementary and inductive coding schemes.

We now present a way of encoding formulae by elements of acceptable structures. Let $\tau := \tau_{\mathrm{rel}} \dot{\cup} \tau_{\mathrm{const}}$ be the disjoint union of a finite set $\tau_{\mathrm{rel}} :=$

$\{P_1, \dots, P_l\}$ of relation symbols and a finite set $\tau_{\text{const}} := \{c_1, \dots, c_m\}$ of constant symbols. Fix an acceptable τ-structure $\mathfrak{A}$.

W.l.o.g. we agree on the following conventions about the structure of IFP-formulae φ.

- No fixed-point variable is bound twice in the same formula.
- The involved fixed-point variables R_i are numbered from 1 to the number k of fixed-point operators occurring in the formula. Let φ_i and φ_j be the formulae defining the fixed point inductions on R_i and R_j respectively. Then, for no $i < j \leq k$, φ_i is a sub-formula of φ_j.
- All formulae are of the form $[\mathbf{ifp}_{R_1,\overline{x}_1}\, \varphi_1](\overline{x}_1)$.
- We only allow $\vee$ and $\exists$ in the formulae whereas $\wedge$ and $\forall$ are forbidden.
- All fixed-point operators are of the form $[\mathbf{ifp}_{R,\overline{x}}\, R\overline{x} \vee \varphi(R, \overline{x})]$, i.e. the operators are syntactically made inflationary.
- Finally, we assume that in formulae $[\mathbf{ifp}_{R,x_{i_1},\dots,x_{i_k}}\, \varphi]$, atoms involving the fixed-point variable R may only be used in the form $Rx_{i_1}, \dots, x_{i_k}$.

Clearly, any IFP-formula can be brought into this form. The actual encoding of formulae is based on a function $||\varphi||$ taking formulae or terms in IFP$[\tau]$ to elements of $\mathcal{N}$. Let $\mathbf{c}, \mathbf{var}, \dots$ denote arbitrary but fixed and distinct elements of $\mathcal{N}$. The function is inductively defined as follows.

$$\begin{aligned}
||c_i|| &:= \langle \mathbf{c}, i \rangle && c_i \in \tau_{\text{const}}\\
||x_i|| &:= \langle \mathbf{var}, i \rangle\\
||P_i\overline{a}|| &:= \langle \mathbf{rel}, i, \langle ||\overline{a}|| \rangle \rangle && P_i \in \tau_{\text{Rel}}\\
||\varphi_1 \vee \varphi_2|| &:= \langle \mathbf{or}, ||\varphi_1||, ||\varphi_2|| \rangle\\
||\neg\varphi|| &:= \langle \mathbf{neg}, ||\varphi|| \rangle\\
||\exists x_i \varphi|| &:= \langle \mathbf{exists}, i, ||\varphi|| \rangle\\
||R_i\overline{a}|| &:= \langle \mathbf{fp\text{-}var}, i, \langle ||\overline{a}|| \rangle \rangle && \text{for fixed-point variables } R_i\\
||\,[\mathbf{ifp}_{R_i,\overline{x}}\, \varphi](\overline{a})|| &:= \langle \mathbf{fp\text{-}op}, i, \langle ||\overline{a}|| \rangle \rangle
\end{aligned}$$

Here $\langle ||\overline{a}|| \rangle$ is an abbreviation for $\langle ||a_1||, \dots, ||a_k|| \rangle$ where k is the arity of $\overline{a}$. In this encoding of formulae, sub-formulae involving fixed-point variables are only coded by the number of the involved fixed-point variable but no code of the formula defining it is stored. The next definition deals with this.

7.8 Definition. *Let φ be a formula in* IFP$[\tau]$ *and let the fixed-point operators occurring in it be* $[\mathbf{ifp}_{R_1,\overline{x}_1}\, \varphi_1]$, ..., $[\mathbf{ifp}_{R_k,\overline{x}_k}\, \varphi_k]$. *The φ_i are called the* defining formulae of φ *and each individual φ_i is called the* defining formula of the fixed-point variable R_i.

The function code taking formulae to their codes in $\mathcal{N}$ is defined as

$$\begin{aligned}
\mathit{code}:\ \text{IFP}[\tau] &\longrightarrow \mathcal{N}\\
\varphi &\longmapsto \langle ||\varphi_1||, \dots, ||\varphi_k|| \rangle,
\end{aligned}$$

where $\varphi_1, \dots, \varphi_k$ are the defining formulae of φ.

Below, we will use encodings of formulae to show that there are relations on acceptable structures which are PFP but not IFP-definable. We first fix some notation that will be used in the sequel.

7.9 Definition. *Let $\varphi(\overline{x})$ be a formula with free variables $\overline{x} := x_{i_1}, \dots, x_{i_k}$ for some k. The* code a of a sequence matches φ, *if $lh(a) \geq i_j$ for all $1 \leq j \leq k$.*

We write $a \models \varphi$, if a matches φ and φ is true in $\mathfrak{A}$ under the variable assignment

$$\beta : x_i \longmapsto \begin{cases} (a)_i & \textit{for all } 1 \leq i \leq \mathrm{lh}(a) \\ 0 & \textit{otherwise.} \end{cases}$$

If c is the code of φ we also write $a \models c$ for $a \models \varphi$.

We state the following lemma whose proof is technical but not very difficult.

7.10 Lemma. *There is a* PFP*-formula* formula(x) *that is true for all c which are valid codes of* IFP*-formulae.*

7.2.2 Separating Partial and Inflationary Fixed-Point Logic

In this section, we show that partial fixed-point logic is strictly more expressive than inflationary fixed-point logic. Fix an acceptable structure $\mathfrak{A}$ with universe A.

7.11 Definition. *The relation* $\textsc{Sat}_{\mathrm{IFP}} \subseteq A^2$ *is defined as*

$$\textsc{Sat}_{\mathrm{IFP}} := \{(c, a) : c \textit{ is the code of an } \mathrm{IFP}[\tau]\textit{-formula } \varphi \textit{ and } a \models \varphi\}.$$

A simple diagonalisation argument establishes the next lemma.

7.12 Lemma. $\textsc{Sat}_{\mathrm{IFP}}$ *is not definable in* IFP.

Proof. Suppose, $\textsc{Sat}_{\mathrm{IFP}}$ was definable in IFP. Then the relation $R(x) := \neg\textsc{Sat}_{\mathrm{IFP}}(x, \langle x \rangle)$ would also be definable in IFP, by a formula $\varphi(x)$ say. Let c be the code of φ. Thus, as φ defines R, for all x, $R(x) \iff \textsc{Sat}_{\mathrm{IFP}}(c, \langle x \rangle)$ but, by definition of R, for all x, $R(x) \iff \neg\textsc{Sat}_{\mathrm{IFP}}(x, \langle x \rangle)$. For $x = c$ we get a contradiction. □

We show now that $\textsc{Sat}_{\mathrm{IFP}}$ is definable in PFP. For this, we inductively build up a ternary relation $R(c, i, a) \subseteq A^3$ such that $(c, i, a) \in R$ if, and only if, c is the code of a formula $\varphi \in \mathrm{IFP}[\tau]$ with defining formulae $\varphi_1, \dots, \varphi_k$, i is an element of $\{1, \dots, k\}$, and a is the code of a variable assignment matching the free variables in φ_i such that

$$(\mathfrak{A}, \mathit{stage}(c, 1), \dots, \mathit{stage}(c, k)), a \models \varphi_i,$$

i.e. φ_i is true under the variable assignment a where all the fixed-point relations R_j are interpreted by the sets $\mathit{stage}(c,j)$ defined as

$$\mathit{stage}(c,j) := \{\overline{a} : \overline{a} \text{ is encoded by } a \text{ and } (c,j,a) \in R\}.$$

This relation will be built up by a partial fixed-point induction such that the following invariance property is preserved.

7.13 Invariance Property.

- *For all c, i, a, if $(c,i,a) \in R$ then c is the code of a formula $\varphi \in \mathrm{IFP}[\tau]$, with defining formulae $\varphi_1, \dots, \varphi_k$, i is an element of $\{1, \dots, k\}$, and a is the code of a variable assignment matching the free variables in φ such that*

$$(\mathfrak{A}, \mathrm{stage}(c,1), \dots, \mathrm{stage}(c,k)), a \models \varphi_i,$$

  i.e. φ_i is true under the variable assignment a where all free fixed-point relations R_j are interpreted by the sets $\mathrm{stage}(c,j)$*.*

- *Let α be a stage of the induction on R and let i and c be as above. Consider the induction on R_i induced by $\varphi_i(\overline{x}, R_i, R_1, \dots, R_{i-1})$, where for $1 \le j < i$, the variables R_j are interpreted by* $\mathrm{stage}(c,j)$*. Then there is a stage R_i^β such that $R_i^\beta = \mathrm{stage}(c,i)$, i.e.*

$$R_i^\beta = \{\overline{a} : \ \overline{a} \textit{ is encoded by } a \textit{ and } (c,i,a) \in R^\alpha\}.$$

Before presenting a formula defining R, we need two auxiliary formulae *first-order* and *fpr*. The formula *first-order*(R,c,i,a) assumes that the invariance property in 7.13 is satisfied by R. In this case, it defines the set of all (c,i,a) such that $a \models \varphi_i$, under the assumption that all free fixed-point variables R_j of φ_i are interpreted by $\mathit{stage}(c,j)$ and for all sub-formulae of φ_i of the form $[\mathbf{ifp}_{R_l,\overline{x}_l}\, \varphi_l]$ the fixed point defined by this formula is $\mathit{stage}(c,l)$. Obviously, these assumptions are too optimistic for all i, as the second assumption will generally be true only for some, but not for all i. However, *first-order* will be used in a formula defining the relation R described above and there it will be guaranteed that it is only used for values of i for which both assumptions are satisfied.

In the following, we treat variables $t, t_1, \dots$ as boolean variables, i.e. the only values they can take are 0 and 1, and we use expressions like $t = t_1 \lor t_2$ with the obvious semantics. We also use notation like "$c \widehat{=} \varphi_{c_1} \lor \varphi_{c_2}$" meaning that c is the code of a formula $\varphi := \varphi_1 \lor \varphi_2$ and c_1, c_2 are the codes of the sub-formulae.

$$
\begin{aligned}
&\textit{first-order}(c,i,a) :=\\
&\quad [\mathbf{pfp}_{Q,c,a,t}\ \text{“}c\widehat{=}\exists x_j\varphi_{c'}\text{”} \wedge ((\exists a'\, Qc'a'1 \wedge \forall i \neq j\, (a)_i = (a')_i \wedge t = 1) \vee\\
&\qquad\qquad\qquad\qquad\qquad (\forall a'\, (\forall i \neq j\, (a)_i = (a')_i) \rightarrow Qc'a'0) \wedge t = 0)) \vee\\
&\qquad\qquad \text{“}c\widehat{=}\varphi_{c_1} \vee \varphi_{c_2}\text{”} \wedge (\exists t_1 \exists t_2 (Qc_1at_1 \wedge Qc_2at_2 \wedge t = t_1 \vee t_2) \vee\\
&\qquad\qquad \text{“}c\widehat{=}\neg\varphi_{c'}\text{”} \wedge (\exists t'\, Qc'at' \wedge t = \neg t') \vee\\
&\qquad\qquad \text{“}c\widehat{=}P_i x_{i_1} \dots x_{i_k}\text{”} \wedge (t \leftrightarrow P_i(a)_{i_1} \dots (a)_{i_k}) \vee\\
&\qquad\qquad \text{“}c\widehat{=}R_i\overline{x}\text{”} \wedge (t \leftrightarrow Rcia) \vee\\
&\qquad\qquad \text{“}c\widehat{=}[\mathbf{ifp}_{R_i,\overline{x}}\,\varphi_i]\text{”} \wedge (t \leftrightarrow Rcia)\\
&\quad\ ]((c)_i, a, 1)
\end{aligned}
$$

To ease notation, the cases where constants occur in the atomic formulae have not been made explicit. Constants can be treated in the same way as variables. The correctness of the construction is proved in the following lemma.

7.14 Lemma. *Let R be a ternary relation satisfying the invariance property in 7.13. Then, for all c, i, a, such that c is the code of a formula φ with defining sub-formulae $\varphi_1, \dots, \varphi_k$ and $i \in \{1, \dots, k\}$,*

$$(\mathfrak{A}, R) \models \text{first-order}(c,i,a) \quad \textit{if, and only if,} \quad a \models \varphi_i,$$

where all free fixed-point variables R_j and all sub-formulae of the form $[\mathbf{ifp}_{R_j,\overline{x}_j}\varphi_j]$ are interpreted by the sets stage(R, j).

Proof. The lemma is proved by induction on the structure of φ. As the argument is fairly standard, we do not present the full proof here but refer to [Mos74a, Chapter 5] for details. In its induction stages, the formula builds up a ternary relation Q such that if $(c, a, t) \in Q$ then $t = 1$ and $c \models a$ or $t = 0$ and $c \not\models a$. We demonstrate the idea by proving the case for existential quantification.

Suppose c is the code of a formula $\varphi := \exists x_j \varphi_{c'}$ and c' is the code of $\varphi_{c'}$. There are three cases to be considered. In the first two cases, the truth value for the sub-formula $\varphi_{c'}$ has already been computed.

Suppose, $\exists x_j \varphi_{c'}$ is true. Then there is (the code a' of) a variable assignment that agrees with a on all positions except j such that $a' \models \varphi_{c'}$. In this case, $(\exists a'\, Qc'a'1 \wedge \forall i \neq j\, (a)_i = (a')_i)$ is satisfied and the triple $(c, a, 1)$ is added to Q.

Now suppose $\exists x_j \varphi_{c'}$ is not satisfied. Then $a' \not\models \varphi_{c'}$ for all (codes a' of) truth assignments which agree with a on all positions except j. Thus, for all such a' the triple $(c, a', 0)$ is contained in Q and therefore $\forall a'\, (\forall i \neq j\, (a)_i = (a')_i) \rightarrow Qc'a'0$ is satisfied and the triple $(c, a, 0)$ is added to Q.

Finally, in the third case, the truth value for $\varphi_{c'}$ has not yet been determined and therefore nothing is added to Q for c.

Note how the truth of sub-formulae involving fixed point relations is directly taken from the relation R. □

We also need a formula $\mathit{fpr}(R, c, i)$ that is true for c and i if $\mathit{stage}(c, i)$ is the fixed point of the induction on φ_i where all free fixed-point variables R_j of φ_i are interpreted by $\mathit{stage}(c, j)$.

$$\mathit{fpr}(R, c, i) := \forall a(\mathit{first\text{-}order}(R, c, i, a) \rightarrow R(c, i, a)).$$

Clearly, under the same assumptions as in Lemma 7.14, $(\mathfrak{A}, R) \models \mathit{fpr}(c, i)$ if, and only if, $\mathit{stage}(c, i)$ is the fixed-point of φ_i. We are now ready to define the main formula.

$$\begin{aligned}
\mathit{compute}(c, a) := {} & \mathit{formula}(c) \wedge \\
& [\mathbf{pfp}_{R,c,i,a} \; (\exists l\, 1 \leq l \leq \mathrm{lh}(c) \wedge \neg \mathit{fpr}(R, c, l) \wedge \\
& \qquad \forall j\, (l < j \leq \mathrm{lh}(c) \rightarrow \mathit{fpr}(R, c, j)) \wedge \\
& \qquad ((i = l \wedge \mathit{first\text{-}order}(c, i, a)) \vee (i < l \wedge \mathit{Rcia}))) \vee \\
& \qquad (\forall l \in \{1, \ldots, \mathrm{lh}(c)\}\; \mathit{fpr}(R, c, j)) \wedge \mathit{Rcia} \\
&](c, 1, a).
\end{aligned}$$

As discussed in Lemma 7.10, $\mathit{formula}(c)$ is true for all codes c of formulae in IFP. Recall the way a formula φ is coded by $c := \langle ||\varphi_1||, \ldots, ||\varphi_k|| \rangle$. The formula *compute* first defines the unique l such that the fixed points of all φ_j with $j > l$ are already computed in R but the induction on φ_l has not yet reached its fixed point. For this l, $\mathit{first\text{-}order}(c, l, a)$ is evaluated, i.e. the next stage of the induction on φ_j is computed. Further, all triples (c, j, a) such that $j < l$ are kept in R, i.e. the current stages of the induction on φ_j with $j < l$ are left untouched. On the other hand, all triples (c, j, a) for $j > l$ are removed from R, i.e. the fixed-point induction on the formulae φ_j, which might depend on R_l, are set back to the empty set.

Thus, in the end there will be no such l as all fixed points are already computed. In this case the relation R is left unchanged and thus the fixed point of *compute* has been reached. This proves the following lemma.

7.15 Lemma. $\textsc{Sat}_{\text{IFP}}$ *is definable in* PFP.

The proof of the next theorem and its corollary is now immediate.

7.16 Theorem. PFP *is more expressive than* IFP *on acceptable structures.*

7.17 Corollary. PFP *is more expressive than* IFP *on all structures in which an acceptable structure is* PFP*-interpretable.*

Among the structures in which an acceptable structure is PFP-interpretable are $(\omega, <)$ and $(\mathbb{R}, <, +)$ and all expansions of it, e.g. the ordered field of reals. Examples of structures not interpretable in an acceptable structure are structures over the empty signature or a signature containing constant symbols only, but also the real line $(\mathbb{R}, <)$.

Chapter 8

Expressive Equivalence of Least and Inflationary Fixed-Point Logic

In this section, we establish the equivalence of least and inflationary fixed-point logic. As noted above, in the restriction to finite structures, the equivalence has already been proved by Gurevich and Shelah [GS86]. We first present their proof and explain where its extension to infinite structures fails.

8.1 Equivalence on Finite Structures

Consider again the proof of Theorem 3.37. As shown there, the stage comparison relations of any IFP-formula[1] $\varphi(R, \overline{x})$ are definable by the formulae $[\mathbf{ifp} \leq : S](\overline{x}, \overline{y})$ and $[\mathbf{ifp} \prec : S](\overline{x}, \overline{y})$ respectively, where S is the system of formulae defined as

$$S := \begin{cases} \overline{x} \leq \overline{y} & \longleftarrow \quad \varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{y}) \wedge \varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{y}) \\ \overline{x} \prec \overline{y} & \longleftarrow \quad \varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{x}) \wedge \neg\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x}). \end{cases}$$

Now suppose $\varphi(R, \overline{x})$ is itself an LFP-formula but not necessarily positive in R. It was shown by Gurevich and Shelah, that in restriction to finite structures, the stage comparison relations for the inflationary induction on φ are definable in LFP. As the inflationary fixed point can easily be obtained from the stage comparison relations (see Lemma 3.10), this shows that on finite structures, every inflationary fixed point of an LFP-formula can be obtained as a least fixed point also. By induction on the number of **ifp**-operators in the formulae, the equivalence of IFP and LFP follows immediately.

[1]Recall our convention that φ is supposed to be of the form $R\overline{x} \vee \varphi'(R, \overline{x})$.

The following lemma and its proof demonstrate how the stage comparison relations for an arbitrary LFP-formula $\varphi(R,\overline{x})$ can be defined in LFP. See also the original paper [GS86].

8.1 Lemma. *On finite structures, the stage comparison relations $\leq_\varphi$ and $\prec_\varphi$ of any formula $\varphi(R,\overline{x}) \in$ LFP, not necessarily positive in R, are LFP-definable.*

Proof. To prove the lemma we convert the system S above to an equivalent system T of formulae, which are positive in their free fixed-point variables. W.l.o.g. we assume that φ is of the form $R\overline{x} \vee \varphi'$. The problem to be solved is that if every atom $R\overline{u}$ in φ is replaced by a new atom involving $\prec$, then at all places where R is used negatively, also the new relation $\prec$ is used negatively. Therefore, we have to come up with a definition of the complement R^c of R by a formula positive in $\prec$ and $\leq$.

For this, let $\mathfrak{A}$ be a finite structure of size n. Clearly, if k is the arity of R, then there is some $m \leq n^k$ such that the induction of φ on $\mathfrak{A}$ reaches its fixed point at stage m. Consider the sequence of stages $(R^\alpha)_{\alpha \leq m}$ induced by φ on $\mathfrak{A}$. Let $\leq_\varphi$ and $\prec_\varphi$ be the stage comparison relations of φ. For every stage R^α, with $m \geq \alpha > 0$, there is a tuple $\overline{z}$ whose rank is precisely α, i.e. $\overline{z} \in R^\alpha - R^{\alpha-1}$. For any such tuple $\overline{z}$, $\{\overline{u} : \overline{u} \leq_\varphi \overline{z}\} = R^\alpha$ and $\{\overline{u} : \overline{z} \prec_\varphi \overline{u}\} = (R^\alpha)^c$. This is used to define an induction process, positive in $\leq$ and $\prec$, defining the relations $\leq_\varphi$ and $\prec_\varphi$.

Let $\varphi(\overline{y}, R\overline{u}/\overline{u} \leq \overline{z}, \neg R\overline{u}/\overline{z} \prec \overline{u})$ be the formula obtained from φ by replacing every positive occurrence of an atom $R\overline{u}$ by $\overline{u} \leq \overline{z}$ and every negative occurrence by $\overline{z} \prec \overline{u}$. Suppose the relations $\leq$ and $\prec$ are defined up to some stage $\beta \geq 1$, i.e. $\overline{x} \leq \overline{y}$ if, and only if, $|\overline{x}| \leq |\overline{y}| \leq \beta$ and likewise for $\prec$. Then the formula

$$\begin{aligned}\psi_{\leq}(\leq,\prec,\overline{x},\overline{y}) := \exists\overline{z}(\overline{z} \prec \overline{y} \wedge \varphi(\overline{x}, R\overline{u}/\overline{u} \leq \overline{z}, \neg R\overline{u}/\overline{z} \prec \overline{u}) \wedge \\ \varphi(\overline{y}, R\overline{u}/\overline{u} \leq \overline{z}, \neg R\overline{u}/\overline{z} \prec \overline{u})),\end{aligned}$$

becomes true for those pairs $(\overline{x},\overline{y})$ such that $1 \leq |\overline{y}|_\varphi \leq \beta+1$ and $|\overline{x}|_\varphi \leq |\overline{y}|_\varphi$. For all tuples $\overline{y}$ of rank $1 \leq \xi \leq \beta+1$, we can take some $\overline{z}$ of rank $\xi - 1$ such that $\overline{y}$ satisfies $\varphi(\overline{y}, R\overline{u}/\overline{u} \leq \overline{z}, \neg R\overline{u}/\overline{z} \prec \overline{u})$. Clearly, $\varphi(\overline{x}, R\overline{u}/\overline{u} \leq \overline{z}, \neg R\overline{u}/\overline{z} \prec \overline{u})$ is then satisfied by those tuples $\overline{x}$ whose rank is at most ξ.

Analogously, the formula

$$\begin{aligned}\psi_{\prec}(\leq,\prec,\overline{x},\overline{y}) := \exists\overline{z}(\overline{z} \leq \overline{z} \wedge \varphi(\overline{x}, R\overline{u}/\overline{u} \leq \overline{z}, \neg R\overline{u}/\overline{z} \prec \overline{u}) \wedge \\ \neg\varphi(\overline{y}, R\overline{u}/\neg\overline{z} \prec \overline{u}, \neg R\overline{u}/\neg\overline{u} \leq \overline{z})),\end{aligned}$$

becomes true for those pairs $(\overline{x},\overline{y})$ such that $|\overline{x}|_\varphi \leq \beta+1$ and $|\overline{x}|_\varphi < |\overline{y}|_\varphi$. To see this, let $\overline{z}$ be a tuple such that $\overline{z} \leq \overline{z}$ becomes true. Thus, the rank ξ of $\overline{z}$ is at most β. Now, for such a tuple $\overline{z}$, $\overline{u} \leq \overline{z}$ defines the stage R^ξ and $\overline{z} \prec \overline{u}$ defines its complement. Thus, $\varphi(\overline{x}, R\overline{u}/\overline{u} \leq \overline{z}, \neg R\overline{u}/\overline{z} \prec \overline{u})$ is satisfied by all tuples $\overline{x}$ of rank at most $\xi+1$. Further, $\neg\varphi(\overline{y}, R\overline{u}/\neg\overline{z} \prec \overline{u}, \neg R\overline{u}/\neg\overline{u} \leq \overline{z})$

is satisfied by those $\overline{y}$ whose rank is greater than $\xi + 1$. Thus, as for every rank up to β we can find tuples $\overline{z}$ of precisely this rank, the claim is proved.

So far, we have defined formulae that, given some stage $\beta \geq 1$, define the next stage of the induction on the stage comparison relations $\leq$ and $\prec$. What is left to be done is to get the process started, i.e. to define the stage comparison relations for those pairs $(\overline{x}, \overline{y})$ where $\overline{x}$ is of rank 0 (for $\prec$) or both, $\overline{x}$ and $\overline{y}$ are of rank 0 (for $\leq$). This, of course, can easily be defined by the formula $\varphi(\overline{x}, \varnothing) \wedge \varphi(\overline{y}, \varnothing)$ for $\leq$ and $\varphi(\overline{x}, \varnothing) \wedge \neg\varphi(\overline{y}, \varnothing)$ for $\prec$, where $\varphi(\overline{x}, \varnothing)$ means that every occurrence of an atom $R\overline{u}$ is replaced by a formula equivalent to *false*.

Thus, if T is the system of formulae defined as

$$T := \begin{cases} \overline{x} \leq \overline{y} & \longleftarrow \quad (\varphi(\overline{x}, \varnothing) \wedge \varphi(\overline{y}, \varnothing)) \vee \psi_{\leq}(\overline{x}, \overline{y}) \\ \overline{x} \prec \overline{y} & \longleftarrow \quad (\varphi(\overline{x}, \varnothing) \wedge \neg\varphi(\overline{y}, \varnothing)) \vee \psi_{\prec}(\overline{x}, \overline{y}) \end{cases}$$

then $[\mathbf{lfp} \leq : T](\overline{x}, \overline{y})$ defines $\leq_\varphi$ and $[\mathbf{lfp} \prec : T](\overline{x}, \overline{y})$ defines $\prec_\varphi$. □

Using that $\overline{a} \in \varphi^\infty$ if, and only if, $\overline{a} \leq_\varphi \overline{a}$ (see Lemma 3.10), the next theorem follows immediately by induction on the number of fixed-point operators in the IFP-formulae.

8.2 Theorem (Gurevich-Shelah [GS86]). *For every formula in* IFP *there is an* LFP*-formula equivalent to it on all finite structures.*

Another consequence of the lemma is the normal form Theorem proved by Immerman for IFP and LFP on finite structures.

8.3 Theorem (Immerman [Imm86]). *On finite structures, every formula in* LFP *is equivalent to a formula with at most one occurrence of an* **lfp***-operator.*

Sketch. If $\leq$ and $\prec$ are the stage comparison relations for φ on a finite structure, then the formula

$$\varphi_{max}(\overline{x}) := \overline{x} \leq \overline{x} \wedge \forall \overline{z}(\overline{z} \leq \overline{x} \vee \neg\varphi(\overline{z}, R\overline{u}/\neg\overline{x} \prec \overline{u}, \neg R\overline{u}/\neg\overline{u} \leq \overline{x}))$$

defines the elements of maximal rank in the fixed point of φ. Now, a formula $\psi(\overline{x}) := \neg[\mathbf{lfp}_{R,\overline{x}}\,\varphi](\overline{x})$ in LFP is equivalent to the formula

$$\forall \overline{y} \neg\varphi(\overline{y}, \varnothing) \vee \exists \overline{z}(\varphi_{max}(\overline{z}) \wedge \overline{z} \prec \overline{x}).$$

If the fixed point of φ is empty, then the first disjunct becomes true for all tuples. Otherwise, i.e. if the fixed point of φ is not empty, then $\exists \overline{z}(\varphi_{max}(\overline{z}) \wedge \overline{z} \prec \overline{x}))$ defines a tuple $\overline{z}$ of maximal rank with respect to the stage comparison relation for φ. Clearly, the complement of φ's fixed point is just the set of elements of rank greater than the rank of $\overline{z}$. □

We now aim at extending the equivalence of IFP and LFP to arbitrary, not necessarily finite structures. If $\mathfrak{A}$ is an infinite structure, the sequence of stages induced by an LFP-formula $\varphi(R, \overline{x})$ on $\mathfrak{A}$ is no longer guaranteed to be finite. The formulae used in the proof above still define the correct stage comparison relations up to stage ω, i.e. for all finite stages. However, at stage ω - and all other infinite limit stages also - it is no longer true that there is a tuple $\overline{z}$ of rank less than ω such that $\overline{u} \leq \overline{z}$ defines $R^{<\omega}$. For, each such tuple $\overline{z}$ is itself of finite rank $\beta < \omega$ and therefore $\overline{u} \leq \overline{z}$ defines the stage $R^{\beta} \subsetneq R^{\omega}$. Thus, to extend the result to infinite structures, we have to treat the limit stages differently.

8.2 Equivalence in the General Case

In this section we aim at establishing the equivalence of IFP and LFP on arbitrary structures. Towards this, let $\varphi'(R, \overline{x})$ be in LFP, not necessarily positive in R, and consider the formula $\varphi := R\overline{x} \vee \varphi'(\overline{x})$. Clearly, φ and φ' have the same inflationary fixed point. Fix φ for the rest of the section.

Notation. Let $\psi_1(\overline{x})$, $\psi_2(\overline{x})$ be formulae, which may or may not contain the relation symbol R, such that $\overline{x}$ and R have the same arity. We write $\varphi(\overline{x}, R\overline{u}/\psi_1(\overline{u}), \neg R\overline{u}/\psi_2(\overline{u}))$ to denote the formula obtained from φ by replacing each positive occurrence of atoms of the form $R\overline{u}$, for some tuple $\overline{u}$ of terms, by $\psi_1(\overline{u})$ and each negative occurrence of atoms of the form $R\overline{u}$ by $\neg\psi_2(\overline{u})$. Clearly, the formula is positive both in ψ_1 and ψ_2 and thus positive in R if ψ_1 and ψ_2 are. $\square$

We aim at defining the stage comparison relation $\prec_\varphi$ for φ in LFP. Consider again the proof of the Stage Comparison Theorem 3.37. We showed that $\prec_\varphi$ can be defined by the inflationary fixed point of the formula

$$\begin{aligned}\varphi'(\prec, \overline{x}, \overline{y}) \;:=\; & \varphi(\overline{x}, R\overline{u}/\overline{u} \prec \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x}) \;\wedge \\ & \neg\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x}),\end{aligned}$$

where $\prec$ is a second-order variable of appropriate arity. To turn this into a formula in LFP we have to replace the formula $\neg\overline{u} \prec \overline{x}$ by a definition positive in $\prec$. Essentially, we define a second formula $\vartheta(\ll, \overline{x}, \overline{y})$, with free second-order variables $\ll$ and $\prec$, such that ϑ is negative in $\prec$ and if $\prec$ is interpreted by a given stage $\prec^\alpha$, for some ordinal α, then the least fixed point $\ll^\infty$ of ϑ is just $\prec^\alpha$. We can then use $[\mathbf{lfp}_{\ll,\overline{x},\overline{y}}\vartheta]$ negatively to get the desired positive definition of $\prec$.

Unfortunately, by definition, the relation defined by such a formula must increase with increasing stages $\prec^\alpha$. On the other hand, as ϑ is supposed to be negative – and therefore antitone – in $\prec$, the relation defined by ϑ must decrease with increasing stages $\prec^\alpha$. Thus, in general, we cannot hope for such a formula to exist. Instead we will use a formula defining a slightly different relation. But it might be helpful to keep the original idea in mind.

Consider the following formula

$$\chi(\overline{x},\overline{y}) := [\mathbf{lfp}_{\prec,\overline{x},\overline{y}}\,\chi'(\overline{x},\overline{y})](\overline{x},\overline{y}),$$

where

$$\begin{aligned}\chi'(\overline{x},\overline{y}) :=\ & \varphi(\overline{x}, R\overline{u}/u \prec \overline{x}, \neg R\overline{u}/\neg\overline{u} \lhd \overline{x}) \wedge \\ & \forall\overline{u}(\overline{u} \prec \overline{x} \vee \neg\overline{u} \lhd \overline{x}) \wedge \\ & \neg\varphi(\overline{y}, R\overline{u}/\overline{u} \lhd \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x})\end{aligned}$$

and

$$\overline{x} \lhd \overline{y} := [\mathbf{lfp}_{\ll,\overline{x},\overline{y}}\,\vartheta(\ll,\overline{x},\overline{y})](\overline{x},\overline{y})$$

where

$$\begin{aligned}\vartheta(\overline{x},\overline{y}) :=\ & \varphi(\overline{x}, R\overline{u}/\overline{u} \ll \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x}\}) \wedge \\ & \neg\exists\overline{u}(\overline{u} \prec \overline{x} \wedge \neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y})) \wedge \\ & \neg\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x}, \neg R\overline{u}/\neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y})).\end{aligned}$$

Obviously, the formula χ' is positive in $\prec$ and is itself a formula in LFP. Thus the least fixed point of χ' exists. We claim that this fixed point defines the stage comparison relation $\prec_\varphi$ of φ. Before proving this we first have to establish some facts about the sub-formula ϑ. Recall from the beginning of this section that φ is supposed to be of the form $R\overline{x} \vee \varphi'$. This is important for the proofs below as it ensures that whenever a tuple $\overline{x}$ satisfies φ at a stage α, it satisfies φ at all higher stages also.

8.4 Lemma. *Consider the fixed-point induction on ϑ where $\prec$ is interpreted by $\prec^{<\alpha}$, i.e. $\overline{x} \prec \overline{y}$ if, and only if, $\overline{x} \in \varphi^{<\alpha}$ and $|\overline{x}|_\varphi < |\overline{y}|_\varphi$.*

(*i*) *If $\overline{x} \in \varphi^\alpha$ or $\overline{y} \in \varphi^\alpha$, then $(\overline{x},\overline{y}) \in \vartheta^\infty$ if, and only if, $|\overline{x}|_\varphi < |\overline{y}|_\varphi$.*

(*ii*) *For all $\overline{y}$ such that $|\overline{y}|_\varphi > \alpha$ there is an $\overline{x}$ such that $|\overline{x}|_\varphi = \alpha$ and $(\overline{x},\overline{y}) \in \vartheta^\infty$.*

(*iii*) *If the fixed-point of $\prec$ has already been reached, i.e. if $\prec^\alpha = \prec^{<\alpha}$, then $\vartheta^\infty = \prec^\alpha$.*

Proof. Throughout this proof, the variable $\prec$ will always be interpreted by the set $\prec^{<\alpha}$. Therefore we will sometimes drop the index and write $\prec$ for $\prec^{<\alpha}$.

1. We prove by induction on β that for all $\beta < \alpha$, $(\overline{x},\overline{y}) \in \vartheta^\beta$, i.e.

   $$(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\beta}) \models \vartheta(\overline{x},\overline{y}), \text{ if, and only if, } \overline{x} \in \varphi^\beta \text{ and } |\overline{x}|_\varphi < |\overline{y}|_\varphi.$$

   Again we omit the index most of the times and write $\ll$ for $\ll^{<\beta}$.

   Suppose that for all $\gamma < \beta$ the claim has been proved. We distinguish between the case where $\overline{x} \in \varphi^\beta$ and $\overline{x} \notin \varphi^\beta$.

- Suppose $\overline{x} \in \varphi^{\beta}$. We show that $(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\beta}) \models \vartheta(\overline{x}, \overline{y})$, if, and only if, $|\overline{x}|_{\varphi} < |\overline{y}|_{\varphi}$.

  By induction hypothesis, if $\overline{x} \in \varphi^{\beta}$ then for all $\overline{u}$, $\overline{u} \ll \overline{x}$ if, and only if, $|\overline{u}|_{\varphi} < |\overline{x}|_{\varphi}$ and, as $\beta < \alpha$, $\neg\overline{u} \prec \overline{x}$ if, and only if, $|\overline{u}|_{\varphi} \geq |\overline{x}|_{\varphi}$. Thus,

  $$(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\beta}) \models \varphi(\overline{x}, R\overline{u}/\overline{u} \ll \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x}\}).$$

  Now consider $\overline{y}$. If $|\overline{y}|_{\varphi} > |\overline{x}|_{\varphi}$, then $\neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y})$ reduces to $\neg\overline{u} \ll \overline{x}$. As $\beta < \alpha$, $\neg\overline{u} \ll^{<\beta} \overline{x}$ is equivalent to $\neg\overline{u} \prec^{<\alpha} \overline{x}$. Therefore there is no $\overline{u}$ satisfying $(\overline{u} \prec \overline{x} \wedge \neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y}))$ and the second conjunct in ϑ is satisfied. Further, $\overline{y}$ does not satisfy $\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x}, \neg R\overline{u}/\neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y}))$ as otherwise $|\overline{y}|_{\varphi} \leq |\overline{x}|_{\varphi}$. Thus, $(\overline{x}, \overline{y}) \in \vartheta^{\beta}$.

  On the other hand, if $|\overline{y}|_{\varphi} < |\overline{x}|_{\varphi}$, then $(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y})$ in the second conjunct reduces to $\overline{u} \prec \overline{y}$ and thus there is a $\overline{u}$ satisfying $\overline{u} \prec \overline{x} \wedge \neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y}))$, $\overline{u} := \overline{y}$ for instance.

  Finally, suppose $|\overline{x}|_{\varphi} = |\overline{y}|_{\varphi}$. By the same argument as above we get that in this case

  $$(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\beta}) \models \varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x}, \neg R\overline{u}/\neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y}))$$

  and thus ϑ is not satisfied.

- Suppose $\overline{x} \notin \varphi^{\beta}$. We show that $\varphi(\overline{x}, R\overline{u}/\overline{u} \ll \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x})$ is not satisfied. By induction hypothesis, $\overline{u} \ll \overline{x}$ defines the set $M := \varphi^{<\beta}$. Clearly, as $\overline{x} \notin \varphi^{\beta}$,

  $$\mathfrak{A} \not\models \varphi(\overline{x}, R\overline{u}/\overline{u} \in M, \neg R\overline{u}/\overline{u} \in M^{c}).$$

  Now consider the set $N := \{\overline{u} : \neg\overline{u} \prec \overline{x}\}$. As $\overline{x} \notin \varphi^{\beta}$, we get $M^{c} \supseteq N$, where M^{c} denotes the complement of M.

  By monotonicity of φ in M and M^{c} it follows that

  $$(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\beta}) \not\models \varphi(\overline{x}, R\overline{u}/\overline{u} \ll \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x}\}.$$

We get that for any pair $(\overline{x}, \overline{y})$, $(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\beta}) \models \vartheta(\overline{x}, \overline{y})$ if, and only if, $\overline{x} \in \varphi^{\beta}$ and $|\overline{x}|_{\varphi} < |\overline{y}|_{\varphi}$.

2. Part 1 implies that $(\overline{x}, \overline{y}) \in \vartheta^{<\alpha}$ if, and only if, $\overline{x} \in \varphi^{<\alpha}$ and $|\overline{x}|_{\varphi} < |\overline{y}|_{\varphi}$. Thus, $\ll^{<\alpha} = \prec^{<\alpha}$. Now consider the next induction step. Again we distinguish between $\overline{x} \in \varphi^{\alpha}$ and $\overline{x} \notin \varphi^{\alpha}$.

   - Suppose $\overline{x} \in \varphi^{\alpha}$. Obviously,

     $$(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\alpha}) \models \varphi(\overline{x}, R\overline{u}/\overline{u} \ll \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x}).$$

If $|\overline{y}|_\varphi \geq |\overline{x}|_\varphi$, then $(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y})$ reduces to $\overline{u} \prec \overline{x}$ and thus $(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\alpha}) \models \neg\varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x}, \neg R\overline{u}/\neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y}))$ if, and only if, $|\overline{y}|_\varphi > |\overline{x}|_\varphi$.

Now suppose $|\overline{y}|_\varphi < |\overline{x}|_\varphi$. Then $|\overline{y}|_\varphi < \alpha$ and there is an $\overline{u}$ satisfying $\overline{u} \prec \overline{x} \wedge \neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y})$, again $\overline{y}$ being itself a witness for this. Thus $\vartheta(\overline{x}, \overline{y})$ is not satisfied.

- Now assume $\overline{x} \notin \varphi^\alpha$. Then

$$(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\alpha}) \not\models \varphi(\overline{x}, R\overline{u}/\overline{u} \ll \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x}\})$$

as $\overline{u} \ll \overline{x}$ defines the set $\overline{u} \in \varphi^{<\alpha}$ and $\neg\overline{u} \prec \overline{x}$ its complement.

It follows, that ϑ^α contains all pairs $(\overline{x}, \overline{y})$ such that $\overline{x} \in \varphi^\alpha$ and $|\overline{x}|_\varphi < |\overline{y}|_\varphi$. This proves Part *(ii)* because if there is a tuple $\overline{y}$ of rank greater than α there must also be a tuple $\overline{x}$ of rank exactly α and this pair would be in ϑ^α.

Further, if the fixed point of $\prec$ has already been reached, i.e. $\prec^\alpha = \prec^{<\alpha}$, then there are no tuples $\overline{x}$ of rank exactly α. In this case, all tuples $(\overline{x}, \overline{y}) \in \vartheta^\alpha$ already occur in $\vartheta^{<\alpha}$ and the fixed point of ϑ has been reached. This proves Part *(iii)* of the lemma. Thus, from now on, we assume that $\prec^{<\alpha} \subsetneq \prec^\alpha$.

3. We show now that at no stage $\gamma > \alpha$ can a pair $(\overline{x}, \overline{y})$ with $\overline{x}, \overline{y} \in \varphi^\alpha$ and $|\overline{y}|_\varphi \leq |\overline{x}|_\varphi$ enter the fixed point. Towards a contradiction let γ be the smallest such stage and let $(\overline{x}, \overline{y})$ be as described. Then the same argument as in the first item of Step 1 yields a contradiction.

4. What is left to be shown is that for no $\overline{x} \notin \varphi^\alpha$ and $\overline{y} \in \varphi^\alpha$ the pair $(\overline{x}, \overline{y})$ enters the fixed point at some higher stage. Towards a contradiction, let γ be the least such stage, i.e. the least stage such that there is a pair $(\overline{x}, \overline{y}) \in \vartheta^\gamma$ with $\overline{x} \notin \varphi^\alpha$ and $\overline{y} \in \varphi^\alpha$. In particular, $(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\gamma}) \models \vartheta(\overline{x}, \overline{y})$.

   Now, as $\overline{x} \notin \varphi^\alpha$, $\overline{u} \prec \overline{x}$ defines just $\varphi^{<\alpha}$ and, as γ was chosen minimal, we get that $\neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y})$ defines the set of tuples $\overline{u}$ such that $|\overline{u}|_\varphi \geq |\overline{y}|_\varphi$. Thus, if $|\overline{y}|_\varphi < \alpha$ then there is a tuple $\overline{u}$ satisfying $\overline{u} \prec \overline{x} \wedge \neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y})$ and thus ϑ is not satisfied by $(\overline{x}, \overline{y})$. On the other hand, if $|\overline{y}|_\varphi = \alpha$, then $(\mathfrak{A}, \prec^{<\alpha}, \ll^{<\alpha}) \models \varphi(\overline{y}, R\overline{u}/\overline{u} \prec \overline{x}, \neg R\overline{u}/\neg(\overline{u} \ll \overline{x} \wedge \overline{u} \ll \overline{y}))$ and again ϑ is not satisfied.

This finishes the proof of the lemma. □

We now prove a technical lemma which will establish the induction step in the proof that the fixed point of χ' defines $\prec_\varphi$.

8.5 Lemma. *Let $\prec^{<\alpha}$ be the strict stage comparison relation up to stage α, i.e., $\overline{x} \prec^{<\alpha} \overline{y}$ if, and only if, $\overline{x} \in \varphi^{<\alpha}$ and $|\overline{x}|_\varphi < |\overline{y}|_\varphi$.*

Then $(\mathfrak{A}, \prec^{<\alpha}) \models \chi'(\overline{x}, \overline{y})$, if, and only if, $\overline{x} \in \varphi^\alpha$ and $|\overline{x}|_\varphi < |\overline{y}|_\varphi$.

Proof. We distinguish between the cases where $\overline{x} \in \varphi^\alpha$ and $\overline{x} \notin \varphi^\alpha$.

- Suppose $\overline{x} \in \varphi^\alpha$. By assumption, $\overline{u} \prec^{<\alpha} \overline{x}$ defines the set $\{\overline{u} : |\overline{u}|_\varphi < |\overline{x}|_\varphi\}$ and, by Part (i) of Lemma 8.4, $\neg\overline{u} \lhd \overline{x}$ defines its complement. Thus, $(\mathfrak{A}, \prec^{<\alpha}) \models \varphi(\overline{x}, R\overline{u}/u \prec \overline{x}, \neg R\overline{u}/\neg\overline{u} \lhd \overline{x})$ and all $\overline{u}$ satisfy $\overline{u} \prec \overline{x} \vee \neg\overline{u} \lhd \overline{x}$.

  Now, $(\mathfrak{A}, \prec^{<\alpha}) \models \neg\varphi(\overline{y}, R\overline{u}/\overline{u} \lhd \overline{x}, \neg R\overline{u}/\neg\overline{u} \prec \overline{x})$ if, and only if, $|\overline{y}|_\varphi > |\overline{x}|_\varphi$.

  Thus, $(\mathfrak{A}, \prec^{<\alpha}) \models \chi'(\overline{x}, \overline{y})$ if, and only if, $|\overline{y}|_\varphi > |\overline{x}|_\varphi$.

- Suppose $\overline{x} \notin \varphi^\alpha$. Then $\overline{u} \prec \overline{x}$ defines the set $\{\overline{u} : \overline{u} \in \varphi^{<\alpha}\}$. If $\varphi^{<\alpha} = \varphi^\alpha$, i.e. if the fixed point of φ has been reached, then, by Part (iii) of Lemma 8.4, we get $\lhd = \prec$ and $(\mathfrak{A}, \prec^{<\alpha}) \not\models \varphi(\overline{x}, R\overline{u}/u \prec \overline{x}, \neg R\overline{u}/\neg\overline{u} \lhd \overline{x})$ and therefore χ' is not satisfied.

  Otherwise, i.e. if $\varphi^{<\alpha} \subsetneq \varphi^\alpha$, then, by Part (ii) of Lemma 8.4, there is a tuple $\overline{a}$ of rank α with $\overline{a} \lhd \overline{x}$. Thus, the conjunct $\forall\overline{u}(\overline{u} \prec \overline{x} \vee \neg\overline{u} \lhd \overline{x})$ is not satisfied as $\overline{a} \lhd \overline{x}$ but $\overline{a} \not\prec \overline{x}$.

This finishes the proof of the lemma. □

As corollary we get that the relation $\prec_\varphi$ is definable in LFP.

8.6 Corollary. *Let $\varphi(R, \overline{x})$ be a formula in* LFP*. Then the stage comparison relation $\prec_\varphi$ of the inflationary fixed point of φ is definable in* LFP*.*

Proof. A simple induction on the stages using the previous lemma shows that $\prec_\varphi$ is defined by the formula χ above. □

The equivalence of LFP and IFP follows immediately.

8.7 Theorem. *For every formula in* IFP *there is an equivalent formula in* LFP*.*

Proof. By Corollary 8.6, for every $\varphi(R, \overline{x}) \in$ LFP the relation $\prec_\varphi$ is definable in LFP. Thus, for all $\overline{x}$, $\overline{x} \in \varphi^\infty$ if, and only if, $\mathfrak{A} \models \varphi(\overline{x}, R\overline{u}/\chi(\overline{u}, \overline{x}))$, where χ is the formula defining $\prec_\varphi$. Thus, the inflationary fixed point of an LFP-formula can be defined in LFP.

For arbitrary formulae $\varphi \in$ IFP, the theorem follows by induction on the number of inflationary fixed points in φ converting them to least fixed points from the inside out. □

The theorem shows that also on infinite structures, least and inflationary fixed-point logic have the same expressive power. But, contrary to the case of finite structures where the translation of IFP-formulae to equivalent LFP-formulae does not alter the fixed-point structure, in the general case their structure in terms of alternations between **lfp**-operators and negation and the nesting depth of fixed-point operators becomes more complicated. It might be possible to reduce the increase in the number of alternations of the resulting LFP-formulae. However, we will show below that an increase in the number of alternations cannot be avoided.

8.8 Theorem. *For every $n \geq 0$,*

$$\mathrm{LFP}^a_n \subseteq \mathrm{IFP}^d_n \subseteq \mathrm{LFP}^a_{3^n}$$

Proof. Let $\varphi \in \mathrm{LFP}^a_n$ be a formula with alternation depth n. It is known that nested **lfp**-operators which all occur positively can be contracted to a single **lfp**-operator increasing the arity. Thus, every formula in LFP^a_n is equivalent to a formula with n nested fixed-points. By Theorem 3.34 above, this formula again is equivalent to an IFP formula with nesting depth n.

Towards the second containment, note that using the method of Theorem 8.7 to convert an IFP-formula to an equivalent LFP-formula, the translation of each individual **ifp**-operator at most triples the alternation depth. The theorem now follows by induction. □

We immediately get the following corollaries.

8.9 Corollary. *For any structure, the alternation depth hierarchy for* LFP *collapses if, and only if, the nesting depth hierarchy for* IFP *collapses.*

The corollary implies that for every structure $\mathfrak{A}$, either both the nesting and the alternation depth hierarchy collapse on $\mathfrak{A}$ for LFP and IFP, or we get the following diagram.

	Alternation	Nesting
LFP	strict	collapse
IFP	collapse	strict

It is open whether there are infinite structures on which the alternation and nesting depth hierarchies for LFP and IFP collapse but where LFP is still more expressive than FO.

An example of a class of structures where the hierarchies are strict is the class of acceptable structures (see Definition 7.7).

8.10 Corollary. *Let $\mathfrak{A}$ be acceptable. Then the nesting depth hierarchy for* IFP *is strict. The same holds, if $\mathfrak{A}$ is not acceptable but allows the definition of a coding scheme in* IFP.

Proof. This follows immediately from Corollary 8.9 and the results of Moschovakis [Mos74a] and Bradfield [Bra98a] on the alternation hierarchy of LFP on acceptable structures. See Theorem 3.25. □

8.3 Open Problems

In the preceding section we established the equivalence of least and inflationary fixed-point logic. The proof was by giving an explicit transformation of inflationary into least fixed-point formulae. As we have seen, this transformation increases the complexity and alternation depth of the resulting formulae significantly, i.e. exponentially. It would be interesting to know whether the exponential blow up in the number of fixed-point operators is necessary or whether the proof can be simplified to a linear increase in the alternation depth.

Another open problem concerns the arity of the involved fixed-point operators. The transformation of an **ifp**-operator into an **lfp**-operator as given above doubles the arity of the fixed-point relation. Whether this increase in arity can be avoided is an important open problem. Note that giving a positive answer to this question on finite ordered structures would separate PTIME from LOGSPACE. For, it was shown by Grohe and Imhof [Gro94, Imh96a], that on finite ordered structures, deterministic transitive-closure logic DTC, which captures LOGSPACE, is contained in binary LFP and monadic IFP. As LFP captures PTIME, a positive answer for some $k \geq 2$ to the problem on finite ordered structures would imply DTC $\subsetneq$ LFP and therefore LOGSPACE $\neq$ PTIME.

Part II

Modal Fixed-Point Logics

Chapter 9

Modal Logics and Bisimulation

The central issue in the area of verification are formal methods for verifying that a given process, e.g. a hard- or software system, meets certain criteria or specifications. Typically, the processes are modelled by *transition systems*, which essentially are directed, edge and node labelled graphs. A node in a transition system represents a state in which the process can get and the edge relation models the transition between states the process can take.

Logics now serve as the basis of specification languages. The criteria or specifications the process is supposed to satisfy are formalised in the logic. Testing whether the process meets the criteria boils down to checking whether the transition system modelling the process satisfies the formula corresponding to the specification.

For this purpose, logics like LFP based on first-order logic are much too powerful. For one, all these logics have an undecidable satisfiability problem and their model checking complexity is far too high. One the other hand, they are able to express properties of the transition system which are not reflected in the process the system models. An important requirement for specification languages is that they should not be able to distinguish between different models describing the same process, i.e. model the same behaviour. The standard notion used to describe behavioural equivalence of processes is the so-called *bisimulation*, which will be described in some detail in Section 9.2. Thus, no specification language should be able to distinguish between bisimilar models. First-order logics and all its extensions, however, do not respect bisimulation in this sense and thus do not form a suitable basis upon which specification languages can be built.

Therefore, instead of first-order logic, a much weaker logic, the so-called *modal logic*, is used and extensions of modal logic form the basis for important specification languages such as LTL, CTL, and the μ-calculus. Modal logic is the topic of Section 9.3 and much of this second part of the thesis is devoted to the study of fixed-point extensions of modal logic.

9.1 Transition Systems

Transition systems are directed, edge and node labelled graphs.

9.1 Definition (Transition system). *Fix a set $\mathcal{A}$ of actions and a set $\mathcal{P}$ of atomic propositions. A* transition system for $\mathcal{A}$ and $\mathcal{P}$, *also called* Kripke structure, *is a structure $\mathcal{K} := (V, (E_a)_{a\in\mathcal{A}}, (p)_{p\in\mathcal{P}})$ with universe V, binary relations $E_a \subseteq V \times V$ for each $a \in \mathcal{A}$, and monadic relations $p \subseteq V$ for each atomic proposition $p \in \mathcal{P}$.*

We do not notationally distinguish between atomic propositions and their interpretations. Like processes, which always have an initial state, transition systems are always considered with a distinguished node, called the *root* of the system. We write $\mathcal{K}, v$ for a transition system $\mathcal{K}$ with distinguished node v.

The following example demonstrates the use of transition systems for the modelling of processes. This will serve as a running example on which the various notions presented later are motivated.

9.2 Example. *Consider a print server managing one printer for two different users. Before a user can print a file, he first has to send a* request message *to the server. If the server has not received a request from the other user before or has finished printing the other user's file, the first user can send his file which is then printed. If a printer request arrives at the server while it is printing a file, the request is scheduled and processed after the file has been printed. For simplicity, we assume that no user can send a request while he is printing a file. Figure 9.1 shows a transition system modelling the print server described here. The label R_i stands for User i having requested the printer and P_i stands for User i printing a file.*

The next definition collects some frequently used notions about transition systems.

9.3 Definition. (*i*) *A transition system $\mathcal{K}, v$ is* connected *if for every node u, there is a path in $\mathcal{K}$ from v to u.*

(*ii*) *Let $\mathcal{K}$ be a transition system with universe V. A node $v \in V$ is* well-founded *if no infinite path in $\mathcal{K}$ starts at v.*

(*iii*) *The* height $h(v)$ *of a well-founded node $v \in V$ is defined as the least strict upper bound of the height of its children. In particular, leaves are of height* 0.

(*iv*) *A transition system $\mathcal{K}$ with distinguished node v is* well-founded *if v is well-founded. If $\mathcal{K}, v$ is well-founded, its* height *is defined as the height of v.*

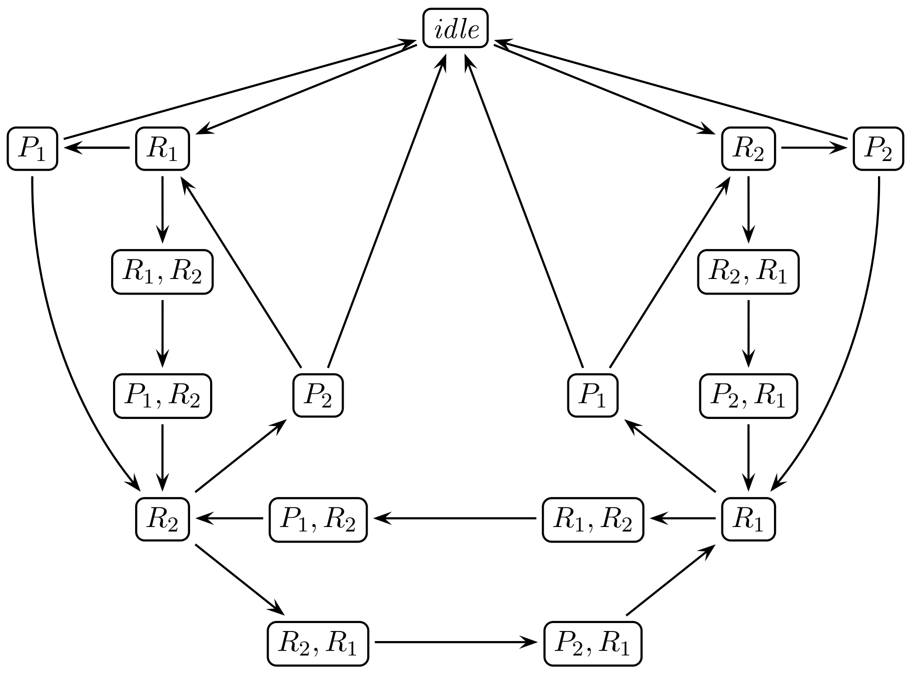


Figure 9.1: Transition system modelling the print server described in Example 9.2.

(*v*) *A transition system is called a* tree, *if*

- *for every state v, there is at most one state u, and at most one action a such that $(u, v) \in E_a$,*
- *there is exactly one state r, called the* root *of the tree, for which there is no state having a transition to r, and*
- *every state is reachable from the root r.*

In almost all areas of logic, there are notions telling us whether two structures are "the same" with respect to the requirements needed in the particular area. For first-order model theory, this is usually the isomorphism of structures. Sometimes a weaker notion is used, the *back and forth equivalence*, which corresponds to elementary equivalence.

As mentioned above, transition systems are used to model the behaviour of processes for the purpose of verification. In this context, isomorphism and elementary equivalence are far too strong to capture behavioural equivalence. Reconsider Example 9.2 and Figure 9.1. There are two distinct nodes in the transition system with label just P_1. This corresponds to the situation where User 1 is printing a file and there is no request from User 2. From both nodes the *idle* state and a state labelled by R_2 can be reached in one step. And, again, from the two nodes labelled R_2, states with the same labelling can be reached on both sides. In the server process modelled by this transition system, the two nodes labelled by P_1 correspond to the same state of the process and are in this sense equivalent. The transition system where one of these two nodes is removed and all edges going into it are redirected to the other node, should fall into the same equivalence class as the given one. However, the two systems obtained in this way are neither isomorphic nor elementary equivalent.

Therefore, a weaker notion of equivalence between transition systems, or between nodes of transition systems is used, the so called *bisimulation*.

9.2 Bisimulation

Bisimulation is a notion of behavioural equivalence for transition systems and no reasonable modal logic can distinguish between two systems that are bisimulation equivalent.

9.4 Definition (Bisimulation). *Given two transition systems $\mathcal{K}, v$ and $\mathcal{K}', v'$, with distinguished states v and v' respectively, we say that $\mathcal{K}, v$ is* bisimulation equivalent*, or shorter* bisimilar*, to $\mathcal{K}', v'$, in terms $\mathcal{K}, v \sim \mathcal{K}', v'$, if there is a relation $Z \subseteq V \times V'$ between the states of $\mathcal{K}$ and the states of $\mathcal{K}'$ such that*

1. $(v, v') \in Z$,
2. *for every atomic proposition $p \in \mathcal{P}$ and every pair $(u, u') \in Z$, $u \in p^{\mathcal{K}}$ if, and only if, $u' \in p^{\mathcal{K}'}$,*
3. *for every $(u, u') \in Z$, and every $w \in V$ such that $(u, w) \in E_a^{\mathcal{K}}$, there is a $w' \in V'$ with $(u', w') \in E_a^{\mathcal{K}'}$ and $(w, w') \in Z$, and conversely*
4. *for every $(u, u') \in Z$, and every $w' \in V'$ such that $(u', w') \in E_a^{\mathcal{K}'}$, there is a $w \in V$ with $(u, w) \in E_a^{\mathcal{K}}$ and $(w, w') \in Z$.*

A class $\mathcal{C}$ of transition systems is closed under bisimulation, *if for all $\mathcal{K}, v \sim \mathcal{K}', v'$, whenever $\mathcal{K}, v \in \mathcal{C}$ then also $\mathcal{K}', v' \in \mathcal{C}$.*

As an example, consider again the transition system in Figure 9.1. It is easily seen, that the two nodes labelled P_1 are bisimilar. In fact, any two nodes carrying the same label are bisimilar.

By definition, bisimulation only "looks forward" in a transition system, i.e. whether two nodes are bisimilar depends only on the part of the transition system reachable from one of the nodes. For, if there is a bisimulation Z between $\mathcal{K}, v$ and $\mathcal{K}', v'$, then the relation Z' obtained from Z by removing all pairs (u, w) from Z where u or w is not reachable from v and v' respectively, still satisfies the conditions in Definition 9.4. This also meets our intuition of bisimulation as notion of behavioural equivalence between processes. Processes that only differ in states that they will never be in should still be considered equivalent. For instance, the nodes v and v' in the transition systems $\mathcal{K}$ and $\mathcal{K}'$ defined as

$$\begin{tikzcd}[column sep=small] & & v \bullet Q \arrow[dl] \arrow[dr] & \\ \mathcal{K}, v : & P \bullet & & \bullet P \end{tikzcd} \qquad \begin{tikzcd} & \bullet Q \arrow[d] \\ \mathcal{K}', v' : & v' \bullet Q \arrow[d] \\ & \bullet P \end{tikzcd}$$

are bisimilar.

The next proposition concerns a property of bisimulations that is of particular importance to the study of modal logics.

9.5 Proposition. *For every transition system $\mathcal{K}$, and any state v in $\mathcal{K}$, there is a tree $\mathcal{T}$ with root r – called the* unravelling *of $\mathcal{K}, v$ – such that $\mathcal{K}, v \sim \mathcal{T}, r$.*

One consequence of this is that any logic that respects bisimulation, i.e. cannot distinguish between bisimilar structures, has the tree model property: Every satisfiable formula of this logic has a model that is a tree. This is a useful and important property of such logics, since many problems allow more efficient algorithms for trees than for arbitrary transition systems.

In general, a transition system may have many pairwise bisimilar nodes. However, there are situations where it is convenient to work with systems in which no two different nodes are bisimilar. It can easily be shown that every transition system $\mathcal{K}$ is bisimilar to such a structure, which is called the quotient $\mathcal{K}_{/\sim}$ of $\mathcal{K}$ under bisimulation.

9.6 Definition (Quotient under bisimulation). *Let $\mathcal{K} := (V, (E_a)_{a \in \mathcal{A}}, (p)_{p \in \mathcal{P}})$ be a transition system over a set $\mathcal{A}$ of actions and a set $\mathcal{P}$ of proposition symbols. For every node $v \in V$ let $[v] := \{v' : \mathcal{K}, v \sim \mathcal{K}, v'\}$ be the set of nodes bisimilar to v.*

The quotient $\mathcal{K}_{/\sim}$ of $\mathcal{K}$ under bisimulation *is defined as the structure $\mathcal{K}_{/\sim} := (V_{/\sim}, (E_{a/\sim})_{a \in \mathcal{A}}, (p_{/\sim})_{p \in \mathcal{P}})$, where*

- $V_{/\sim} := \{[v] : v \in V\}$,
- $E_{a/\sim} := \{([u], [v]) : (u, v) \in E_a^{\mathcal{K}}\}$ *for every $a \in \mathcal{A}$, and*
- $p_{/\sim} := \{[v] : v \in p^{\mathcal{K}}\}$ *for every $p \in \mathcal{P}$.*

9.7 Proposition. *Every transition system $\mathcal{K}, v$ is bisimilar to its quotient $\mathcal{K}_{/\sim}, [v]$ under bisimulation.*

9.3 Modal Logic

In this section we introduce propositional modal logic which will be the basis for various fixed-point logics presented in later chapters. All these logics respect bisimulations and therefore have the tree model property.

9.8 Definition (Syntax of modal logic). *Fix a set $\mathcal{A}$ of actions and a set $\mathcal{P}$ of proposition symbols. The formulae of* modal logic (ML) *are inductively defined as follows.*

- *false, true* $\in$ ML *and for every proposition symbol $p \in \mathcal{P}$, p is a formula in* ML.
- *If $\varphi, \psi \in$ ML, then also $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $\neg\psi$ are formulae in* ML.
- *If φ is a formula in* ML *and $a \in \mathcal{A}$ is an action, then $\langle a\rangle\psi$ and $[a]\psi$ are also formulae in* ML. *If there is only one action in $\mathcal{A}$, we simply write $\Box$ and $\Diamond$ for $[a]$ and $\langle a\rangle$, respectively.*

The next definition presents the semantics of modal logic.

9.9 Definition (Semantics of modal logic). *The formulae of* ML *are evaluated on transition systems at a particular state. Given a formula ψ and a transition system $\mathcal{K}$ with state v, we write $\mathcal{K}, v \models \psi$ to express that the formula ψ holds in $\mathcal{K}$ at state v. We also write $[\![\psi]\!]^{\mathcal{K}}$ to denote the set of states v, where $\mathcal{K}, v \models \psi$.*

- *In the case of atomic propositions, i.e. where $\psi := p$, we define $[\![p]\!]^{\mathcal{K}} := p^{\mathcal{K}}$.*
- *Boolean connectives are treated in the natural way, e.g. $[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{K}} = [\![\varphi_1]\!]^{\mathcal{K}} \cup [\![\varphi_2]\!]^{\mathcal{K}}$.*
- *Finally, to define the semantics of the modal operators, we put*

$$[\![\langle a\rangle\psi]\!]^{\mathcal{K}} := \left\{ v : \begin{array}{c} \textit{there exists a state } w \textit{ such that} \\ (v,w) \in E_a \textit{ and } w \in [\![\psi]\!]^{\mathcal{K}} \end{array} \right\}$$

and

$$[\![[a]\psi]\!]^{\mathcal{K}} := \left\{ v : \begin{array}{c} \textit{for all } w \textit{ such that } (v,w) \in E_a, \\ \textit{we have } w \in [\![\psi]\!]^{\mathcal{K}} \end{array} \right\}.$$

$\langle a\rangle$ and $[a]$ can be seen as existential and universal quantifiers 'along a-transitions'.

As an example for a formula in modal logic consider

$$\varphi := idle \wedge \Diamond(R_1 \wedge \Diamond(P_1 \wedge \Diamond idle))$$

expressing that, in the print server modelled by the transition system in Figure 9.1, it is possible that the server is idle, User 1 requests the printer, prints his file and the server becomes idle again.

As mentioned above, modal logic respects bisimulation in the sense that bisimilar structures are indistinguishable by ML-formulae. Logics respecting bisimulations in this way are called *bisimulation invariant.*

9.10 Definition. *A logic $\mathcal{L}$ is* bisimulation invariant*, if for every formula $\varphi \in \mathcal{L}$ and any pair $\mathcal{K}, v$ and $\mathcal{K}', v'$ of transition systems, $\mathcal{K}, v \models \varphi$ if, and only if, $\mathcal{K}', v' \models \varphi$.*

The next proposition relates modal logic to bisimulation.

9.11 Proposition. *Modal logic is invariant under bisimulation. The converse is not true, i.e. there are transition systems $\mathcal{K}, v$ and $\mathcal{K}', v'$ which are not bisimilar but have the same* ML*-theory.*

In common terminology, logics that are invariant under bisimulation are called modal logics. There is an extension of modal logic – *infinitary modal logic* – which precisely characterises bisimulation.

9.12 Definition (Infinitary modal logic). Infinitary modal logic (ML^{∞}) *is defined as the extension of* ML *by the following formula building rule: If Φ is a set of formulae in ML^{∞} then $\bigwedge \Phi$ and $\bigvee \Phi$ are also formulae in ML^{∞} with the natural semantics.*

In [vB83], van Benthem proved that equivalence in infinitary modal logic corresponds to bisimulation equivalence.

9.13 Proposition. *Two transition systems $\mathcal{K}, v$ and $\mathcal{K}', v'$ are bisimilar if, and only if, they satisfy the same formulae in ML^{∞}.*

We proceed by listing some properties that are of interest for algorithmic and complexity issues.

9.14 Definition. *Let $\mathcal{L}$ be a logic interpreted over transition systems.*

- *$\mathcal{L}$ has the* tree model property*, if every satisfiable formula in $\mathcal{L}$ has a tree model.*
- *$\mathcal{L}$ has the* finite model property*, if every satisfiable formula has a finite model.*

As every transition system is bisimilar to a tree, any logic that is invariant under bisimulation automatically has the tree model property.

9.15 Proposition. *Modal logic* ML *has the tree model property and the finite model property. Even more, it can be shown that every satisfiable* ML*-formula has a model that is a finite tree.*

As mentioned above, modal logic is invariant under bisimulation. It is also clear that ML is contained in first-order logic. It has been shown by van Benthem [vB76], that modal logic is not only contained in FO, but that every property of transition systems that is closed under bisimulation and definable in first-order logic is also definable in modal logic.

9.16 Definition. *Let $\mathcal{L}, \mathcal{L}'$ be logics. $\mathcal{L}$ is the* bisimulation invariant fragment *of $\mathcal{L}'$ if every formula of $\mathcal{L}$ is equivalent to a formula of $\mathcal{L}'$ and further, every class $\mathcal{C}$ of transition systems that is closed under bisimulation and definable in $\mathcal{L}'$ is also definable in $\mathcal{L}$.*

The following proposition is due to van Benthem [vB76]. The finite model theory version of it is due to Rosen [Ros97].

9.17 Proposition. *Modal logic is the bisimulation invariant fragment of first-order logic.*

Chapter 10

The Modal μ-Calculus

In this chapter we consider the extension of modal logic by an operator to form least fixed points. This logic, the *modal μ-calculus* (L_μ), extends ML in the same way as LFP extends FO.

10.1 Definition (Modal μ-calculus). *The* modal μ-calculus (L_μ) *extends basic modal logic by the following rule. If $\varphi(X)$ is a formula of L_μ, and positive in its free propositional variable X, then $\mu X.\varphi$ and $\nu X.\varphi$ are also formulae of L_μ.*

On any transition system $\mathcal{K}, v$ with universe V providing an interpretation of all propositional variables in φ except for X, the formula φ induces an operator F_φ taking a set $X \subseteq V$ of nodes to the set $[\![\varphi]\!]^{\mathcal{K},X} := \{u \in V : (\mathcal{K}, X), u \models \varphi\}$.

As φ is positive in X, this operator is monotone and has a least fixed point $\mathbf{lfp}(F_\varphi)$ *and a greatest fixed point* $\mathbf{gfp}(F_\varphi)$. *Now, define $[\![\mu X.\varphi]\!]^{\mathcal{K}} := \mathbf{lfp}(F_\varphi)$ and $[\![\nu X.\varphi]\!]^{\mathcal{K}} := \mathbf{gfp}(F_\varphi)$.*

Clearly, the Knaster-Tarski Theorem (Theorem 3.5) applies to L_μ and thus the least fixed point can be built up inductively by the sequence

$$\begin{aligned} X^0 &:= \varnothing \\ X^{\alpha+1} &:= F_\varphi(X^\alpha) \\ X^\lambda &:= \bigcup_{\xi<\lambda} X^\xi \qquad \text{for limit ordinals } \lambda \end{aligned}$$

of stages. The inductive definition of greatest fixed points is analogous.

As mentioned in Section 3.1, least and greatest fixed points are dual to each other. Consequently, for any formula φ, the formula $\nu X.\varphi$ is equivalent to $\neg\mu X.\neg\varphi(\neg X)$, where $\varphi(\neg X)$ denotes the formula obtained from φ by replacing all occurrences of X with $\neg X$.

10.2 Example. *On any transition system $\mathcal{K}$ with universe V, the formula $\varphi := \mu X.\Box X$ defines the well-founded part of $\mathcal{K}$, i.e. the set of those elements*

$v \in V$ from which no infinite path starts. For this, note that in stage α of the induction on $\Box X$, X^α contains exactly the elements of height less than α.

We continue Example 9.2 and formalise some specifications in the modal μ-calculus.

10.3 Example. *Consider again the print server in Example 9.2.*

- *The formula* $\text{idle} \wedge \Box(\text{reachable}(\text{idle}))$*, where*

$$\textit{reachable}(\varphi) := \mu X.\varphi \vee \Diamond X,$$

  expresses that if the server is idle then whatever happens in the next step it is possible for the server to become idle again. Here, $\text{reachable}(\varphi)$ *is true for all nodes v from which a node where φ holds can be reached.*

- *The formula* $\bigwedge_{i=1,2} \textit{everywhere}(R_i \to \mu X.P_i \vee \Box X)$*, where*

$$\textit{everywhere}(\varphi) := \nu Y.\varphi \wedge \Box Y,$$

  expresses that whenever a user sends a printer request, he eventually has his file printed. Here, the formula $\textit{everywhere}(\varphi)$ is true at those nodes v such that every path starting at v consists exclusively of nodes where φ holds. The sub-formula $\mu X.P_i \vee \Box X$ becomes true at those nodes v, where every path starting at v eventually reaches a node where P_i holds.

- *Finally, we combine the formulae* everywhere *and* reachable *and extend the specification in the first item. The formula*

$$\text{idle} \wedge \textit{everywhere}(\textit{reachable}(\text{idle}))$$

  expresses that from every state in which the server can get, the idle state is reachable, i.e. no user action whatsoever can crash the server.

It is often useful to consider L_μ-formulae in a certain normal form, called *guarded normal form.*

10.1 Definition. *A μ-calculus formula is* guarded*, if all propositional variables that are bound by and thus occur in the scope of a fixed-point operator are also in the scope of a next-modality that is itself in the scope of the fixed-point operator.*

It was shown in [KVW00] that every μ-calculus formula is equivalent to a guarded formula.

Simultaneous fixed points. As for least fixed-point logic, we allow simultaneous inductions in L_μ. Simultaneous L_μ-formulae are of the form $(\mu X_i : S)$, where S is a system

$$S := \begin{cases} X_1 & \leftarrow & \varphi_1(X_1, \dots, X_k) \\ & \vdots & \\ X_k & \leftarrow & \varphi_k(X_1, \dots, X_k) \end{cases}$$

of rules such that the φ_i are positive in all X_j. On a transition system $\mathcal{K}$ with universe V, the system S of formulae induces an operator F_S : $\mathrm{Pow}(V)^k \to \mathrm{Pow}(V)^k$ taking a tuple $\overline{X} := (X_1, \dots, X_k) \in \mathrm{Pow}(V)^k$ to the tuple $(F_{\varphi_1}(\overline{X}), \dots, F_{\varphi_k}(\overline{X}))$, where F_{φ_i} is the operator induced by φ_i defined as

$$\begin{array}{rccl} F_{\varphi_i} & : & \mathrm{Pow}(V)^k & \longrightarrow & \mathrm{Pow}(V) \\ & & \overline{X} & \longmapsto & [\![\varphi_i]\!]^{(\mathcal{K},\overline{X})}. \end{array}$$

Since the φ_i are positive in the variables $\overline{X}$, this operator is monotone and has a least fixed point $(X_1^\infty, \dots, X_k^\infty)$. We put $[\![(\mu X_i : S)]\!]^{(\mathcal{K},\overline{X})} := X_i^\infty$ and likewise for $(\nu X_i : S)$.

In Lemma 3.18, we proved that simultaneous inductions in least fixed-point logic can be eliminated in favour of simple inductions. The proof of this lemma did not use any property special to first-order or least fixed-point logic and can literally be applied to the μ-calculus.

10.4 Theorem. *In the modal μ-calculus, simultaneous inductions can be eliminated, i.e. L_μ has the same expressive power with and without simultaneous inductions.*

We now state, without proofs, a number of results about the modal μ-calculus.

10.5 Proposition.

- *On classes of structures of bounded cardinality, $L_\mu \subseteq \mathrm{ML}^\infty$. Thus, L_μ is invariant under bisimulation. (See Lemma 11.2 below.)*
- *As $L_\mu \subseteq \mathrm{ML}^\infty$ it has the tree model property. By a result of Kozen [Koz88], it also has the finite model property but, contrary to* ML*, not every satisfiable L_μ-formula has a finite tree model.*

The next proposition is due to Bradfield [Bra98a, Bra98b].

10.6 Proposition. *The alternation hierarchy for the modal μ-calculus is strict, i.e. increasing number of alternations between least and greatest fixed points result in greater expressive power.*

We now turn towards the complexity of the algorithmic problems connected with L_μ.

10.7 Proposition.

- *The satisfiability problem for the modal μ-calculus is* EXPTIME*-complete [EJ88].*
- *The model checking problem for L_μ is in* NP $\cap$ CO-NP*. In fact, the result can even be strengthened to* UP $\cap$ CO-UP *[Jur98].*

The proposition states an UP $\cap$ CO-UP bound for the complexity of the model checking problem for L_μ. However, this bound is only an upper bound and it is conceivable that model checking can be done in polynomial time. Finally, we relate L_μ to the monadic fragments of least fixed-point and second-order logic.

10.8 Proposition.

- *Clearly, $L_\mu \subseteq$* M-LFP $\subseteq$ MSO*. The converse is also true, in the sense that every* MSO*-definable class of transition systems that is closed under bisimulation is definable in L_μ. Thus, L_μ is the bisimulation invariant fragment of* MSO *[JW96].*
- *As a consequence of the Janin-Walukiewicz theorem, the class of languages definable in L_μ is precisely the class of regular languages.*

As the proposition states, the modal μ-calculus is the bisimulation invariant fragment of monadic second-order logic. However, the method used to prove this relies heavily on the fact that every transition system is bisimilar to a – potentially infinite – tree. Whether the theorem also holds in the restriction to finite models is one of the major open problems in this area.

The Higher Dimensional μ-Calculus. We briefly present the higher dimensional μ-calculus introduced by Martin Otto [Ott99]. As noted above, the modal μ-calculus characterises precisely the class of regular languages. It follows immediately, that it falls short of expressing all polynomial time decidable languages. On reason for its limited power is the restriction to monadic inductions, i.e. fixed-point inductions defining subsets of the universe.

In [Ott99], Otto introduced an extension of L_μ in which fixed-point inductions over relations of higher arity are possible. We refrain from giving a formal definition of this logic as we will not use individual formulae. See [Ott99] for details. The key property of L_μ^ω that makes it interesting to us is that all polynomial time decidable classes of finite transition systems that are closed under bisimulation are definable in it. As the evaluation problem for L_μ^ω is itself in polynomial time, this gives a precise characterisation of bisimulation invariant PTIME.

10.9 Theorem. *L_μ^ω captures bisimulation invariant* PTIME*, i.e. every class of finite transition systems that is definable in L_μ^ω is decidable in* PTIME *and, conversely, every class of transition systems that is closed under bisimulation and decidable in* PTIME *can be defined by a formula of L_μ^ω.*

Chapter 11

The Modal Iteration Calculus

In the previous chapter, we introduced the modal μ-calculus, the least fixed-point extension of ML. We proceed by studying the extension of modal logic by inflationary fixed points. The resulting logic, the *modal iteration calculus* (MIC), is propositional modal logic (ML), augmented with simultaneous inflationary fixed points.

11.1 Definition (Syntax and semantics of MIC). *The* modal iteration calculus (MIC) *extends propositional modal logic by the following rule. If $\varphi_1, \ldots, \varphi_k$ are formulae of* MIC, *and $X_1, \ldots, X_k$ are propositional variables, then*

$$S := \begin{cases} X_1 & \leftarrow & \varphi_1(X_1, \ldots, X_k) \\ & \vdots & \\ X_k & \leftarrow & \varphi_k(X_1, \ldots, X_k) \end{cases}$$

is a system *of rules, and* (**ifp** $X_i : S$) *is a formula of* MIC. *If S consists of a single rule $X \leftarrow \varphi$ we simplify the notation and write* (**ifp** $X \leftarrow \varphi$) *instead of* (**ifp** $X : X \leftarrow \varphi$). *Let* 1MIC *denote the fragment of* MIC *without simultaneous inductions, i.e. where inductions are allowed only over systems with one rule only.*

On every transition system $\mathcal{K}$, the system S defines, for each ordinal α, a tuple $\overline{X}^\alpha := (X_1^\alpha, \ldots, X_k^\alpha)$ of sets of states, via the following inflationary induction (for $i = 1, \ldots, k$).

$$\begin{aligned} X_i^0 &:= \varnothing, \\ X_i^{\alpha+1} &:= X_i^\alpha \cup [\![\varphi_i]\!]^{(\mathcal{K}, \overline{X}^\alpha)}, \\ X_i^\lambda &:= \bigcup_{\xi<\lambda} X_i^\xi \text{ for limit ordinals } \lambda. \end{aligned}$$

$(X_1^\alpha, \dots, X_k^\alpha)$ is called the stage α of the inflationary induction of S on $\mathcal{K}$. As the stages are increasing, this induction reaches a fixed point $(X_1^\infty, \dots, X_k^\infty)$. Now we put $[\![(\mathbf{ifp} X_i : S)]\!]^{\mathcal{K}} := X_i^\infty$.

Various examples of MIC-formulae will be given below. The following lemma relates MIC to the modal logics introduced so far.

11.2 Lemma. *$L_\mu \subseteq$ MIC. Further, MIC $\subseteq$ ML$^\infty$ on every class of structures of bounded cardinality.*

Proof. Clearly, if X occurs only positively in ψ, then $\mu X.\psi \equiv \mathbf{ifp}\ X \leftarrow \psi$. Hence $L_\mu \subseteq$ MIC. Towards establishing the containment of MIC in ML$^\infty$, let $(\mathbf{ifp}\ X : \varphi(X))$ be a formula in MIC. To simplify notation, we only consider simple inductions. The case for simultaneous inductions is analogous.

For every ordinal α, we inductively define a formula $\psi^\alpha \in$ ML$^\infty$ which, over any transition system, defines the stage α of the induction on φ.

$$\begin{array}{lll} \psi^0 & := \mathit{false}, & \\ \psi^{\alpha+1} & := \psi^\alpha \vee \varphi(X/\psi^\alpha), & \\ \psi^\lambda & := \bigvee_{\xi<\lambda} \psi^\xi & \text{for limit ordinals } \lambda. \end{array}$$

Here, $\varphi(X/\psi^\alpha)$ is obtained from φ by replacing every occurrence of X in φ by ψ^α. As closure ordinals are bounded on classes of structures of bounded cardinality, the claim follows immediately. □

A consequence of the lemma is that MIC is indeed a modal logic, i.e. invariant under bisimulation.

11.3 Corollary. MIC *is invariant under bisimulation and has the tree model property.*

Note that on classes of structures of unbounded cardinality, L_μ and MIC are not contained in ML$^\infty$. For instance, we have shown above that well-foundedness is expressed by the L_μ-formula $\mu X.\Box X$, but is known not to be expressible in ML$^\infty$.

We now demonstrate that MIC is strictly more expressive than L_μ. Recall that every formula of L_μ can be translated into a formula of monadic second-order logic (MSO). Moreover, it is a well-known classical result that the only languages of finite words expressible in MSO are the regular languages [Büc60]. Here, finite words w are represented by transition systems as follows.

11.4 Definition. *Let Σ be a non-empty finite alphabet. For every word $w \in \Sigma^*$, define a transition system $\mathcal{T}_w := (V := \{0, \dots, |w|-1\}, E, (a)_{a\in\Sigma})$ with root 0, where E is the successor relation truncated at $|w|-1$ and $a^{\mathcal{T}_w}$ contains all $n \in V$ such that at position n, w contains the letter a.*

We aim at showing that there are languages definable in MIC which are not regular. For this, we first introduce some auxiliary formulae.

11.5 Proposition. *Let φ be a formula in* MIC *and $\mathcal{K}$ be a transition system with universe V.*

- *The formula everywhere$(\varphi) := \neg(\textbf{ifp } X \leftarrow \neg\varphi \vee \Diamond X)$ is true at a node $v \in V$ if φ holds at every node reachable from it.*
- *The formula somewhere$(\varphi) := (\textbf{ifp } X \leftarrow \varphi \vee \Diamond X)$ is true at a node $v \in V$ if there is some node u reachable from v where φ holds true.*
- *Finally, the formula nowhere$(\varphi) := \neg$somewhere(φ) is true at a node v if there is no node reachable from it where φ holds true.*

The proposition follows easily by induction on the stages. The formulae introduced in the proposition will frequently be used in the sequel. At various places, they will be used with a formula φ which itself contains a fixed-point operator binding X. Without further notice, we assume that in these cases the fixed-point variables are renamed appropriately. We are now ready to prove that MIC is strictly more expressive than L_μ.

11.6 Proposition. *There is a language that is expressible in* MIC *but not in* MSO.

Proof. We claim that the language $L := \{a^n b^m : n \leq m\}$ is definable in MIC. As it is not regular it is not definable in monadic second-order logic. Towards establishing the claim, we first consider the formula

$$\pi(X) = (\textbf{ifp } Y \leftarrow \Diamond(b \wedge \neg X) \vee \Diamond(a \wedge X \wedge Y))$$

which actually is equivalent to a L_μ-formula. On every word $w := w_0 \cdots w_{n-1}$ in $\{a, b\}^*$ and $X \subseteq \{0, \ldots, n-1\}$, the formula is true if the word w starts with a (possibly empty) a-sequence inside X followed by a b outside X. Now the formula $(\textbf{ifp } X \leftarrow (a \wedge \pi(X)) \vee (b \wedge \Box X))$ defines inside $a^* b^*$ the language L. As $a^* b^*$ is definable in L_μ we can take the conjunction of this definition with the above formula to obtain a definition of L which works on all words in $\{a, b\}^*$. $\Box$

The following corollary is an immediate consequence of the proposition.

11.7 Corollary. MIC *is strictly more expressive than L_μ, i.e. $L_\mu \subsetneq$* MIC.

In the previous section, we mentioned that, as for LFP, the alternation hierarchy for the modal μ-calculus is strict whereas it was shown in Part I that for inflationary fixed-point logic the alternation hierarchy collapses. We show now that this also holds true for the modal iteration calculus.

11.8 Theorem. *Every formula in* MIC *is equivalent to a formula where negation occurs only in front of the atoms.*

Proof. Let $\varphi := \neg(\mathbf{ifp}\ X_i : S)$ be any formula in MIC. We claim that φ is equivalent to the formula $\psi := (\mathbf{ifp}\ Y : S')$, where S' is obtained from S by adding the rule $Y \leftarrow \mathit{everywhere}(\bigwedge_{i=1}^{k}(\varphi_i \rightarrow X_i)) \wedge \neg X_i$. Here, the $X_1, \dots, X_k$ are all fixed-point variables for which a rule exists in S. Towards establishing the claim, consider the induction stages induced by S' on a structure $\mathcal{K}$. Obviously, the stages of the variables $X_1, \dots, X_k$ are the same as in the induction on S. Now let v be a node in $\mathcal{K}$. If, at some stage α, on all nodes reachable from v the conjunct $\bigwedge_{i=1}^{k}(\varphi_i \rightarrow X_i)$ becomes true, then this implies that the fixed point of the induction on S has been reached in the part of $\mathcal{K}$ reachable from v. Therefore, v is added to Y if, and only if, v does not occur in the fixed point of X_i. This proves the claim. □

We now investigate on the algorithmic properties of MIC. We consider the satisfiability problem first and proceed then to the model checking complexity.

11.1 The Satisfiability Problem for MIC

In this section we prove that the satisfiability problem for MIC is undecidable in a very strong sense. Given that MIC is invariant under bisimulation, we can restrict attention to trees. In particular, we will only consider well-founded trees. Recall from Definition 9.3 above that the *height* $h(v)$ of a node v in a well-founded tree $\mathcal{T}$ was defined as the least strict upper bound of the heights of its children. For any node v in a tree $\mathcal{T}$, we write $\mathcal{T}(v)$ for the subtree of $\mathcal{T}$ with root v. We first show that the nodes of finite height and the nodes of height ω are definable in MIC.

11.9 Lemma. *Let S be the system*

$$\begin{aligned} X &\leftarrow \Box \mathit{false} \vee (\Box X \wedge \Diamond \neg Y) \\ Y &\leftarrow X. \end{aligned}$$

On every tree $\mathcal{T}$, $[\![\mathbf{ifp}\ X : S]\!]^{\mathcal{T}} = [\![\mathbf{ifp}\ Y : S]\!]^{\mathcal{T}} = \{v : h(v) < \omega\}$.

Proof. By induction we see that for each $i < \omega$, $X^i = \{v : h(v) < i\}$ and $Y^i = X^{i-1} = \{v : h(v) < i-1\}$. As a consequence, $X^\omega = Y^\omega = \{v : h(v) < \omega\}$. One further iteration shows that $X^{\omega+1} = Y^{\omega+1} = X^\omega$. □

With the system S exhibited in Lemma 11.9 we obtain the formulae

$$\text{finite-height} := (\mathbf{ifp}\ X : S)$$

and

$$\omega\text{-height} := \neg\text{finite-height} \wedge \Box\text{finite-height}$$

which define, respectively, the nodes of finite height and the nodes of height ω. Note that ω-height is a satisfiable formula all of whose models are infinite. This immediately implies the following proposition.

11.10 Proposition. MIC *does not have the finite model property.*

We next show that the satisfiability problem for MIC is undecidable. In fact MIC interprets full arithmetic on the heights of nodes. To prove this, we first define some auxiliary formulae that will be used frequently throughout this chapter. We always assume that the underlying structure is a well-founded tree.

- Recall from above the formula *everywhere*(φ). The formula is true at a node v if φ holds at all nodes of the subtree $\mathcal{T}(v)$.
- Dually, the formula *somewhere*(φ) states that φ holds somewhere in the subtree of the current node v.
- We say that a set X (in a tree $\mathcal{T}$) *encodes* the ordinal α if $X = \{v : h(v) < \alpha\}$. Let *ordinal*$(X)$ be the conjunction of the formula *everywhere*$(X \rightarrow \Box X)$ with
$$\neg\textbf{ifp}\, Z : \begin{cases} Y & \leftarrow \quad \Box Y \\ Z & \leftarrow \quad \mathit{somewhere}(\neg Y \land \Box Y \land X) \land \\ & \qquad \mathit{somewhere}(\neg Y \land \Box Y \land \neg X). \end{cases}$$
It expresses that X encodes some ordinal. Indeed *everywhere*$(X \rightarrow \Box X)$ says that with each node $v \in X$, the entire subtree rooted at v is contained in X. The second conjunct performs an inflationary induction which, at each stage $\beta + 1$, adds to Y all nodes of height β and adds to Z all nodes u of height greater than β just in case that both, X and its complement, contain nodes from $\mathcal{T}_u$ of height β. Hence, at the end of the induction the root of the tree will *not* be contained in Z if, and only if, X does not distinguish between nodes of the same height. Together the two conjuncts imply that X contains all nodes up to some height.
- The formula
$$\mathit{number}(X) := \mathit{ordinal}(X) \land \mathit{somewhere}(\text{finite-height} \land \neg X)$$
says that X encodes a finite number $n < \omega$ (inside a tree of height greater than n).

We now show that addition and multiplication of natural numbers encoded by sets of nodes as described above can be defined in MIC.

11.11 Lemma. *Let $\mathcal{T}$ be a well-founded tree of height ω. There exist formulae* $\text{plus}(S,T)$ *and* $\text{times}(S,T)$ *of* MIC *such that, whenever the sets S and T encode in $\mathcal{T}$ the natural numbers s and t, then $[\![\text{plus}(S,T)]\!]^{\mathcal{T}}$ encodes $s+t$, and $[\![\text{times}(S,T)]\!]^{\mathcal{T}}$ encodes $s \cdot t$.*

Proof. Define the formula $plus(S,T)$ as

$$\text{plus}(S,T) := \textbf{ifp } Y : \left\{ \begin{array}{lll} X & \leftarrow & \Box X \\ Y & \leftarrow & S \vee (\Box Y \wedge \textit{somewhere}(X) \wedge \\ & & \qquad\qquad \textit{everywhere}(X \to T)). \end{array} \right.$$

Obviously, at each stage n, we have $X^n = \{v : h(v) < n\}$. We claim that for each n, $Y^{n+1} = \{v : h(v) < s + \min(n,t)\}$. For $n = 0$ this is clear as the conjunct *somewhere*(X) prevents the Y-rule from being active at the first stage. For $n > 0$ the inclusion $X^n \subseteq T$ is true if, and only if, $n \leq t$. Hence, we have $Y^{n+1} = \{v : h(v) < s + n\}$ in the case that $n \leq t$ and $Y^{n+1} = Y^n = \cdots = Y^{t+1}$ otherwise. To express multiplication define

$$\text{times}(S,T) := \textbf{ifp } Y : \left\{ \begin{array}{lll} X & \leftarrow & \Box X \\ Y & \leftarrow & \text{plus}(Y,S) \wedge \textit{everywhere}(\Box X \to T). \end{array} \right.$$

We claim that $Y^n = \{v : h(v) < s \cdot \min(n,t)\}$. This is trivially true for $n = 0$. If it is true for $n < t$, then $Y^{n+1} = \{v : h(v) < sn + s\} = \{v : h(v) < s(n+1)\}$. Finally for $n \geq t$, $\Box X^n$ defines $\{v : h(v) < n + 1\}$ which is not contained in $T = \{v : h(v) < t\}$, hence $Y^{n+1} = Y^n = \cdots = Y^t$. $\Box$

An immediate consequence of the lemma is the following corollary.

11.12 Corollary. *For every polynomial $f(x_1,\dots,x_r)$ with coefficients in the natural numbers there exists a formula $\psi_f(X_1,\dots,X_r) \in$ MIC such that for every tree $\mathcal{T}$ of height ω and all sets $S_1,\dots,S_r$ encoding natural numbers $s_1,\dots,s_r < \omega$*

$$[\![\psi_f(S_1,\dots,S_r)]\!]^{\mathcal{T}} = \{v : h(v) < f(s_1,\dots,s_r)\}.$$

Proof. The proof is by induction on f.

- $\psi_0 := \textit{false}$.
- $\psi_1 := \Box \textit{false}$.
- $\psi_{x_i} := X_i$.
- $\psi_{f+g} :=$ plus$[S/\psi_f, T/\psi_g]$, i.e. the formula obtained by replacing in plus(S,T) the variables S and T by ψ_f and ψ_g respectively.
- $\psi_{f\cdot g} :=$ times$[S/\psi_f, T/\psi_g]$.

$\Box$

The corollary establishes the main step in the proof of the following theorem showing that for every first-order sentence in the language of arithmetic, there is a MIC-formula that is satisfiable if, and only if, the FO-sentence is true in the arithmetic.

11.13 Theorem. *For every first-order sentence ψ in the vocabulary $\{+, \cdot, 0, 1\}$, there exists a formula $\psi^* \in$ MIC such that ψ is true in the standard model $(\mathbb{N}, +, \cdot, 0, 1)$ of arithmetic if, and only if, ψ^* is satisfiable.*

Proof. The proof is by induction on the structure of the first-order formulae ψ. We have to show that for each such formula $\psi(y_1, \dots, y_r)$ there exists a MIC-formula $\psi^*(Y_1, \dots, Y_r)$ such that on rooted trees $\mathcal{T}, w$ of height ω and for all sets $S_1, \dots, S_r$ that encode numbers $s_1, \dots, s_r$ on $\mathcal{T}$ we have that $(\mathbb{N}, +, \cdot, 0, 1) \models \psi(s_1, \dots, s_r)$ if, and only if, $\mathcal{T}, w \models \psi^*(S_1, \dots, S_r)$.

- We have already seen that there exists a MIC-axiom ω-height axiomatising the models that are bisimilar to a tree of height ω.
- Further, we can express set equalities $X = Y$ by *everywhere*$(X \leftrightarrow Y)$ and we know how to represent polynomials by MIC-formulae. Thus, for every atomic first-order formula ψ over the arithmetic there is a MIC-formula that, for any given interpretation of the free variables, is true at the root of the tree $\mathcal{T}$ if, and only if, ψ is true in the arithmetic.
- The case of Boolean combinations is trivial.
- What remains is to translate quantifiers.

  Let ψ be a formula of the form $\psi(\overline{y}) := \exists x \varphi(x, \overline{y})$. By induction hypothesis, there is a formula $\varphi^*(X, \overline{Y})$ corresponding to $\varphi(x, \overline{y})$. Now consider

  $$\psi^*(\overline{Y}) := \textbf{ifp } Z : \left\{ \begin{array}{l} X \leftarrow \Box X \\ Z \leftarrow \varphi^*(X, \overline{Y}) \wedge \text{number}(X). \end{array} \right.$$

  In its induction stages, X enumerates all natural numbers, i.e. all sets $\{v : h(v) < n\}$ for some n. Now, the root w of $\mathcal{T}$ is added to Z if φ^* is satisfied for some value of X, i.e. for some natural number n encoded by X. Note that the conjunct number(X) is needed for the final stage of X, i.e. when X contains the whole tree and therefore does not encode a finite number anymore.

This finishes the proof of the theorem. □

The following corollary is now immediate.

11.14 Corollary. *The satisfiability problem for* MIC *is undecidable. In fact, it is not even in the arithmetical hierarchy. It is, however, in Σ^1_2, the second level of the analytical hierarchy.*

Containment of Sat(MIC) in Σ^1_2 is an immediate consequence of Corollary 3.41, where the corresponding bound was proved for IFP. The proof given above appears to rely crucially on the use of simultaneous inductions. Indeed, we will show below that the formula constructed in the proof of Lemma 11.9 cannot be expressed without simultaneous inductions (see Theorem 11.21). However, in Section 11.4 we prove that, nevertheless, first-order arithmetic can be reduced to the satisfiability problem for 1MIC.

11.2 The Model Checking Problem for MIC

Recall that the model checking problem for the μ-calculus is in UP$\cap$ co-UP (see Proposition 10.7), and is conjectured by some to be solvable in polynomial time. We show now that, provided PSPACE $\neq$ NP, MIC is algorithmically much more complicated.

First, observe that the naive bottom-up evaluation algorithm for MIC-formulae uses polynomial time with respect to the size of the input structure, and polynomial space (and exponential time) with respect to the length of the formula. Let $\mathcal{K}$ be a transition system with n nodes and m edges. The size $||\mathcal{K}||$ of appropriate encodings of $\mathcal{K}$ as an input for a model checking algorithm is $O(n+m)$. It is easily seen that the extension $[\![\varphi]\!]^{\mathcal{K}}$ of a formula $\varphi \in$ ML on a finite transition system $\mathcal{K}$ can be computed in time $O(|\varphi| \cdot ||\mathcal{K}||)$. Further, any inflationary induction **ifp** $X_i : [X_1 \leftarrow \varphi_1, \dots, X_k \leftarrow \varphi_k]$ reaches a fixed point on $\mathcal{K}$ after at most kn iterations. Hence, the bottom-up evaluation of a MIC-formula ψ on $\mathcal{K}$ with d nested simultaneous inflationary fixed points, each of width k, needs at most $O((kn)^d)$ basic evaluation steps. For each fixed point variable occurring in the formula, $2n$ bits of workspace are needed to record the current value and the last value of the induction. This gives the following complexity results.

11.15 Proposition. *Let ψ be a* MIC*-formula of nesting depth d with simultaneous inductions of width at most k. On any transition system $\mathcal{K}$ with n nodes, ψ can be evaluated in time $O((kn)^d \cdot |\psi| \cdot ||\mathcal{K}||)$ and space $O(|\psi| \cdot n)$.*

In terms of common complexity classes the results can be stated as follows.

11.16 Theorem. (1) *With respect to combined complexity, the model-checking problem for* MIC *on finite structures is in* PSPACE.

(2) *For any fixed formula $\psi \in$* MIC*, the model-checking problem for ψ on finite structures is solvable in polynomial time and linear space.*

We prove next that, contrary to the case of the μ-calculus, the complexity results obtained by this naive algorithm cannot be improved significantly.

11.17 Theorem. *There are transition systems $\mathcal{K}$, such that the model checking problem for* MIC *on $\mathcal{K}$ is* PSPACE*-complete, even for* MIC*-formulae without simultaneous inductions.*

Proof. The proof is by reduction from QBF (the evaluation problem for quantified Boolean formulae). For MIC with simultaneous inductions, this is trivial: Let $\mathcal{K}$ be the transition system consisting just of a single point v. With every quantified Boolean formula ψ we associate a MIC-formula ψ^* such that ψ is valid if, and only if, $\mathcal{K}, v \models \psi^*$. For this, we introduce a fixed-point variable for each propositional variable X in the QBF-formula ψ. The truth values of the variables in ψ are represented by the truth value of X on the node v in $\mathcal{K}$, i.e. an interpretation of a QBF-variable X by *true* corresponds to the interpretation of the fixed-point variable by $\{v\}$.

For the translation, we first eliminate the universal quantifiers in ψ in favour of negations and existential quantifiers. Then we inductively replace all sub-formulae $\varphi(\overline{Y}) := \exists X \vartheta(X, \overline{Y})$ by inflationary fixed points

$$\varphi^*(\overline{Y}) := \mathbf{ifp}\ Z : \begin{cases} X \leftarrow \mathit{true} \\ Z \leftarrow \vartheta^*(X, \overline{Y}). \end{cases}$$

We have to show that, for any interpretation of $\overline{Y}$, we have $(\mathcal{K}, \overline{Y}), v \models \varphi^*(\overline{Y})$ if, and only if, $\varphi(\overline{Y})$ is true. For this, consider the stages induced by φ^*. Clearly, $X^0 = \varnothing$ and $X^1 = \{v\}$. Hence, v is included in Z^1 if $\varphi(0, \overline{Y})$ holds, and in Z^2 if $\varphi(1, \overline{Y})$ is true. Thus, φ^* holds at v if, and only if, φ is true.

For MIC without simultaneous fixed points, the construction is somewhat more complicated. Let $\mathcal{K}$ be the transition system consisting of two points 0,1, the atomic proposition $p = \{1\}$ and the complete transition relation $\{0,1\} \times \{0,1\}$.

Let $\alpha(X) := \neg X \wedge (p \rightarrow \Diamond X)$. Further, let $\varphi[X/\alpha(X)]$ denote the formula obtained from φ by replacing every free occurrence of X by $\alpha(X)$. The transformation from QBF-formulae ψ to MIC-formulae ψ^* without simultaneous fixed points is inductively defined as follows.

(1) For $\psi := X$ we set $\psi^* := (p \wedge X) \vee (\neg p \wedge \Diamond X)$,

(2) $(\neg\psi)^* := \neg\psi^*$ and $(\psi \circ \varphi)^* := \psi^* \circ \varphi^*$ for $\circ \in \{\wedge, \vee\}$,

(3) for $\psi := \forall X \varphi$ we put $\psi^* := \Box(\mathbf{ifp}\ X \leftarrow \alpha(X) \wedge \varphi^*[X/\alpha(X)])$.

(4) Existential quantifiers are eliminated in favour of negation and universal quantification.

In this translation, the truth value *true* of a propositional variable is represented by any subset of $\{0,1\}$ that contains the node 1. We claim that for any QBF-formula ψ, any interpretation $\overline{Y}$ for the free variables of ψ represented by $\overline{Y}^*$, we have

$$[\![\psi^*]\!]^{(\mathcal{K}, \overline{Y}^*)} = \begin{cases} \{0,1\} & \text{if } \psi(\overline{Y}) \text{ is } \mathit{true} \\ \varnothing & \text{if } \psi(\overline{Y}) \text{ is } \mathit{false}. \end{cases}$$

The proof is by induction on the structure of ψ.

(1) For $\psi := X$ and for Boolean combinations of formulae the verification of the claim is straightforward.

(2) Let $\psi := \forall X \varphi(X, \overline{Y})$. Note that $[\![\alpha(X)]\!]^{(\mathcal{K},X)} = \{0\}$ for $X = \varnothing$ and $[\![\alpha(X)]\!]^{(\mathcal{K},X)} = \{1\}$ for $X = \{0\}$. Thus, the first induction step of $\vartheta := (\textbf{ifp}\, X \leftarrow \alpha(X) \wedge \varphi^*[X/\alpha(X)])$ produces $X^1 = \{0\} \cap [\![\varphi^*(0)]\!]^{(\mathcal{K},\overline{Y}^*)}$, which, by induction, is $\{0\}$ if $\varphi(0,\overline{Y})$ evaluates to true and $X^1 = \varnothing$ otherwise. If the value $X^1 = \{0\}$ was produced, then the second iteration step produces $X^2 = X^1 \cup (\{1\} \cap [\![\varphi^*(1)]\!]^{(\mathcal{K},\overline{Y}^*)})$. Thus,

$$[\![\psi^*]\!]^{(\mathcal{K},\overline{Y}^*)} = \begin{cases} \varnothing & \text{if } \neg\varphi(0,\overline{Y}) \\ \{0\} & \text{if } \varphi(0,\overline{Y}) \wedge \neg\varphi(1,\overline{Y}) \\ \{0,1\} & \text{if } \varphi(0,\overline{Y}) \wedge \varphi(1,\overline{Y}). \end{cases}$$

As required, in the first two cases the formula $\Box(\textbf{ifp}X \leftarrow \alpha(X) \wedge \varphi^*[X/\alpha(X)])$ is false at both states, and in the last case it is globally true.

Finally note that in both cases, i.e. whether simultaneous inductions are allowed or forbidden, the formula ψ^* can be computed in linear time from ψ. This immediately implies the theorem. □

As the theorem shows, the model checking complexity of MIC is much higher than for the μ-calculus and for practical applications in verification it might already be too complex.

We now turn towards establishing some expressibility results on the class of languages definable in MIC.

11.3 Languages and MIC

In Proposition 11.6 we already saw a language that was not regular but definable in MIC. In this section we show that not only non-regular languages but also languages that are not even context-free can be defined in MIC.

11.18 Theorem. *There is a language definable in* MIC *that is not context-free.*

Proof. Consider the language L over the alphabet $\{a, b, c, d\}$ defined as

$$L = \{cwdwd : w \in \{a, b\}^*\}.$$

Using pumping arguments it is easily verified that L is not a context-free language. To see that it is definable in MIC, first note that the formula

$$\begin{aligned} \alpha := {} & c \wedge \Box \mathit{nowhere}(c) \wedge \mathit{somewhere}(d \wedge \Box \mathit{somewhere}(d)) \wedge \\ & \mathit{everywhere}[(d \wedge \Box \mathit{somewhere}(d)) \rightarrow \Box \mathit{everywhere}(d \rightarrow \Box \mathit{nowhere}(d))] \end{aligned}$$

defines the set of strings $\{cxdyd : x, y \in \{a,b\}^*\}$. Now, the desired formula is the conjunction of α with the formula

$$\varphi := \neg \mathbf{ifp}\ X \leftarrow (\neg c \wedge (\Box X \vee d)) \vee (c \wedge \mathit{somewhere}(\psi))$$

where ψ is the formula

$$\begin{aligned}\neg X \wedge \Box X \wedge (&(b \wedge \mathit{somewhere}(a \wedge \Box X \wedge \neg X))\vee\\ &(a \wedge \mathit{somewhere}(b \wedge \Box X \wedge \neg X)) \vee\\ &(c \wedge \mathit{somewhere}((a \vee b) \wedge \Box X \wedge \neg X)) \vee\\ &((a \vee b) \wedge \mathit{somewhere}(d \wedge \Box X \wedge \neg X))).\end{aligned}$$

To see that this defines the language L, note that in evaluating the outer most fixed point in the definition of φ on a string $cxdyd$, with $x, y \in \{a,b\}^*$, the stage X^i contains the suffix of x and y, respectively, of length $n-1$. At the first stage, the sub-formula $(\Box X \vee d)$ ensures that the two letters labelled by d are included. Further, in the next stage, the last element of each of x and y is included, and inductively, $\Box X$ guarantees that the last element not yet included in X (in either x or y) is added.

Finally, the first letter in the word, labelled c, is included in X if, and only if, the formula ψ is true of some element, at some stage i. But, ψ is true at an element at stage i if, and only if, it is the $(i+1)$st element from the right in x and it is different from the corresponding element of y. Therefore, the first letter in the word is included in the fixed point if, and only if, the strings x and y differ at some point. Hence, as desired, $\varphi \wedge \alpha$ defines the language L. Note, that the first two lines in ψ deal with the cases where the words x and y differ in the letter at a certain position whereas the last two lines take care of x and y being of different length. $\Box$

We can also add the observation that the formula constructed in the proof of the theorem does not involve any simultaneous inductions, and therefore there are non-context-free languages definable in 1MIC.

To place the expressive power of MIC in the Chomsky hierarchy, we note that every language definable in MIC can be defined by a context-sensitive grammar. This follows from the observation made in Section 11.2 that any class of finite structures defined by a formula of MIC is decidable in linear space, and the result that all languages decidable by nondeterministic linear space machines are definable by context-sensitive grammars.

Finally, we mention another expressibility result for MIC that was proved in [DGK01].

11.19 Theorem. *Every language that can be decided in* $\textsc{Dtime}(\mathcal{O}(n))$ *can be defined in* MIC.

11.4 Simple Inductions

In the previous sections we have seen a couple of expressibility results for MIC. Many of these made use of simultaneous inductions and we remarked at several places that the formulae used there could not be defined in 1MIC. In this section we investigate on the relationship between simple and simultaneous inductions in the modal iteration calculus.

It is easy to see that the equivalence $\mu XY.(\psi, \varphi) \equiv \mu X.\psi(X, \mu Y.\varphi(X, Y))$ used to show that simultaneous inductions in the μ-calculus can be eliminated fails in both directions when we take inflationary instead of least fixed points. However, it still is conceivable that simultaneous inductions could be eliminated by more complicated techniques. We show here that this is not the case, i.e. simultaneous inflationary inductions provide more expressive power than simple ones. However, we have seen in Section 11.2 that the model checking problem was PSPACE-complete even for formulae without simultaneous inductions. In the second part of this section we show that, similar to MIC, the satisfiability problem for formulae without simultaneous inductions is still undecidable and not in the arithmetical hierarchy.

11.4.1 Simple vs. Simultaneous Inductions

In this section we show that MIC is strictly more expressive than 1MIC. For this, we identify a property that is definable in MIC with simultaneous inductions but not without. We first give an informal presentation of the problem. It is formally defined below.

Consider the family $(\mathcal{T}_\alpha)_{\alpha \leq \omega}$ of trees inductively defined as follows. Let $\mathcal{T}_0$ be a tree consisting of a single node only. For $\alpha > 0$, define $\mathcal{T}_\alpha$ as the tree consisting of a root v_α and for each $\beta < \alpha$ a successor v_β of v_α which is the root of a tree $\mathcal{T}_\beta$. See Figure 11.1.

By construction, the height of a tree $\mathcal{T}_\alpha$ is just α. We will prove below, that for every formula φ in 1MIC that is true at the root v_ω of $\mathcal{T}_\omega$, there is a finite number $n < \omega$ such that φ is also true at the root of every tree $\mathcal{T}_\beta$ with $\beta > n$. Thus, 1MIC cannot distinguish between trees of finite and infinite height.

The intuitive reason for this is as follows. Consider the next figure illustrating the definition of the trees $\mathcal{T}_\alpha$.

Suppose φ is of the form **ifp** $X : \psi(X)$, where ψ is in ML, and $\mathcal{T}_\omega, v_\omega \models \varphi$. At the first stage where the root v_ω is included into the fixed point of ψ, this happens because the successors of the root satisfy some criteria that ψ recognises. Now, if ψ contains a sub-formula $\Diamond\vartheta$ satisfied by $\mathcal{T}_\omega, v_\omega$ then there must be a subtree $\mathcal{T}_n$ satisfying ϑ. As $\mathcal{T}_n$ is also a subtree of the tree $\mathcal{T}_{n+1}$, the tree $\mathcal{T}_{n+1}$ and in fact all $\mathcal{T}_m$ with $m > n$ satisfy $\Diamond\vartheta$. Similarly, if a formula $\Box\vartheta$ holds at v_ω, then it also holds at all $\mathcal{T}_m$. Thus, for each sub-formula ϑ of ψ there is some fixed $n < \omega$ such that ϑ holds at all $\mathcal{T}_m$

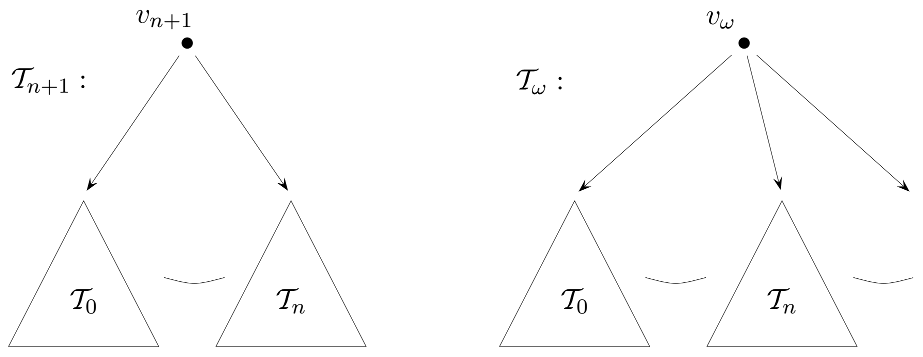


Figure 11.1: Illustration of the trees $\mathcal{T}_{n+1}$ and $\mathcal{T}_\omega$.

with $m > n$. Letting k be the maximum of these n, we get that ψ also holds at all $\mathcal{T}_m, v_m$ with $m \geq k$ and thus φ is true for $\mathcal{T}_m$ also.

To present the formal argument for the intuition given here, it is more convenient to work on ordinals instead of trees. For any ordinal α, let $\mathcal{O}_\alpha$ denote the structure $(\{\beta : \beta \leq \alpha\}, >)$. That is, the elements of the structure are all ordinals less than or equal to α, and there is an edge from β to γ if $\beta > \gamma$. The ordinal α is the maximal ordinal in the set, and we refer to it as the *root* of the structure $\mathcal{O}_\alpha$. Note that $\mathcal{O}_\alpha$ is bisimilar to the tree $\mathcal{T}_\alpha$ defined above.

For a formula φ, we write $\mathcal{O}_\alpha \models \varphi$ as shorthand for $\mathcal{O}_\alpha, \alpha \models \varphi$. It is easily seen that for any ordinal $\beta \leq \alpha$, $\mathcal{O}_\alpha, \beta \models \varphi$ if, and only if, $\mathcal{O}_\beta \models \varphi$. This is because the elements reachable by $>$-paths from β are exactly the ordinals below β, since $>$ is transitive. As a final bit of notation, if X is any atomic proposition on $\mathcal{O}_\alpha$, we write $\mathcal{O}_\alpha, \beta \models \varphi(X^-)$ to denote that $\mathcal{O}_\alpha, \beta \models \varphi(X - \{\beta\})$.

We begin with a few observations about the evaluation of inductive formulae on the structures $\mathcal{O}_\alpha$, which will be useful in the proof of the following lemma. First, we note that in $\mathcal{O}_\beta$, the maximum closure ordinal of any induction is $\beta + 1$. This can be proved by a straightforward induction on the ordinals. One consequence is that if $\mathcal{O}_\alpha, \beta \models (\mathbf{ifp}\ X \leftarrow \psi)$, then $\beta \in X^{\beta+1}$. Furthermore, since the truth of a formula φ at β can only depend on elements $\gamma \leq \beta$, we have that for any sets X and Y, if $X \cap (\beta+1) = Y \cap (\beta+1)$, then $\mathcal{O}_\alpha, \beta \models \varphi(X)$ if, and only if, $\mathcal{O}_\alpha, \beta \models \varphi(Y)$.

11.20 Lemma. *Let φ be a formula of* 1MIC. *If $X_1, \ldots, X_k \subseteq \omega$ are atomic propositions such that $\mathcal{O}_\omega \models \varphi(X_1, \ldots, X_k)$, then there is a finite N such that for all $n > N$, $\mathcal{O}_\omega, n \models \varphi(X_1^-, \ldots, X_k^-)$.*

Proof. Note that, by the hypothesis of the lemma, $\omega \notin X_i$ for any i. As we are now working in a fixed structure $\mathcal{O}_\omega$, we will say a formula φ holds (or is true) at α, to mean that $\mathcal{O}_\omega, \alpha \models \varphi$, which is the case if, and only if, $\mathcal{O}_\alpha \models \varphi$. The lemma is proved by induction on the nesting depth d of

ifp-operators.
Basis: If $d = 0$, the formula contains no occurrences of the **ifp**-operator, and is therefore equivalent to a formula of ML, where all negations are at the atoms. A simple induction on the structure of the formula then establishes the result.

- Any atomic formula X_i is false at ω by hypothesis, and, by definition, for all n, $\mathcal{O}_\omega, n \not\models X_i^-$. Therefore the claim holds for all atoms.
- For negated atoms the dual argument holds, i.e. any formula $\neg X_i$ is true at ω and for all n, $\mathcal{O}_\omega, n \models \neg X_i^-$.
- The case of the Boolean connectives $\wedge$ and $\vee$ is trivial.
- If φ is $\Diamond\psi$, then it is clear that $\mathcal{O}_\omega \models \varphi$ if, and only if, there is an $N < \omega$ such that $\mathcal{O}_N \models \psi$ if, and only if, for all $n > N$, $\mathcal{O}_\omega, n \models \Diamond\psi$ and therefore $\mathcal{O}_\omega, n \models \varphi(\overline{X}^-)$.
- Similarly, if φ is $\Box\psi$, $\mathcal{O}_\omega \models \varphi$ if, and only if, for all n, $\mathcal{O}_\omega, n \models \psi$ if, and only if, for all $n < \omega$, $\mathcal{O}_\omega, n \models \Box\psi(\overline{X}^-)$.

Induction step: If φ is a formula with depth $d+1$ of nesting of **ifp** operators, then it is a Boolean combination of formulae ϑ_i, each of which either has depth at most d, is of the form **ifp** $X \leftarrow \psi$, where ψ is a formula of depth at most d, or is of the form $\Diamond\psi$ or $\Box\psi$, where ψ has depth at most $d+1$. We assume that negations are always pushed inside modalities.

Clearly, if the claim holds for formulae ϑ_i and ϑ_j, then it also holds for $\vartheta_i \wedge \vartheta_j$ and $\vartheta_i \vee \vartheta_j$, just by taking N to be the maximum of N_i and N_j, which witness the claim for the two formulae. Thus, it suffices to prove the claim for the following four cases.

(1) $\vartheta := \Diamond\psi$. If $\mathcal{O}_\omega \models \vartheta$, then there is an $N < \omega$ such that $\mathcal{O}_\omega, N \models \psi$, and therefore, for all $n > N$, $\mathcal{O}_\omega, n \models \vartheta(\overline{X}^-)$.

(2) $\vartheta := \Box\psi$. If $\mathcal{O}_\omega \models \Box\psi$, then for all n, $\mathcal{O}_\omega, n \models \psi$ and therefore $\mathcal{O}_\omega, n \models \vartheta(\overline{X}^-)$.

(3) $\vartheta :=$ (**ifp** $X \leftarrow \psi$). Suppose $\mathcal{O}_\omega \models \vartheta$. Then, there is a stage α such that $\omega \notin X^\alpha$ and $\mathcal{O}_\omega \models \psi(X_1, \ldots, X_k, X^\alpha)$, i.e. $\omega \in X^{\alpha+1}$. By induction hypothesis, there is an N such that for all $n > N$, $\mathcal{O}_\omega, n \models \psi(X_1^-, \ldots, X_k^-, X^{\alpha-})$. Now, for each such n, if n is in stage X^α of the induction on $\psi(X_1^-, \ldots, X_k^-)$, then, by the inflationary semantics, $\mathcal{O}_\omega, n \models$ (**ifp** $X \leftarrow \psi)(X_1^-, \ldots, X_k^-)$, and we are done.

So, suppose n is not in stage X^α of the induction of $\psi(X_1^-, \ldots, X_k^-)$. However, since $X_i^- \cap n = X_i \cap n$, it follows that stage α of the induction of $\psi(X_1^-, \ldots, X_k^-)$ on $\mathcal{O}_n$ is exactly $X^{\alpha-} \cap (n+1)$. Thus, $\mathcal{O}_\omega, n \models \psi(X_1^-, \ldots, X_k^-, X^{\alpha-})$ implies $\mathcal{O}_\omega, n \models$ (**ifp** $X \leftarrow \psi)(X_1^-, \ldots, X_k^-)$.

(4) $\vartheta := \neg(\mathbf{ifp}\ X \leftarrow \psi)$. Suppose $\mathcal{O}_\omega \models \vartheta$. We have to show that there is some $N < \omega$ such that for all $n > N$, $\mathcal{O}_\omega, n \models \vartheta$. Towards a contradiction, assume that there are infinitely many n such that $\mathcal{O}_\omega, n \models (\mathbf{ifp}\ X \leftarrow \psi)(X_1^-, \ldots, X_k^-)$. Since $\mathcal{O}_\omega \models \neg(\mathbf{ifp}\ X \leftarrow \psi)$, it is the case that $\mathcal{O}_\omega \models \neg\psi(X_1, \ldots, X_k, X^\omega)$, and therefore, by induction hypothesis, there is an $N < \omega$ such that

$$\text{for all } n > N, \mathcal{O}_\omega, n \models \neg\psi(X_1^-, \ldots, X_k^-, X^{\omega-}). \qquad (*)$$

Note that in any $\mathcal{O}_\beta$, the closure ordinal of any induction is at most $\beta + 1$. Hence, for any $\beta < N$, $\mathcal{O}_\omega, \beta \models \psi(X_1, \ldots, X_k, X^\omega)$ if, and only if, $\mathcal{O}_\omega, \beta \models \psi(X_1, \ldots, X_k, X^N)$. As, by assumption, there are infinitely many n such that $\mathcal{O}_\omega, n \models (\mathbf{ifp}\ X \leftarrow \psi)(X_1^-, \ldots, X_k^-)$, there are infinitely many such greater than N. For each of these, there must be a least finite ordinal a_n such that $\mathcal{O}_\omega, n \models \psi(X_1^-, \ldots, X_k^-, X^{a_n})$. We distinguish two cases:

(a) There is a finite ordinal α that is the upper bound of all such a_n. In this case, we can show that there is a finite bound on the elements satisfying $(\mathbf{ifp}\ X \leftarrow \psi)(X_1^-, \ldots, X_k^-)$, from which the claim follows. To establish the finite bound, we show by induction on the stages, that each stage X^β contains only nodes up to some finite height. Clearly, this is the case for $\beta = 0$, as X^β is empty. Inductively, if X^β is bounded in height, then there is a formula of ML defining the set (see Lemma 11.2). Substituting this formula for X in ψ, we obtain a formula ψ' equivalent to $\psi(X^\beta)$, with **ifp** nesting depth d. Therefore, by the main induction hypothesis, since $\mathcal{O}_\omega \models \neg\psi'(X_1, \ldots, X_k)$, there is an M such that $\mathcal{O}_\omega, n \models \neg\psi'(X_1^-, \ldots, X_k^-)$ for all $n > M$. It follows that for all $n > M$, $n \notin X^{\beta+1}$. This implies that there is a finite bound on the height of the elements in X^α, contradicting the assumption that there are infinitely many n such that $\mathcal{O}_\omega, n \models (\mathbf{ifp}\ X \leftarrow \psi)(X_1^-, \ldots, X_k^-)$.

(b) There is no finite bound on the stages a_n. Thus, for any finite stage α there is some $a_n > \alpha$ such that $n \notin X^{a_n}$ and $\mathcal{O}_\omega, n \models \psi(X_1^-, \ldots, X_k^-, X^{a_n})$. Choose α and a_n such that $a_n > N$. Let c be the minimal node such that $c \notin X^{a_n}$ but $c \in X^\beta$ for some $\beta > a_n$. Clearly, for every node $m < a_n$, $m \in X^{a_n}$ if, and only if, $m \in X^\omega$. It follows that $c \geq a_n$ and thus $c > N$. Further, as c was chosen minimal,

$$\{m : m < c\} \cap X^{a_n} = \{m : m < c\} \cap X^\omega,$$

i.e. on the set of nodes reachable but different from c the fixed point of X is reached at stage a_n. Therefore,

$$\begin{aligned}&\mathcal{O}_\omega, c \models \psi(X_1^-, \dots, X_k^-, X^{a_n}) \qquad \text{if, and only if,}\\ &\mathcal{O}_\omega, c \models \psi(X_1^-, \dots, X_k^-, X^{\omega-}).\end{aligned}$$

As $c \geq a_n > N$ and, by $(*)$, $\mathcal{O}_\omega, c \not\models \psi(X_1^-, \dots, X_k^-, X^{\omega-})$ we get that $c \notin X^\beta$ for some $\beta > a_n$, contradicting the assumption.

This finishes the proof of the lemma. □

Recall that any structure $\mathcal{O}_\alpha$ is bisimulation equivalent to a well-founded tree of height α. It is a straightforward consequence of the lemma that the formula *finite-height* defined in Section 11.1 is not equivalent to any formula of 1MIC. We hence have established the following separation result.

11.21 Theorem. MIC *is strictly more powerful than* 1MIC.

While the separation given in Theorem 11.21 is proved on an infinite structure, the proof of Lemma 11.20 actually shows that the separation holds even when we restrict ourselves to finite structures. For, consider the collection of finite structures $\mathcal{O}_n$, $n < \omega$. The construction in Lemma 11.20 shows that for any formula φ of 1MIC, the set $\{n : \mathcal{O}_n \models \varphi\}$ is either finite or co-finite. Now, consider the formula η defined as

$$\eta := \textbf{ifp}\, Y : \begin{cases} X \leftarrow \Box\Box X \\ Y \leftarrow \neg X \wedge \Box X. \end{cases}$$

It can be verified that $\mathcal{O}_n \models \eta$ if, and only if, n is even. Hence, η is not equivalent to any formula of 1MIC.

The notation 1MIC was chosen, naturally, to suggest that we can define, for any natural number k, the fragment kMIC consisting of those formulae of MIC in which no **ifp** operation is defined over a system with more than k simultaneous formulae. The natural question that arises is whether increasing k gives rise to a hierarchy of increasing expressive power. We do not, at the moment, know of a method to settle this question one way or the other.

11.4.2 Infinity Axioms and the Satisfiability Problem

We have seen above that simple inductions in MIC provide less expressive power than simultaneous inductions. We show now that even without simultaneous inductions the finite model property fails and the satisfiability problem remains undecidable, in fact not even arithmetical.

Essentially, we use the same method as in Section 11.1 to reduce the decision problem for the first-order theory of arithmetic to Sat(1MIC). The proofs there relied crucially on simultaneous inductions on two variables X and Y and we have seen above that the formulae used there are not equivalent to any formulae of 1MIC. Instead, we use the following trick to simulate the simultaneous induction. Let $\mathcal{T}$ be a tree of height ω as used in Section 11.1. To simulate an induction on two variables we recursively make two copies of the successors of each node in $\mathcal{T}$ and label the root of one of the copies by the proposition a and the root of the other by b. Let $\mathcal{T}'$ be this new tree. Thus, every node in $\mathcal{T}'$ of finite height is labelled by a or b and the height of each of these nodes u equals the length of the longest path entirely labelled by a's from a successor of u to a leaf which, again, is the same as the length of the longest path entirely labelled by b's from a successor of u to a leaf. On this model, we can simulate the simultaneous induction on two variables X and Y by an induction on one variable Z by letting Z contain all copies of nodes contained in X which are labelled by a and all copies of nodes contained in Y labelled by b.

We now turn to the axiomatisation of this tree model. Besides the proposition symbols a and b already mentioned there are two other propositions, namely w, with which only the root is labelled, and s, labelling only the direct successors of the root. As in Section 11.1, we restrict attention to well-founded tree models, i.e. tree models of the formula $\mu X.\Box X$.

We first define a formula *label* ensuring that its models are labelled as described above. Let the formula *label* be defined as

$$\begin{aligned}\text{label} := \quad & w \wedge \neg a \wedge \neg b \wedge \neg s \wedge \Box s \wedge \Box\Box \mathit{everywhere}(\neg s) \wedge \\ & \Box \mathit{everywhere}(\neg w \wedge (a \vee b) \wedge \neg(a \wedge b)),\end{aligned}$$

where *everywhere* is defined as in Section 11.1 as $\mathit{everywhere}(\varphi) := \textbf{ifp}\ X \leftarrow \varphi \wedge \Box X$.

11.22 Proposition. *In any model $\mathcal{T}, v$ of* label *the root and only the root v is labelled by w, all direct successors of v, and only those, are labelled by s, and all nodes reachable but different from v are either labelled by a or by b but not by both.*

The next step is to ensure that the models we are going to describe are of height ω. Towards this end, we first introduce some notation.

11.23 Definition. *A node labelled by a is called an a-*node*. An a-path between two nodes is a path between them consisting only of a-nodes. Finally, we inductively define the a-*height *α of an a-node u as follows.*

- *The a-height of leaves labelled by a is defined to be 0.*
- *For all other a-nodes, their a-height is defined as the least strict upper bound of all β such that β is the a-height of a successor of u labelled by a.*

The notion of b-nodes, b-paths, and the b-height of a node is defined analogously.

Let the formula ϑ be defined as

$$\vartheta := (b \wedge \Box(b \rightarrow X)) \vee (a \wedge \Box(a \rightarrow X)).$$

A simple induction on the stages establishes the following lemma.

11.24 Lemma. *Let $\mathcal{T}$ be a tree. Then, for all α, the α-th stage X^α of the induction on ϑ contains exactly the nodes of a- or b-height less than α.* $\Box$

Consider the formula

$$\text{infinity} := \text{label} \wedge \textit{everywhere}(\Diamond a \leftrightarrow \Diamond b) \wedge \text{inf} \wedge \neg\textbf{ifp } X \leftarrow \varphi,$$

where

$$\text{inf} := \neg\textbf{ifp } X \leftarrow (b \wedge \Diamond b \wedge \Box(b \rightarrow \Box\textit{false})) \vee \varphi.$$

and

$$\begin{aligned}\varphi := \vartheta \vee\ & (w \wedge \Box(b \rightarrow X) \wedge \Diamond(a \wedge \neg X)) \vee \\ & (w \wedge \Box(a \rightarrow X) \wedge \Diamond(b \wedge \neg X)).\end{aligned}$$

11.25 Lemma. *Let $\mathcal{T}, v \models \textit{label} \wedge \textit{everywhere}(\Diamond a \leftrightarrow \Diamond b)$ and let $\gamma_a := \sup\{\alpha : \alpha$ is the a-height of a direct successor of $v\}$ and $\gamma_b := \sup\{\alpha : \alpha$ is the b-height of direct successor of $v\}$.*

(*i*) *$\mathcal{T}, v \models \neg\textbf{ifp } X \leftarrow \varphi$ if, and only if, $\gamma_a = \gamma_b$.*

(*ii*) *$\mathcal{T}, v \models \text{inf}$ if, and only if, γ_a and γ_b are finite and $\gamma_a + 1 = \gamma_b$ or both are infinite and $\gamma_a = \gamma_b$.*

Proof.

(*i*) For Part (*i*), we show that the root v satisfies $\textbf{ifp } X \leftarrow \varphi$ if, and only if, there is a b-successor of v whose b-height is greater than the a-height of any a-successor of v or vice versa.

Towards the forth direction, suppose $\mathcal{T}, v \models \textbf{ifp } X \leftarrow \varphi$. Thus, there is a stage α such that $(\mathcal{T}, X^\alpha), v \models \varphi$. As $\mathcal{T}$ is a model of the formula *label* above, this implies

$$\begin{aligned}(\mathcal{T}, X^\alpha), v \models\ & (w \wedge \Box(b \rightarrow X) \wedge \Diamond(a \wedge \neg X)) \vee \\ & (w \wedge \Box(a \rightarrow X) \wedge \Diamond(b \wedge \neg X)).\end{aligned}$$

Suppose $(\mathcal{T}, X^\alpha), v \models (w \wedge \Box(b \rightarrow X) \wedge \Diamond(a \wedge \neg X))$. Thus, using Lemma 11.24, all b-successors of v are of height less than α, whereas there is at least one a-successor whose height is greater than or equal to α. The case where $(\mathcal{T}, X^\alpha), v \models (w \wedge \Box(a \rightarrow X) \wedge \Diamond(b \wedge \neg X))$ is analogous.

For the converse, if v has a b-successor whose b-height α is greater than the a-height of any a-successor, then at stage α, $X^{<\alpha}$ contains all a-successors but not all b-successors of v. Thus, v would be in X^α. The case where there is an a-successor of v whose a-height is greater than the b-height of any b-successor of v is analogous.

Thus we have shown that the root v satisfies $\mathbf{ifp}\ X \leftarrow \varphi$ if, and only if, there is a b-successor of v whose b-height is greater than the a-height of any a-successor of v or vice versa. This proves the first part of the lemma.

(*ii*) To establish Part (*ii*), we first prove by induction on the finite stages, that for all $0 < n < \omega$, X^n contains exactly the a-nodes of height less than n and the b-nodes of height less than or equal to n. For the case of nodes labelled by a this follows immediately from Lemma 11.24 as the additional disjuncts only affect b-nodes (and the root v).

Let $n = 1$. Obviously, X^1 contains all b-nodes of height 0 and all b-nodes satisfying $\Diamond \mathit{true} \wedge \Box\Box \mathit{false}$. Clearly, such a node u must be of height 1, as it only has leaves as successors. As $\mathcal{T}$ is a model of the formula *label*, this means that u must have a b-successor and thus is of b-height 1.

For the induction step assume the claim has already been proved for all $n' < n$. An argument as in the proof of Part (*i*) above shows that X^n contains all b-nodes of b-height less than or equal to n. This finishes the induction.

Now consider the stage ω. We have seen that X^ω contains all nodes of finite a- or b-height and the same argument as in Lemma 11.24 shows that for all $\alpha \geq \omega$, X^α contains all nodes of a- or b-height less than α.

Now suppose that the root v of the tree occurs in X^∞. There must be a stage $\alpha+1$ such that v occurs in $X^{\alpha+1}$ but not in X^α, i.e. $(\mathcal{T}, X^\alpha), v \models (w \wedge \Box(b \rightarrow X) \wedge \Diamond(a \wedge \neg X)) \vee (w \wedge \Box(a \rightarrow X) \wedge \Diamond(b \wedge \neg X))$. Suppose that $(\mathcal{T}, X^\alpha), v \models (w \wedge \Box(b \rightarrow X) \wedge \Diamond(a \wedge \neg X))$. Thus, there is an a-successor of v whose a-height is greater than the b-height of any b-successor. If α is infinite, this means that $\gamma_a > \gamma_b$. If α is finite, this means that all b-successors of v are of height less than or equal to α whereas there is an a-successor of v of height greater than or equal to α. This implies that $\gamma_a + 1 > \gamma_b$.

On the other hand, if $(\mathcal{T}, X^\alpha), v \models (w \wedge \Box(a \rightarrow X) \wedge \Diamond(b \wedge \neg X))$, this implies that $\gamma_b > \gamma_a$ if α is infinite and $\gamma_b > \gamma_a + 1$ otherwise.

This proves that if $\mathcal{T}, v \models \mathit{inf}$ then γ_a and γ_b are finite and $\gamma_a + 1 = \gamma_b$ or both are infinite and $\gamma_a = \gamma_b$. The converse direction follows immediately.

□

A simple consequence of the lemma is the following corollary.

11.26 Corollary. *For every tree $\mathcal{T}$ and node v, if $\mathcal{T}, v \models$ infinity then the height of v is infinite. Thus,* 1MIC *does not have the finite model property.*

To finish the axiomatisation of the intended model, we have to ensure that the models of our formula not just have infinite height but height exactly ω and further, that for all nodes of finite height, all of their successors are of the same height.

To formalise that the tree has height exactly ω we say that the root is of infinite height but none of its successors is. For this let $\mathrm{inf}_s := \mathrm{inf}[w/s, b/(b\wedge \neg s), a/(a \wedge \neg s)]$ be the result of replacing in the formula *inf* above each w by s, each b by $(b \wedge \neg s)$, and each a by $(a \wedge \neg s)$. Analogously, define $\varphi_s := \varphi[w/s, b/(b \wedge \neg s), a/(a \wedge \neg s)]$. Then a similar argument as in Lemma 11.25 shows the following.

11.27 Lemma. *Let the formula ω-*height *be defined as*

$$\omega\textit{-height} := \textit{infinity} \wedge \Box\neg(\textit{inf}_s \wedge \neg\mathbf{ifp}\ X \leftarrow \varphi_s).$$

Then for any tree $\mathcal{T}$ with root v, if $\mathcal{T}, v \models \omega$-height then the height of v is infinite but the height of all successors of v is finite. Thus, the height of v is exactly ω.

Finally, we formalise that for all nodes of finite height all of their successors are of the same height.

11.28 Lemma. *Consider the formula ψ defined as*

$$\psi := \neg\mathbf{ifp}\ X \leftarrow (\neg w \wedge \Box X) \vee (w \wedge \mathbf{ifp}\ Y \leftarrow \Diamond Y \vee (\neg w \wedge \Diamond X \wedge \Diamond\neg X)).$$

Let $\mathcal{T}, v \models \omega$-height $\wedge\, \psi$ be a tree. Then, for all nodes u of finite height all direct successors of u are of the same height.

Proof. As $\mathcal{T}, v \models \omega$-height, no node of finite height is labelled by w. A simple induction on the stages proves that at stage $n \in \omega$, X^n contains all nodes of height less than n. Now assume that at some stage n the root v is included into X, i.e. v satisfies $\mathbf{ifp}\ Y \leftarrow \Diamond Y \vee (\neg w \wedge \Diamond X \wedge \Diamond\neg X)$. Thus there is a node u whose height is greater than n but has a successor in X^n, i.e. one whose height is less than n, and a successor not in X^n, i.e. one whose height is greater than or equal to n.

Conversely, if there is a node u of some finite height n which has a successor of height less than $n-1$, then $\Diamond X \wedge \Diamond\neg X$ becomes true at stage $n-1$ and the root v is included into X^n.

Together, we get that $\mathcal{T}, v \models \psi$ if, and only if, for all nodes u of finite height, all of its successors are of the same height. □

We now turn to the reduction of the decision problem for the first-order theory of arithmetic to the satisfiability problem for 1MIC. In the remainder of this section we only consider models of the formula ω-height $\wedge\ \psi$. As in Section 11.1, we code natural numbers by sets of nodes of certain height. The difference is, that here a number $n \in \omega$ is coded by the set of nodes u of height less than or *equal to* n. In particular, the number 0 is not coded by the empty set but by the set of leaves. To simplify the presentation, we do not work on the first-order theory of arithmetic directly but on the first-order theory of the following structure.

11.29 Definition. *Let* $\mathfrak{N}' := (\mathbb{N}, +, \cdot, 0, 1)$ *be the structure over the universe* $\mathbb{N}$*, where the constants* $0^{\mathfrak{N}'}, 1^{\mathfrak{N}'}$ *and the function* $+^{\mathfrak{N}'}$ *are interpreted as in the standard model* $\mathfrak{N}$ *of arithmetic, and* $\cdot^{\mathfrak{N}'}$ *is interpreted as follows:* $s \cdot^{\mathfrak{N}'} t := 0$ *if* $s = 0$ *or* $t = 0$*, and otherwise* $s \cdot^{\mathfrak{N}'} t := s \cdot^{\mathfrak{N}} (t + 1)$*.*

Clearly, the first order theories of $\mathfrak{N}$ and $\mathfrak{N}'$ can be reduced to each other. Thus reducing the decision problem for the theory of $\mathfrak{N}'$ to Sat(1MIC) shows this problem to be undecidable.

Given the encoding of natural numbers explained above, consider the formula *plus*(S, T) defined as

$$\begin{aligned} \mathrm{plus}(S,T) \;:=\; & (\Box(b \rightarrow \psi_+) \wedge \neg \mathit{everywhere}(T \leftrightarrow \Box \mathit{false}) \wedge \\ & \qquad \neg \mathit{everywhere}(S \leftrightarrow \Box \mathit{false})) \vee \\ & \Box \mathit{false} \vee (\mathit{everywhere}(T \leftrightarrow \Box \mathit{false}) \wedge S) \vee \\ & \qquad (\mathit{everywhere}(S \leftrightarrow \Box \mathit{false}) \wedge T) \end{aligned}$$

where ψ_+ is defined as

$$\begin{aligned} \psi_+ := \mathbf{ifp}\ X \leftarrow \;& (a \wedge \Box(a \rightarrow X)) \vee (b \wedge S) \vee \\ & (b \wedge \Box(b \rightarrow X) \wedge \mathit{everywhere}(\Box\Diamond(a \wedge X) \rightarrow T)). \end{aligned}$$

11.30 Lemma. *If* S *and* T *encode natural numbers* s *and* t *as described above, then* plus(S, T) *encodes the sum* $s + t$*.*

Proof. Let s and t be the natural numbers coded by the sets S and T respectively. Assume $s, t > 0$ and consider the sub-formula ψ_+. We claim that at stage $i > 0$, X^i contains all a-nodes of height less than i and all b-nodes of height at most $\min\{s + i - 1, s + t - 1\}$. For a-nodes, this is trivial. The claim for the b-nodes is proved by induction on the stages $i > 0$.

- Clearly, X^1 contains exactly the b-nodes contained in S and thus, as $t \geq 1$, only nodes of height at most $s + t - 1$.
- Now assume that for $0 < i < t$ the claim has been proved, i.e. X^i contains all a-nodes of height less than i and all b-nodes of height at most $s + (i - 1)$. Therefore, the sub-formula *everywhere*$(\Box\Diamond(a \wedge X) \rightarrow T)$ is globally true and all b-nodes of height at most $s + i$ are included into X^{i+1}, as they satisfy $b \wedge \Box(b \rightarrow X)$.

- Now suppose $i = t$. Thus, X^i contains all a-nodes of height less than t. Clearly, the sub-formula $b \wedge \Box(b \rightarrow X)$ is true only for b-nodes of height at most $s + (t-1) + 1$. Let u be a node of height exactly $s + t$ and thus, as $s \geq 1$, greater than i. Then there is a node of height $t + 1$ in the subtree rooted at u which satisfies $\Box\Diamond(a \wedge X)$ but not T. Therefore, $\mathit{everywhere}(\Box\Diamond(a \wedge X) \rightarrow T)$ is false at u and u is not included into X^{i+1}.

- The same argument shows that at no higher stage, a b-node of height greater than $s+t-1$ can be added to the fixed-point. This proves the claim.

Now consider the formula $\mathit{plus}(S,T)$. By assumption, $s, t > 0$ and therefore the formulae $\neg\mathit{everywhere}(T \leftrightarrow \Box\mathit{false})$ and $\neg\mathit{everywhere}(S \leftrightarrow \Box\mathit{false})$ are true for all nodes except for the leaves. Further, we have seen that the sub-formula $\Box(b \rightarrow \psi_+)$ becomes true for all nodes whose b-successors are of height less than or equal to $s + t - 1$ and, as we are working in models of the formula ω-height $\wedge\ \psi$, for all nodes of height less than or equal to $s + t$.

Now suppose $s = 0$. Then $\neg\mathit{everywhere}(S \leftrightarrow \Box\mathit{false})$ is false for all nodes except for the leaves. Thus, $\Box\mathit{false} \vee \mathit{everywhere}(S \leftrightarrow \Box\mathit{false}) \wedge T$ defines all nodes contained in T and $\mathit{plus}(S,T)$ defines the set of nodes representing the sum $0 + t = t$. The case where $t = 0$ is analogous. This finishes the proof. □

We now turn to the formalisation of multiplication in 1MIC. For this, consider the formula

$$\begin{aligned}\mathrm{times}(S,T) := \Box\mathit{false}\ \vee\ (&\neg\mathit{everywhere}(S \leftrightarrow \Box\mathit{false}) \wedge \\ &\neg\mathit{everywhere}(T \leftrightarrow \Box\mathit{false}) \wedge \Box(b \rightarrow \psi_*)),\end{aligned}$$

where

$$\begin{aligned}\psi_* := \mathbf{ifp}\ X \leftarrow (a \wedge \Box(a \rightarrow X) \vee (b \wedge \mathrm{plus}(\Box(b \rightarrow X), S) \wedge \\ \mathit{everywhere}(\Box\Diamond(a \wedge X)) \rightarrow T)).\end{aligned}$$

11.31 Lemma. *If S and T encode natural numbers s and t as described above, then* $\mathrm{times}(S,T)$ *encodes the product $s \cdot (t+1)$ if $s, t > 0$ and 0 otherwise.*

Proof. First, suppose that s or t equals 0. In this case, one of the sub-formulae $\neg\mathit{everywhere}(S \leftrightarrow \Box\mathit{false})$ or $\neg\mathit{everywhere}(T \leftrightarrow \Box\mathit{false})$ becomes false on all nodes except the leaves and therefore the formula $\mathit{times}(S,T)$ is true only on the leaves and encodes the result 0.

Now suppose $s, t > 0$ and consider the induction on X in the sub-formula ψ_*. We claim that at stage $i > 0$, X^i contains all a-nodes of height less than i and all b-nodes of height at most $\min\{i \cdot s + i - 1, t \cdot s + t - 1\}$. For the a-nodes, the claim is obvious. Now consider a node labelled by b.

- In the first stage, X^1 contains all b-nodes satisfying $\text{plus}(\Box(b \to X), S)$. As X^0 is empty, $\Box(b \to X)$ is true only at the leaves and thus the formula *plus* evaluates to S.

- Now assume the claim has been proved for stage i. If $i < t$, then X^i contains a-nodes of height less than $i < t$. Thus, the sub-formula $\mathit{everywhere}(\Box\Diamond(a \wedge X)) \to T)$ is true for all nodes. By induction hypothesis, X^i contains exactly the b-nodes of height less than or equal to $i{\cdot}s{+}i{-}1$ and therefore $\Box(b \to X)$ becomes true at all nodes of height at most $i \cdot s + i$. By Lemma 11.30, the formula $\text{plus}(\Box(b \to X), S)$ is then true for all nodes of height $(i \cdot s + i + s) = (i+1) \cdot s + (i+1) - 1$.

- Now suppose $i = t$. X^i contains exactly all a-nodes of height less than t and all b-nodes of height at most $t \cdot s + t - 1$. Let u be any b-node not in X^i, i.e. of height greater than $t \cdot s + t - 1$. Then there is a node of height $t + 1$ in the subtree rooted at u satisfying $\Box\Diamond(a \wedge X)$ but not T. Thus, the formula $\mathit{everywhere}(\Box\Diamond(a \wedge X) \to T)$ is false at u and u is not added to the fixed-point. The same argument, of course, holds for all stages $i > t$ and thus, in restriction to the b-nodes, the fixed point of X has been reached. This finishes the proof of the claim.

Now consider the formula *times*. By assumption, $s, t > 0$ and therefore the sub-formulae $\neg\mathit{everywhere}(S \leftrightarrow \Box\mathit{false})$ and $\neg\mathit{everywhere}(T \leftrightarrow \Box\mathit{false})$ are both true for all nodes except the leaves. Thus, $\text{times}(S, T)$ becomes true for all leaves – as they satisfy $\Box\mathit{false}$ – and for all nodes whose b-successors satisfy ψ_*, i.e. are of height $((t+1) \cdot s - 1)$. Together, $\text{times}(S, T)$ represents $(t + 1) \cdot s$. $\Box$

The following corollary shows that the evaluation of polynomials in the structure $\mathfrak{N}'$ can be reduced to the evaluation of 1MIC formulae.

11.32 Corollary. *For every polynomial $f(x_1, \ldots, x_r)$ over $\mathfrak{N}'$ with coefficients in the natural numbers there exists a formula $\psi_f(X_1, \ldots, X_r) \in$ 1MIC such that for every tree $\mathcal{T}, v \models \omega$-height $\wedge\, \psi$ and all sets $S_1, \ldots, S_r$ encoding numbers $s_1, \ldots, s_r \in \omega$*

$$[\![\psi_f(S_1, \ldots, S_r)]\!]^{\mathcal{T}} = \{v : h(v) \leq f^{\mathfrak{N}'}(s_1, \ldots, s_r)\},$$

where $f^{\mathfrak{N}'}(s_1, \ldots, s_r)$ denotes the result of f in $\mathfrak{N}'$.

Proof. The proof is by induction on f.

- $\psi_0 := \Box\mathit{false}$.

- $\psi_1 := \Box\Box\mathit{false}$.

- $\psi_x := X$.

- $\psi_{f+g} := \text{plus}[S/\psi_f, T/\psi_g]$, i.e. the formula obtained by replacing in $\text{plus}(S,T)$ the variables S and T by ψ_f and ψ_g, respectively.
- $\psi_{f\cdot g} := \text{times}[S/\psi_f, T/\psi_g]$.

$\square$

11.33 Theorem. *For every* FO*-sentence ψ in the vocabulary $\{+,\cdot,0,1\}$ of arithmetic, there exists a formula $\psi^* \in$* 1MIC *such that ψ is true in $\mathfrak{N}'$ if, and only if, ψ^* is satisfiable.*

Proof. By induction on formulae $\psi(\overline{x}) \in \text{FO}[+,\cdot,0,1]$ we construct a formula $\psi^*(\overline{x}) \in \text{1MIC}$ such that for all $\overline{n} \in \omega$ with encodings $\overline{N}$, $\psi(\overline{n})$ is true in $\mathfrak{N}'$ if, and only if, $\psi^*(\overline{N})$ is true at the root v of any model $\mathcal{T}, v$ of the formula ω-height $\wedge\, \psi$ above.

We have already seen how to transform polynomials over $\mathfrak{N}'$ and equality $x = y$ can be expressed by *everywhere*$(X \leftrightarrow Y)$. What remains is to translate quantifiers. Let $\psi(\overline{y}) := \exists x\, \varphi(x, y_1, \dots, y_k)$ be a formula in $\text{FO}[+,\cdot,0,1]$. By induction we get a corresponding formula $\varphi^*(X, Y_1, \dots, Y_k)$ in 1MIC. Let

$$\psi^*(\overline{Y}) := w \wedge \mathbf{ifp}\ X \leftarrow (\neg w \wedge \Box X) \vee (w \wedge \varphi^*(X/\neg w \wedge \Box X)).$$

Let $\mathcal{T}, v$ be any model of ω-height $\wedge\, \psi$. We claim that for all numbers $\overline{n}$ encoded by sets $\overline{N}$, $\psi(\overline{y})$ is true in $\mathfrak{N}'$ if, and only if, $\mathcal{T}, v \models \psi^*(\overline{N})$. A simple induction on the stages proves that at stage $i < \omega$, X^i contains all a and b-nodes of height less than i. Further, X^∞ contains the root v if at some stage X, the formula φ^* becomes true at the root where the variable X has been replaced by $\neg w \wedge \Box X$, i.e. at each stage i, the variable X in φ^* is interpreted by the set of nodes of height $\leq i$. Thus, in the course of the induction, φ^* is evaluated for all sets encoding natural numbers and, by induction hypothesis, becomes true at the root if, and only if, for at least one $i \in \mathbb{N}$, $\varphi(i, \overline{n})$, and thus $\psi(\overline{n})$, becomes true. This proves the claim and finishes the proof of the theorem. $\square$

11.34 Corollary. *The satisfiability problem for* 1MIC *is undecidable. In fact, it is not even in the arithmetical hierarchy.*

11.5 Comparison of Least and Inflationary Fixed-Point Inductions

In Chapter 8 we studied extensions of first-order logic by least and inflationary fixed-point operators. As we have seen, the two logics are equivalent in expressive power yet differ in various other aspects.

The results about the modal iteration calculus reported in the previous sections clearly show that in the context of modal logic, least and inflationary fixed-point inductions have rather different properties. Whereas the modal μ-calculus enjoys desirable algorithmical properties such as exponential time satisfiability testing and an NP $\cap$ CO-NP model checking problem, the corresponding problems for MIC are highly undecidable and PSPACE-complete, respectively. On the other hand, MIC has much more expressive power than L_μ.

This raises the question, what the essential properties of logics are that make the translation of inflationary into least fixed points possible. We will try to shed light on this question by considering various fragments of LFP and IFP which generalise the modal least and inflationary fixed-point logic.

Essentially, modal fixed-point logics are restricted in two ways. Firstly, the underlying modal logic allows only a very restricted form of quantification, namely quantification along edges in the graph, and it does not have free first-order variables which might be used to mark distant parts of the graph. Secondly, the fixed-point part is effectively restricted to monadic inductions, i.e. inductions over sets. Thus, fragments of LFP and IFP with expressive power somewhere between the modal and full fixed-point logics can be formed by allowing first-order variables and more powerful quantifiers in the underlying logic or by allowing fixed-point inductions of higher arity.

Extending the Underlying Logic

Natural candidates for the first approach are fixed-point extensions of the guarded fragment of FO. The *guarded fragment* was introduced by Andréka, van Benthem, and Németi in [AvBN98] as a fragment of FO to explain the good algorithmic properties of modal logic. A least fixed-point logic (μGF) based on the guarded fragment has been considered by Grädel and Walukiewicz in [GW99].

Unfortunately, inflationary fixed points do not fit well into the guarded fragment: Clearly, IFP is contained in second-order logic (SO). The corresponding guarded second-order logic (GSO) has been studied by Grädel, Hirsch, and Otto in [GHO02]. They showed that on words, GSO collapses to MSO and thus the only languages that are definable in GSO are the regular languages. As already MIC can define non-regular languages any reasonable definition of a guarded inflationary fixed-point logic iGF would no longer be contained in GSO and thus leave the realms of guarded logics.

So instead of guarded logics, we may consider the monadic fixed-point logics M-LFP and M-IFP, i.e. allow arbitrary first-order formulae but restrict the fixed-point part to monadic inductions. An immediate consequence of the Knaster-Tarski Theorem 3.3 is that M-LFP is contained in MSO. Thus again, all M-LFP-definable languages are regular whereas already in MIC and therefore also in M-IFP there are non-regular languages definable. Thus in the restriction to monadic inductions, inflationary fixed points are more expressive than least fixed points.

Extending the Fixed-Point Part

Following the second approach mentioned above, there are two possibilities: allowing fixed-point inductions of higher arity while keeping modal logic as underlying formalism or considering full first-order logic but restricting the fixed-point relations to some fixed arity.

A variant of the modal μ-calculus - the *higher dimensional μ-calculus* (L_μ^ω) - which allows inductions of higher arity has been introduced in [Ott99] by Martin Otto to define a logic that captures bisimulation-invariant PTIME, i.e. every property of transition systems that is invariant under bisimulation and decidable in polynomial time is definable in L_μ^ω. He also showed that L_μ^ω is the bisimulation invariant fragment of LFP. It is clear that the analogous higher dimensional inflationary fixed-point logic, a logic that we might call MIC$^\omega$, is still contained in IFP and therefore, by Theorem 8.7, in LFP. As any property definable in MIC$^\omega$ is bisimulation invariant, this implies that L_μ^ω and MIC$^\omega$ are equivalent.

Finally, we briefly consider the fixed arity fragments of LFP and IFP. Let LFP^k and IFP^k denote the fragments where all fixed-point variables are of arity at most k. We have already seen that in the arity one fragments, i.e. M-LFP and M-IFP, inflationary fixed points are more expressive than least fixed points. However, M-IFP is contained in LFP^2, i.e. LFP with binary fixed-point variables. In fact, using the stage comparison method, it is easily seen that every formula in IFP^k is equivalent to a formula in LFP^{2k}.[1] It seems unlikely, that this increase in arity of the involved fixed-point variables can be avoided. Therefore we put forth the following conjecture.

11.35 Conjecture. *For every $k < \omega$,* LFP$^k \lneq$ IFP^k.

Note that an answer to this question on finite ordered structures is closely related to important open questions in complexity theory, namely the equivalence of the complexity classes LOGSPACE and PTIME.[2] Figure 11.2 gives a summary of what we have seen in the preceding paragraphs.

The results hint at a possible reason why MIC and L_μ are so different, namely not so much the fact that both are modal logics but that both are effectively monadic logics. By defining those elements that are not yet in the fixed point but will enter in the next stage, inflationary inductions gain some access to the ordinals indexing the stages and with it are capable of a limited amount of counting. This technique has frequently been employed

[1]Note, however, that this translation introduces an unbounded number of parameters, i.e. free variables $\overline{y}$ in sub-formulae $[\mathbf{lfp}_{R,\overline{x}}\varphi(R,\overline{x},\overline{y})](\overline{x})$ other than $\overline{x}$. Thus, the resulting LFP-formulae are not of bounded width.

[2]It was shown by Grohe and Imhof [Gro94, Imh96a], that on finite ordered structures, deterministic transitive-closure logic DTC, which captures LOGSPACE, is contained in LFP^2 and M-IFP. As LFP captures PTIME, a positive answer for some $k \geq 2$ to the conjecture on finite ordered structures would imply LOGSPACE $\neq$ PTIME.

		fixed points:	
	monadic	arity-restricted	unrestricted
ML	$L_\mu \lneq \mathrm{MIC}$		$L_\mu^\omega = \mathrm{MIC}^\omega$
GF		$\mu\mathrm{GF} \lneq i\mathrm{GF}$	
FO	$\text{M-LFP} \lneq \text{M-IFP}$	$\mathrm{LFP}^k \overset{?}{\lneq} \mathrm{IFP}^k$	$\mathrm{LFP} = \mathrm{IFP}$

Figure 11.2: Least vs. inflationary fixed points.

throughout this chapter and many results about MIC have been proved by expressibility results involving counting.

11.6 Perspectives and Open Problems

A line of research active in recent years relates fixed-point logics to infinite two-player games. In particular the relation between least fixed-point logics – such as the modal μ-calculus – and parity games has proved very fruitful and indeed, many competitive evaluation methods currently available for such logics are based on model-checking games. See [GTW02] for a recent text on this subject.

However, so far the game based approach has primarily focused on logics like L_μ, LTL, and CTL as used in the area of verification. In particular, model checking games for inflationary fixed-point logics like MIC have not been considered so far.

For various reasons, it would be interesting to define such a game for IFP and MIC. Although the model-checking problem for MIC is known to be PSPACE-complete, one might be able to obtain game-based model-checking algorithms for MIC with reasonable running time in practise.

Also, by showing that the winner of a game for inflationary fixed-point logics is definable in LFP, we would get a completely different proof for the equivalence of LFP and IFP than the one given in Section 8.2. In this way, the equivalence proof is split into two parts, namely the combinatorial problem of defining a game for IFP and the logical problem of defining the winner of such a game in LFP. This might help to produce a less technical proof for the equivalence of the two logics. However, the constructive and explicit translation of IFP-formulae into LFP-formulae would obviously be lost in this approach.

Chapter 12

The Modal Partial Iteration Calculus

Recall the different semantics we gave to partial fixed-point logic in Chapter 7. We continue the study of partial fixed-point inductions in the context of modal logic. In particular, we will introduce a modal partial fixed-point logic and study its properties. We first define the syntax and comment on the semantics later.

12.1 Definition (Syntax of MPC). *The* modal partial iteration calculus (MPC) *extends modal logic by the following rule. If* $\varphi_1, \dots, \varphi_k$ *are formulae of* MPC*, and* $X_1, \dots, X_k$ *are propositional variables, then*

$$S := \begin{cases} X_1 & \leftarrow & \varphi_1(X_1, \dots, X_k) \\ & \vdots & \\ X_k & \leftarrow & \varphi_k(X_1, \dots, X_k) \end{cases}$$

is a system *of rules, and* $(\mathbf{pfp}\ X_i : S)$ *is a formula of* MPC*. If* S *consists of a single rule* $X \leftarrow \varphi$ *we simplify the notation and write* $(\mathbf{pfp}\ X \leftarrow \varphi)$ *instead of* $(\mathbf{pfp}\ X : X \leftarrow \varphi)$*.*

We now turn towards a semantics for MPC.

12.1 Semantics for Modal Partial Fixed-Point Inductions

The first semantics for partial fixed-point logic considered in Part I was a semantics that arose in the area of finite model theory. We argued that the finite model semantics is not satisfactory as it is a) restricted to finite models and b) does not generalise as easily to other logical formalisms as the alternative semantics. The following example gives further justification to this claim.

Consider MPC under the finite model semantics: On any finite transition system $\mathcal{K}$ with universe V, a formula $\psi := (\mathbf{pfp}\ X_i : S)$ as above, defines for every ordinal α a tuple $\overline{X}^{\alpha} := (X_1^{\alpha}, \dots, X_k^{\alpha})$ of sets, via the following induction rules.

$$\begin{aligned} X_i^0 &:= \varnothing \\ X_i^{\alpha+1} &:= [\![\varphi_i]\!]^{(\mathcal{K}, \overline{X}^{\alpha})} \end{aligned}$$

If there is a stage α such that $\overline{X}^{\alpha} = \overline{X}^{\alpha+1}$, then $[\![\psi]\!]^{\mathcal{K}} = X_i^{\alpha}$. It is easily seen that this straightforward adaptation of the finite model semantics to the context of modal logic is not closed under bisimulation. For, consider the formula

$$\psi := \mathbf{pfp}\ X : \begin{cases} X \leftarrow w \vee (\Box \mathit{false} \wedge \neg Y) \vee \Diamond X \\ Y \leftarrow \Box \mathit{false} \end{cases}$$

and the induction it defines on the transition system $\mathcal{K}, w := w \to \bullet$. Obviously, X^1 contains both nodes, and all other stages only contain w. Therefore, the fixed point defined by ψ is the set $\{w\}$.

Now consider the transition system $\mathcal{K}', w' := w \to \bullet \qquad \bullet \leftrightarrow \bullet \to \bullet$. Clearly, the two transition systems $\mathcal{K}, w$ and $\mathcal{K}', w'$ are bisimilar, as the new nodes in $\mathcal{K}'$ are not reachable from w'. But, in $\mathcal{K}'$, the stage X^2 contains not only the node labelled by w, but also the node in the middle of the second component. Further, X^3 contains, besides w, also the left node in the new component and in the process of the fixed-point induction, these two sets alternate. Thus, there is no stage α where $X^{\alpha} = X^{\alpha+1}$ and the fixed point defined by ψ on $\mathcal{K}', w'$ is empty.

Invariance under bisimulation is considered a defining feature of modal logics, wherefore this definition of a *modal* partial fixed-point semantics is not satisfactory.

To establish invariance under bisimulation, we modify the semantics as follows.

12.2 Definition. *Let $\mathcal{K}$ be a transition system with universe V and let S be a system of rules. Consider again the sequence of tuples $\overline{X}^{\alpha}$ as defined above. For every node $v \in V$, let $\mathcal{K}_v$ be the subgraph of $\mathcal{K}$ induced by the set V_v of nodes reachable from v. We put $\mathcal{K}, v \models (\mathbf{pfp}\ X_i : S)$ if, and only if, there is a stage α such that $X_i^{\alpha} \cap V_v = X_i^{\alpha+1} \cap V_v$ and $v \in X_i^{\alpha}$.*

In this definition, whether a node v is in the fixed point defined by ψ depends only on the part of the transition system reachable from it. With this modification, closure under bisimulation is again established. However, this semantics is also not satisfactory, as the following lemma demonstrates.

12.3 Lemma. *Let $\psi := (\mathbf{pfp}\ X : \varphi)$ be a formula in* MPC *under the semantics of Definition 12.2. On any string $\mathcal{T}_w$ of length n, if the fixed point of ψ on $\mathcal{T}_w$ is not empty, then it is reached within n steps.*

Proof. The proof is by induction on the length n of the string. The claim is trivial for $n = 1$. Now assume the claim has been established for all $0 < m < n$ and let $\mathcal{T}_w, v$ be a string of length n. Towards a contradiction, suppose that the fixed point of ψ on $\mathcal{T}_w$ exists but is not reached within n stages, i.e. $X^n \neq X^{n+1}$. By induction hypothesis, after $n-1$ stages the fixed point of the substring rooted at the successor of the root v has been reached. Thus, as $X^n \neq X^{n+1}$ either $v \in X^n - X^{n+1}$ or $v \in X^{n+1} - X^n$.

First suppose that $v \in X^n$ but $v \notin X^{n+1}$. Let U be the set of nodes reachable from the successor of v. By induction hypothesis, in the restriction to the nodes of U, the fixed point of φ is reached within $n-1$ steps and therefore $X^{n-1} \cap U = X^n \cap U = X^{n+1} \cap U$. Now, by assumption, $v \in X^n$ but $v \notin X^{n+1}$. It follows that $v \notin X^{n-1}$ and thus $X^{n+1} = X^{n-1} \neq X^n$. As φ is deterministic, the fixed point of φ does not exist on $\mathcal{K}$, contradicting the assumption.

The case where $v \notin X^n$ is analogous. □

A consequence of the proof is that on all strings where the fixed point of a formula $(\textbf{pfp}\ X : \varphi)$ exists, it is actually reached by an inflationary induction. Although this does not necessarily imply that MIC and MPC under this semantics are equivalent on strings, it shows that this is not what one would intuitively expect from a partial fixed-point semantics.

We now turn towards the alternative semantics for partial fixed-point logic as introduced in Chapter 7.

12.4 Definition (Semantics of MPC). *Let S be a system of rules defined as*

$$S := \begin{cases} X_1 & \leftarrow & \varphi_1(X_1, \dots, X_k) \\ & \vdots & \\ X_k & \leftarrow & \varphi_k(X_1, \dots, X_k) \end{cases}$$

and let $\psi := (\textbf{pfp}\ X_i : S)$ be a formula in MPC. *On any transition system $\mathcal{K}$ with universe V, the system S induces a sequence of stages defined as*

$$\begin{aligned} X_i^0 &:= \varnothing, \\ X_i^{\alpha+1} &:= [\![\varphi_i]\!]^{(\mathcal{K},\overline{X}^\alpha)} \\ X_i^\lambda &:= \text{final}((X_i^\alpha)_{\alpha<\lambda}) \qquad \textit{for limit ordinals } \lambda, \end{aligned}$$

where $\text{final}((X_i^\alpha)_{\alpha<\lambda})$ *denotes the set of elements $u \in V$ such that there is a $\beta < \lambda$ with $u \in X_i^\xi$ for all $\beta < \xi < \lambda$.*

For every node $u \in \mathcal{K}$ define $\mathcal{K}_u$ as the subgraph of $\mathcal{K}$ induced by the set V_u of nodes reachable from u. Obviously, for every $u \in V$, the sequence $(U^\alpha)_{\alpha \in \text{Ord}}$, with $U^\alpha := X_i^\alpha \cap V_u$, must eventually become cyclic. Let β_2 be minimal such that $U^{\beta_1} = U^{\beta_2}$ for some $\beta_1 < \beta_2$. We put

$$\mathcal{K}, u \models \psi \textit{ if, and only if, } u \in U^\xi \textit{ for all } \beta_1 \leq \xi < \beta_2.$$

As before, the problem whether a node u occurs in the fixed point of ψ is resolved only in terms of the part of $\mathcal{K}$ reachable from u. Although this might look unsatisfactory, it can in general not be avoided: Unlike inductive fixed points, partial fixed points do not always exist. Thus, a rule has to be agreed on for deciding when a stage of the induction is to be taken as the fixed point defined. Due to invariance under bisimulation this decision cannot be made on a global level, as the fixed point of a node cannot depend on parts of the transition system unreachable from it.

However, in restriction to finite transition systems, we can indeed impose a global condition for the partial fixed point of a formula. For this, let for each $u \in V$, m_u and n_u be minimal such that $m_u < n_u$ and $X^{m_u} \cap V_u = X^{n_u} \cap V_u$, where V_u is the set of nodes reachable from u. Now let m be the maximum of all such m_u and n be the product $n := \prod_{u \in V}(n_u - m_u)$. Then, $X_i^m = X_i^{m+n}$ and for all $u \in V$,

$$\mathcal{K}, u \models \psi \text{ if, and only if, } u \in X_i^{\xi} \text{ for all } m \leq \xi \leq m+n.$$

This proves the following lemma.

12.5 Lemma. *Let* $\psi := (\mathbf{pfp}\ X_i : S)$ *be as in Definition 12.4 and* $\overline{X}^{\alpha}$ *be the sequence of stages induced by* ψ *on a finite transition system* $\mathcal{K}$ *with universe* V*. Let* α *be minimal such that there is a* $\beta < \alpha$ *and* $\overline{X}^{\beta} = \overline{X}^{\alpha}$*. Then for all* $u \in V$*,* $\mathcal{K}, u \models \psi$ *if, and only if,* $u \in X_i^{\xi}$ *for all* $\beta \leq \xi \leq \alpha$*.*

Thus, for finite transition systems, we can take the straightforward adaptation of the partial fixed-point semantics in Chapter 7 as semantics for MPC. However, on infinite transition systems, this is not possible as there are examples where the lemma fails, i.e. there are infinite transition systems where the fixed point defined as in Definition 12.4 and the global fixed point as discussed above result in different sets. Without going into details, consider the transition system $\mathcal{K}$ defined as

$$\mathcal{K}: \quad \begin{tikzcd} & 1 \arrow[dl, <->] \arrow[dr, <->] \arrow[dd, <->] & \\ 0 \arrow[dr, <->] & & 3 \arrow[dl, <->] \\ & 2 & \end{tikzcd}$$

and a formula that, for finite ordinals, defines a sequence of stages alternating between $\{0, 1\}$ and $\{0, 2\}$ and on the stage containing only 0 defines the stage $\{3\}$ which repeats itself. On $\mathcal{K}$, the fixed point of the formula contains just $\{0\}$. However, if $\mathcal{K}$ is conjoint with a transition system on which the fixed point of the formula is only reached after infinitely many stages, the fixed point under the global semantics would no longer be $\{0\}$ but $\{3\}$.

From now on, let MPC denote the modal partial fixed-point logic under the semantics defined in Definition 12.4. We proceed by showing that MPC is contained in ML^∞ and thus is indeed a modal logic.

12.6 Lemma. *On classes of structures of bounded cardinality,* $\mathrm{MPC} \subseteq \mathrm{ML}^\infty$.

Proof. The proof is by induction on the structure of the formulae. We only consider the fixed-point rule, the other cases being trivial. Let $(\mathbf{pfp}\ X : \varphi(X))$ be a formula in MPC. For simplicity, we only give the proof for simple inductions. The general case is analogous.

For every ordinal α, we give a formula $\psi^\alpha \in \mathrm{ML}^\infty$ defining, over any transition system, the stage α of the induction on φ. The formulae ψ^α are inductively defined as follows.

$$\begin{array}{ll} \psi^0 & := \mathit{false}, \\ \psi^{\alpha+1} & := \varphi(X/\psi^\alpha), \\ \psi^\lambda & := \bigvee_{\alpha<\lambda} \bigwedge_{\alpha<\xi<\lambda} \psi^\xi \quad \text{for limit ordinals } \lambda. \end{array}$$

Here, $\varphi(X/\psi^\alpha)$ is obtained from φ by replacing every occurrence of X in φ by ψ^α. Obviously, for every transition system $\mathcal{K}$ with universe V and every node $u \in V$, $\mathcal{K}, u \models \psi^\alpha$ if, and only if, $u \in X^\alpha$.

What is left to be done is to give a formula defining the partial fixed point of φ. For this, note that if $\mathcal{C}$ is a class of structures such that the cardinality of every structure $\mathcal{K} \in \mathcal{C}$ is bounded by the cardinal ρ, then the partial fixed point of φ on any structure in $\mathcal{C}$ must be reached within at most $\lambda = 2^\rho$ stages. Thus, the fixed point of φ on any structure in $\mathcal{C}$ can be defined by the formula

$$\begin{array}{rl} \vartheta := & \bigvee_{0\le\beta<\alpha<\lambda}(\mathit{everywhere}(\psi^\alpha \leftrightarrow \psi^\beta) \land \bigwedge_{\beta\le\xi\le\alpha} \psi^\xi \land \\ & \bigwedge_{0\le\beta'<\alpha'<\alpha} \neg\mathit{everywhere}(\psi^{\alpha'} \leftrightarrow \psi^{\beta'})), \end{array}$$

where $\mathit{everywhere}(\varphi)$ is a ML^∞-formula equivalent to the formula defined in Proposition 11.5.

Let $\mathcal{K} \in \mathcal{C}$ be a transition system with universe V, u be a node in V, and U be the set of elements reachable from u. At the node u, ϑ expresses that there are stages $\beta < \alpha \le \lambda$ such that $U \cap X^\beta = U \cap X^\alpha$, u is contained in all stages X^ξ where $\beta \le \xi \le \alpha$, and there is no ordinal $\alpha' < \alpha$ such that for some $\beta' < \alpha'$, $U \cap X^{\alpha'} = U \cap X^{\beta'}$, i.e. α is the least stage where the induction on φ becomes cyclic.

It follows that for every $\mathcal{K} \in \mathcal{C}$ with universe V and every $u \in V$,

$$\mathcal{K}, u \models \vartheta \text{ if, and only if, } \mathcal{K}, u \models (\mathbf{pfp}\ X : \varphi).$$

□

The following corollary follows immediately.

12.7 Corollary. MPC *is closed under bisimulation and has the tree model property.*

We now give some examples for MPC-formulae. The first example demonstrates that on strings, MPC-inductions of exponential length are possible. Note the difference to Lemma 12.3, where it was proved that with the semantics from Definition 12.2 such inductions are impossible.

12.8 Example. *Let $\mathcal{T}_n$ be a string of length n such that the root of $\mathcal{T}_n$ is labelled by w and all other nodes are unlabelled. Consider the formula ψ defined as*

$$\psi := \mathbf{pfp}\ X : \begin{array}{l}(w \wedge X) \vee (X \wedge \mathit{somewhere}(\neg X)) \vee \\ (\neg X \wedge \Box \mathit{everywhere}(X))).\end{array}$$

On any string $\mathcal{T}_n$, ψ enumerates in its stages every subset of nodes from $\mathcal{T}_n$. However, once the root v of $\mathcal{T}_n$, labelled by w, is added to X, it stays in forever. Thus, the fixed point of ψ on $\mathcal{T}_n$ is just $\{v\}$. But before it is reached, an exponential number of stages is defined.

We present a more complicated example in the next section, where it is shown that the trace equivalence problem on unary alphabets is definable in MPC.

12.2 Expressive Power and Complexity

We first give a formal definition of the trace equivalence problem.

12.9 Definition (Trace equivalence problem). *Let Σ be an alphabet, i.e. a non-empty set of proposition symbols. Further, let $\mathcal{F}$ be a proposition symbol not in Σ.*

Let $\mathcal{K}, v$ be a finite rooted transition system. The set $\mathcal{T}(\mathcal{K})$ of traces in *$\mathcal{K}$ is defined as the set of words $w \in \Sigma^*$ such that there is a path labelled by w in $\mathcal{K}$ from v to a node in $\mathcal{F}^{\mathcal{K}}$. Two transition systems $\mathcal{K}, v$ and $\mathcal{K}', v'$ are* trace equivalent, *if $\mathcal{T}(\mathcal{K}) = \mathcal{T}(\mathcal{K}')$.*

The trace equivalence problem *($\mathcal{TE}$) is defined as the decision problem of the class*

$$\mathcal{TE}_\Sigma := \left\{ \mathcal{K}, v : \begin{array}{l} \mathcal{K}, v \textit{ is finite and for all } u, w \textit{ such that } v \to u \textit{ and} \\ v \to w, \mathcal{K}, u \textit{ and } \mathcal{K}, w \textit{ are trace equivalent} \end{array} \right\}.$$

The unary trace equivalence problem *is defined as the trace equivalence problem over unary alphabets, i.e. alphabets Σ with $|\Sigma| = 1$.*

The acyclic trace equivalence problem *is defined as the subclass of $\mathcal{TE}$ where all structures are acyclic.*

We show now that unary trace equivalence is definable in MPC.

12.10 Theorem. *Unary trace equivalence on arbitrary finite graphs is definable in* MPC*.*

Proof. Let ψ be the formula defined as

$$\mathbf{pfp}\; Z : \begin{cases} X & \leftarrow (\mathcal{F} \wedge \neg Y) \vee \Diamond X \\ Y & \leftarrow \mathcal{F} \\ Z & \leftarrow (\Diamond X \wedge \Diamond \neg X) \vee Z. \end{cases}$$

Clearly, X^1 contains all nodes contained in $\mathcal{F}$ and in the successive stages, X^n contains all nodes from which a path of length $n-1$ leads to a node in $\mathcal{F}$. Further, Z contains all nodes from which two paths of different length lead to a node in $\mathcal{F}$. In particular, the fixed point of Z contains the root of a transition system $\mathcal{K}, v$ if, and only if, $\mathcal{K}, v \notin \mathcal{TE}$. Thus, $\mathcal{K}, v \models \neg\psi$ if, and only if, $\mathcal{K}, v \in \mathcal{TE}$. □

In Section 14.4, we will prove that unary trace equivalence in not definable in MIC. Also, it is known that the problem is co-NP-complete (see [GJ79, A10.1 page 265]). Finally, it is easily seen that MIC $\subseteq$ MPC. From this, the following corollary follows immediately.

12.11 Corollary. *(i)* MIC *is strictly contained in* MPC*.*

(ii) MPC *is not contained in* L^ω_μ*, provided that* PTIME $\neq$ NP*.*

(iii) There are NP*-complete problems definable in* MPC*.*

We now aim at establishing complexity bounds for the computational problems for MPC. As MIC $\subseteq$ MPC, the undecidability of the satisfiability problem for MIC applies to MPC as well. Thus, we immediately get the following theorem.

12.12 Theorem. *The satisfiability problem for* MPC *is undecidable, it is not even in the arithmetical hierarchy.*

Finally, we show that model checking for MPC is PSPACE-complete.

12.13 Theorem. *The model-checking problem for* MPC *is complete for* PSPACE*.*

Proof. In Section 11.2, we have already seen that the model checking problem for MIC is PSPACE-complete (Theorem 11.17). From this, PSPACE-hardness for MPC model checking follows immediately.

To show membership in PSPACE, note that every stage of an induction on ($\mathbf{pfp}\; X : \varphi$) requires only a linear amount of space. Let $\mathcal{K}$ be a transition system with n nodes. To compute the partial fixed point of φ, compute the first 2^n stages. Clearly, after that many stages, the induction must be cyclic.

Now proceed as follows. First, make three copies U, V, W of the stage $X^{(2^n)}$. The first copy, U, is left untouched in the following computation. In the space consumed by the second copy, V, the induction is continued, i.e. the next induction stages are computed. At each step, the elements that are not in the current stage of V are removed from the third copy, W. As soon as V equals $X^{(2^n)}$, which is stored in U, the computation is stopped. Now, W contains precisely those elements, which occur in every stage of the cycle and therefore contains the fixed point of the induction on φ.

For this, space is needed to store three stages of the induction on φ, i.e. space for three subsets of the universe. Further, some space is needed for the computation of φ on every stage. A simple induction on the number of fixed-point operators in a MPC-formula establishes the claim. $\Box$

Chapter 13

Labelling Indices on Acyclic Transition Systems

In the preceding chapters we introduced a variety of modal fixed-point logics. There are various techniques that may be used to compare their expressive power. Showing that one logic is at least as expressive as another can often be done by giving an explicit translation of formulae of the first to formulae of the second. Trivial examples of this are the inclusion of LFP in IFP and of L_μ in MIC. A more complicated example is given in the translation of IFP into LFP in Chapter 8.

Establishing separations between logics is often more involved. This requires identifying a property expressible in one logic and showing that it is not expressible in the other. Many specialised techniques have been developed for such proofs of inexpressibility: One may consider diagonalisation arguments similar to the one we used in Chapter 7 to separate PFP and IFP. Such methods have also been used by Bradfield to establish that increasing alternations of least and greatest fixed points in the μ-calculus yield greater expressive power [Bra98a]. Another important method are variants of Ehrenfeucht-Fraïssé games, for instance bisimulation games. One can also relate logics to finite automata, allowing for the use of methods such as the pumping lemma.

In this chapter, an alternative complexity measure for modal properties of finite structures, called the *labelling index* of the property, is introduced. We will use this measure to analyse the expressive power of various fixed-point logics considered so far, including the modal logics introduced in the previous chapters. An extended abstract of the results reported in this and the following chapter appeared in [DK02].

The notion of labelling index generalises the concept of the automaticity of languages, introduced by Shallit and Breitbart in [SB96]. The idea is to classify languages not in terms of the automata model or type of grammar needed to accept the whole language but in terms of the growth rates of automata accepting the fragments of the language of words up to a fixed length.

13.1 Definition (Automaticity). *Let Σ be a non-empty alphabet and $\mathcal{L} \subseteq \Sigma^*$ be a language over Σ. For every $n < \omega$ let $\mathcal{L}^{\leq n}$ be the fragment of $\mathcal{L}$ containing all words in $\mathcal{L}$ of length at most n, i.e.*

$$\mathcal{L}^{\leq n} := \{w \in \mathcal{L} : |w| \leq n\}.$$

Clearly, for every $n < \omega$ there is a deterministic finite automaton $\mathcal{A}_n$ that agrees with $\mathcal{L}$ on all words of length at most n, i.e. accepts a word w of length at most n if, and only if, $w \in \mathcal{L}^{\leq n}$. The automaticity *of $\mathcal{L}$ is defined as the function $f : \omega \to \omega$ that assigns to each $n < \omega$ the number of states in a minimal deterministic automaton that accepts a word of length at most n if, and only if, it is in $\mathcal{L}^{\leq n}$.*

Obviously, the regular languages are precisely the languages with constant automaticity. Further, every language $\mathcal{L}$ has at most exponential automaticity. For this, simply let the automata $\mathcal{A}_n$ accepting $\mathcal{L}^{\leq n}$ have one state for each prefix of a word in Σ^n.

The language $\mathcal{L} := \{cwdwd : w \in \{a,b\}^*\}$ over the alphabet $\{a,b,c,d\}$ proved in Section 11.3 to be definable in MIC has exponential automaticity. For, suppose there was for some $n < \omega$ a deterministic automaton $\mathcal{A}_n$ which accepts $\mathcal{L}^{\leq n}$ but has less than $2^{\frac{n-3}{2}}$ states. Let $w := cvdvd$ be a word of length n. Clearly, there are $2^{\frac{n-3}{2}}$ different words cv. As there are less than $2^{\frac{n-3}{2}}$ nodes in $\mathcal{A}_n$, there must be two different words cv_1 and cv_2 such that $\mathcal{A}_n$, starting from its initial state, ends in the same state when reading cv_1 and cv_2. Thus, $\mathcal{A}_n$ accepts cv_1dv_1d if, and only if, it accepts cv_2dv_1d, a contradiction to $\mathcal{A}_n$ accepting $\mathcal{L}^{\leq n}$.

The example demonstrates, that the concept of automaticity on words is not helpful in analysing the expressive power of the modal fixed-point logics introduced so far. The languages definable in L_μ all have constant, i.e. minimal, automaticity, whereas the languages definable in MIC already might have exponential, and thus maximal, automaticity.

In the next two chapters, we aim at extending the concept of automaticity from words to modal properties of arbitrary transition systems. This extension consists of two somewhat independent parts. First, we introduce a measure of size or complexity of transition systems, called their *rank*, that generalises the length of a word. We then introduce an automata-like device that we call *labelling system.* It is shown that every class of transition systems can be defined up to a fixed rank by a labelling system.

For every class $\mathcal{C}$ of transition systems, the function that takes every $n < \omega$ to the size of a smallest labelling system, that agrees with $\mathcal{C}$ on all structures of rank at most n, is called the *labelling index* of $\mathcal{C}$.

By deriving a number of separation results from it, we demonstrate that this is a useful measure for the complexity of classes of finite transition systems. In particular, we will obtain separation results for the modal logics introduced so far.

We also consider the relationship between the labelling index of a class $\mathcal{C}$ and conventional time and space based notions of its complexity. Finally, we investigate the labelling index of the trace equivalence problem over specific classes of structures and derive results about its expressibility in modal logics.

Conceptually, the labelling index is a notion on finite structures. Therefore we agree on the following proviso.

Proviso. All structures in the following two chapters are finite. □

As it turns out, the framework of labelling systems as described above is smoother when applied to classes of acyclic structures than if cycles are allowed. Therefore, we first introduce the concepts on the class of acyclic finite structures, i.e. those bisimilar to finite trees, and generalise it to classes of arbitrary structures in the next chapter.

13.1 The Rank of Trees

The framework we are going to describe consists of two somewhat independent parts, namely the definition of a measure of finite structures, called the *rank* of the structure, and the definition of labelling systems accepting classes of structures up to a given rank. We first define the rank of acyclic structures, which, in this case, simply is the height of a tree or, in the case of acyclic structures which are no trees, the height of the trees bisimilar to it.

13.2 Definition. *Let $\mathcal{K}$ be an acyclic transition system with root v. The* rank of $\mathcal{K}, v$, *in terms* $\text{rank}(\mathcal{K}, v)$ *or simply* $\text{rank}(\mathcal{K})$ *if v is understood, is defined as $h(v) + 1$, i.e. the rank is one plus the height $h(v)$ of v in $\mathcal{K}$, or equivalently, the height of a tree bisimilar to $\mathcal{K}, v$ plus one.*

We are only interested in modal, i.e. bisimulation invariant properties of structures. As any acyclic structure is bisimilar to a tree of the same rank, we can restrict attention to trees.

The key feature that makes the height of a tree interesting for the further investigations is, that it also binds the length of least and inflationary fixed-point inductions. We make this precise in the following lemma.

13.3 Lemma. *Let k*MIC *denote the class of formulae in* MIC *where every system of formulae defining a fixed-point induction has at most k rules. For any tree $\mathcal{T}$ with root v,*

$$k \cdot \mathit{rank}(\mathcal{T}) = \text{cl}_{k\text{MIC}}(\mathcal{T}),$$

where $\mathrm{cl}_{k\mathrm{MIC}}(\mathcal{T})$ *denotes the supremum of all* α *such that there is a system* S *of at most* k *formulae for which* α *is the least ordinal with* $S^\alpha = S^{\alpha+1}$.

Proof. Recall that the rank of a tree is simply its height plus one. Clearly, there is a MIC-induction of length precisely $k \cdot \mathrm{rank}(\mathcal{T})$, e.g. take the system S_k containing for each $0 \le i \le k-1$ a rule $X_i \leftarrow \Box(\bigwedge_{j=0}^{k-1} X_j) \wedge \bigwedge_{j=0}^{i-1} X_j$. Then, for all α, $X_i^{k\cdot\alpha+i}$ contains the nodes of height less than α.

The proof of the converse is by induction on the height of $\mathcal{T}$. For trees of height 0, i.e. consisting of one node only, the closure ordinal of any kMIC-formula must be at most k. For, in each stage of the induction on a system with k rules, the single node in the tree must be added to at least one of the k variables defined by S.

Now assume the height of $\mathcal{T}$ is $n > 1$. By bisimulation invariance, whether a node u enters the fixed point of S at some stage depends only on the subtree rooted at u and not on the other nodes in $\mathcal{T}$ not reachable from it. Thus, by induction hypothesis, we get that the fixed point of S restricted to the successors of the root v of $\mathcal{T}$ must be reached within at most $k(n-1)$ steps. Consequently, the fixed point of S on $\mathcal{T}$ is reached in at most $k \cdot n$ steps. □

The lemma also gives a justification to the definition of the rank of a tree as its height *plus one.*

13.2 Labelling Systems

We are now ready to describe the second part of the framework of labelling indices, the labelling systems. There is a close correspondence between labelling systems and bottom-up tree automata. In fact, on acyclic structures, labelling systems are nothing else than bottom-up tree automata. However, as we are interested in a generalisation to arbitrary structures later on, we introduce labelling systems in full generality. Note that, contrary to the rest of this chapter, the structures considered here may have cycles!

13.4 Definition. *A* labelling system $\mathcal{L}$ *is a quintuple* $\mathcal{L} := (Q, \mathcal{A}, \mathcal{P}, \delta, \mathcal{F})$, *where*

- Q *is a finite set of labels,*
- $\mathcal{A}$ *is a finite set of actions,*
- $\mathcal{P}$ *is a finite set of proposition symbols,*
- $\mathcal{F} \subseteq Q$ *is a set of* accepting labels*, and*
- δ *is a total function* $\delta : 2^{Q\times\mathcal{A}} \times 2^{\mathcal{P}} \rightarrow Q$*, the* transition function*.*

For every transition system $\mathcal{K} := (V, (E_a)_{a\in\mathcal{A}}, (p)_{p\in\mathcal{P}})$ and node $v \in V$, the labelling system $\mathcal{L}$ accepts *$\mathcal{K}, v$, in terms $\mathcal{K}, v \models \mathcal{L}$, if, and only if, there is a function $f : V \to Q$ such that $f(v) \in \mathcal{F}$ and for each $u \in V$,*

$$f(u) = \delta(\mathrm{Succ}(u), \mathrm{Prop}(u))$$

where

$$\begin{array}{rcl} \mathrm{Succ}(u) & := & \{(f(u'), a) : a \in \mathcal{A} \textit{ and } (u, u') \in E_a^{\mathcal{K}}\} \quad \textit{and} \\ \mathrm{Prop}(u) & := & \{p : p \in \mathcal{P} \textit{ and } u \in p^{\mathcal{K}}\}. \end{array}$$

The class of transition systems defined *or* accepted *by $\mathcal{L}$ is defined as the class $\mathcal{C}$ of all transition systems $\mathcal{K}, v$ such that $\mathcal{K}, v \models \mathcal{L}$.*

Note that, as δ is functional, labelling systems are deterministic devices. And indeed, on well-founded trees, they are equivalent to deterministic bottom-up tree automata. On the other hand, if the structures may contain cycles, some form of nondeterminism is present as acceptance is defined in terms of the *existence* of a labelling. Thus, for a given structure and a given labelling system, there may be more than one labelling function witnessing the fact that $\mathcal{L}$ accepts $\mathcal{K}, v$.

The class of structures accepted by a labelling system is not necessarily closed under bisimulation. This can be seen in the following simple example.

13.5 Example. *Consider the labelling system $\mathcal{L} = (Q, \mathcal{A}, \mathcal{P}, \delta, \mathcal{F})$ given by*

- *$Q = \{q_{\text{even}}, q_{\text{odd}}\}$,*
- *$\mathcal{A} = \{a\}$ and $\mathcal{P} = \varnothing$,*
- *$\mathcal{F} = \{q_{\text{even}}\}$ and where*
- *the function δ is given by the rules $\delta(\varnothing) = q_{\text{even}}$, $\delta(\{(q_{\text{even}}, a)\}) = q_{\text{odd}}$, $\delta(\{q_{\text{odd}}, a\}) = q_{\text{even}}$ and $\delta(\{(q_{\text{even}}, a), (q_{\text{odd}}, a)\}) = q_{\text{even}}$, where we have dropped the second argument of δ as it is always $\varnothing$.*

This labelling system accepts a simple cycle if, and only if, it is of even length.

As we are interested in labelling systems that define bisimulation-closed classes of structures, we consider the following definition.

13.6 Definition. *A labelling system $\mathcal{L}$ is* $\sim$-consistent*, if for all transition systems $\mathcal{K}, v$, whenever $\mathcal{K}, v \models \mathcal{L}$ then there is a labelling f witnessing this such that for all u, u', if $\mathcal{K}, u \sim \mathcal{K}, u'$ then $f(u) = f(u')$.*

Another natural condition is obtained by requiring the class of structures defined by $\mathcal{L}$ to be closed under bisimulation.

13.7 Definition. *A labelling system $\mathcal{L}$ is* $\sim$-invariant *if, whenever $\mathcal{K}, v \models \mathcal{L}$ and $\mathcal{K}, v \sim \mathcal{K}', v'$ then $\mathcal{K}', v' \models \mathcal{L}$.*

As it happens, these two definitions are equivalent for the structures that are of interest to our investigations.

13.8 Lemma. *On connected structures, a labelling system is $\sim$-consistent if, and only if, it is $\sim$-invariant.*

Proof. Suppose $\mathcal{L} = (Q, \mathcal{A}, \mathcal{P}, \delta, \mathcal{F})$ is $\sim$-consistent. We show that $\mathcal{L}$ is $\sim$-invariant. Let $\mathcal{K}, v$ be a transition system such that $\mathcal{K}, v \models \mathcal{L}$ and let $f : V \to Q$ be a labelling witnessing this, such that $f(u) = f(u')$ whenever $\mathcal{K}, u \sim \mathcal{K}, u'$. For every structure $\mathcal{K}'$ with universe V' such that $\mathcal{K}', v' \sim \mathcal{K}, v$ define a labelling $f' : V' \to Q$ by $f'(u') := f(u)$ where $u \in V$ is a node such that $\mathcal{K}', u' \sim \mathcal{K}, u$. It is easily verified that f' is a labelling witnessing that $\mathcal{K}', v' \models \mathcal{L}$.

Towards the converse, suppose $\mathcal{L} = (Q, \mathcal{A}, \mathcal{P}, \delta, \mathcal{F})$ is $\sim$-invariant and $\mathcal{K}, v \models \mathcal{L}$. Consider the quotient $\mathcal{K}_{/\sim}$ of $\mathcal{K}$ under $\sim$ as defined in Definition 9.6. Since $\mathcal{K}_{/\sim}, [v] \sim \mathcal{K}, v$, we have, by the $\sim$-invariance of $\mathcal{L}$, that $\mathcal{K}_{/\sim}, [v] \models \mathcal{L}$. Let f be a labelling that witnesses this and define $f' : V \to Q$ by $f'(u) = f([u])$. It is clear that f' is a valid labelling of $\mathcal{K}, v$, and it satisfies the condition that if $\mathcal{K}, u \sim \mathcal{K}, u'$ then $f(u) = f(u')$. □

As we have seen, $\sim$-consistent labelling systems define $\sim$-invariant classes of structures. However, not every bisimulation-closed class $\mathcal{C}$ of structures is definable by a labelling system. But, as we shall see below, $\mathcal{C}$ is defined by a family of systems. In order to define the family we use the rank of a structure as a measure of its size.

13.9 Definition. *A bisimulation closed class $\mathcal{C}$ of structures is* accepted *by a family $\mathfrak{L} := \{\mathcal{L}_n : n < \omega\}$ of labelling systems if for every transition system $\mathcal{K}, v$ of rank n, $\mathcal{K}, v \in \mathcal{C}$ if, and only if, $\mathcal{K}, v \models \mathcal{L}_m$ for all $m \geq n$. $\mathcal{C}$ is called the class of structures* defined *by $\mathfrak{L}$.*

Every bisimulation closed class of transition systems can be accepted by a family of labelling systems as follows. For this, note that the rank[1] is trivially bounded by the size of the transition system and that there are only finitely many bisimulation equivalence classes of structures of size at most n. Taking a state for each such class yields the desired labelling system.

13.10 Lemma. *Let $\mathcal{C}$ be a bisimulation closed class of finite structures. For every n there is a $\sim$-consistent labelling system $\mathcal{L}_n$ such that for any structure $\mathcal{K}$ with rank$(\mathcal{K}) \leq n$, $\mathcal{L}_n$ accepts $\mathcal{K}$ if, and only if, $\mathcal{K} \in \mathcal{C}$.*

[1]So far, we have only defined the rank of acyclic structures. The rank of arbitrary structures will be defined in Chapter 14 below. For now, it suffices to know that the rank is bounded by the number of elements in the structure.

Proof. For a fixed n, there are only finitely many bisimulation equivalence classes. We obtain a labelling system by taking a state for each bisimulation class of a structure of rank at most n and defining, for sets $M \subseteq Q \times \mathcal{A}$ and $P \subseteq \mathcal{P}$, $\delta(M, P)$ to be the bisimulation class of the structure consisting of a root labelled by the propositions in P, which has for each $(q, a) \in M$ an a-successor such that the structure rooted at this successor is in the bisimulation class represented by q. Define $\mathcal{F}$ as the set of states representing the bisimulation class of a structure in $\mathcal{C}$. The lemma now follows immediately. □

We close the section by relating labelling systems to other automata models on graphs considered in the literature.

Connection between Labelling Systems and Automata. Automata models on graphs have been investigated by several authors before. The usual way to define such automata is in terms of tiling systems (see [Tho91, Tho94]). A tiling system consists of a set Q of states and a set of finite graphs - called tiles - labelled by states from Q. A graph $\mathfrak{G}$ is accepted by a tiling system if it can be labelled by states from Q such that the resulting labelled graph can entirely be covered by overlapping tiles.

By definition, tiling and labelling systems seem closely related and indeed we could rephrase the definitions above in terms of tiling systems. But there are subtle differences. Labelling systems are by definition deterministic whereas tiling systems are usually assumed non-deterministic. Further, the label a node gets in a run of a labelling system depends only on its successors and not on its predecessors. Whereas this is a natural requirement for devices defining bisimulation invariant properties, this would be unusual and artificial for tiling systems. Finally, the notion of accepting states is alien to tiling systems.

Therefore we refrained from calling our devices automata or tiling systems but used the term labelling system instead.

13.3 Labelling Indices of Modal Logics

In this section, we aim at establishing upper and lower bounds on the labelling index of classes of acyclic structures definable in modal logics such as ML and its various fixed-point extensions. As noted above, we focus exclusively on classes of acyclic structures. The case where structures may contain cycles is studied in the next chapter.

13.3.1 Modal Logic and the Modal μ-Calculus

It follows immediately from the results of Thatcher and Wright [TW68] and Doner [Don70] that any class of trees definable in MSO can be accepted by a deterministic bottom-up tree automata. As on trees, tree automata and labelling systems are equivalent, this implies that any bisimulation-closed class of trees which is definable in MSO can be accepted by one fixed labelling system, independent of the rank of the structures within this class. This proves the following theorem.

13.11 Theorem. *Any bisimulation-closed class of acyclic structures definable in* MSO *or the modal μ-calculus has constant labelling index.*

The following corollary is immediate.

13.12 Corollary. *Any bisimulation-closed class of acyclic structures definable in modal logic has constant labelling index.*

Of course, the constant labelling index of ML-definable classes of transition systems is not limited to acyclic structures but applies to arbitrary structures as well.

However, we shall see below, that the constant bound on the labelling index of L_μ-definable classes of acyclic transition systems does not extend to arbitrary structures. See Section 14.3 for details. We now turn to the labelling index of classes of structures definable in the modal iteration calculus.

13.3.2 The Modal Iteration Calculus

The aim of this section is to show that every class of acyclic transition systems definable in MIC has at most exponential labelling index. Moreover, there are classes of structures definable in MIC that have an exponential lower bound on their labelling index.

Let φ be a formula in MIC and let Φ be the set of sub-formulae of φ. Further, let $X_1, \dots, X_k$ be the fixed-point variables occurring in φ. W.l.o.g. we assume that no fixed-point variable is bound twice in φ and if $i < j$ then either X_j does not occur free in the formula defining X_i in φ or X_i and X_j are bound together in a simultaneous induction.

Let $\overline{\imath} := i_1, \dots, i_k$ be a tuple of finite ordinals. If ψ is a sub-formula of φ and $\mathcal{K}, v$ is a transition system, we write $(\mathcal{K}, \overline{X}^{\overline{\imath}}), v \models \psi$ to indicate that ψ holds at the node v if each free fixed-point variable X_j in ψ is interpreted by the stage $X_j^{i_j}$ of the induction on $\mathcal{K}$.

13.13 Definition (φ-types). *For every transition system $\mathcal{K}, v$ of rank n we define the* φ-type of $\mathcal{K}, v$ *as the function $f : \{0, \dots, k \cdot n\}^k \to 2^\Phi$ such that for every formula $\psi \in \Phi$, ψ occurs in $f(\overline{\imath})$ if, and only if, $(\mathcal{K}, \overline{X}^{\overline{\imath}}), v \models \psi$ and further, if X_j is bound by a sub-formula of ψ, then $i_j = kn$.*

A function $f : \{0, \dots, k \cdot n\}^k \to 2^\Phi$ is a φ-type *if it is a φ-type of a transition system.*

We use the notion of φ-types to define for each formula $\varphi \in$ MIC a family of labelling systems accepting precisely those acyclic structures which satisfy φ.

13.14 Definition. *Let φ be a formula in* MIC. *For every $n \in \omega$ define the labelling system $\mathcal{L}_\varphi(n) := (Q, \mathcal{A}, \mathcal{P}, \delta, \mathcal{F})$ as follows.*

- *Q consists of all φ-types of acyclic structures $\mathcal{K}, v$ of rank at most n.*
- *For sets $M \subseteq Q \times \mathcal{A}$ and $P \subseteq \mathcal{P}$ define $\delta(M, P)$ as the φ-type of a transition system $\mathcal{K}, v$ where*
  - (i) *at the root v, exactly the propositions in P are true,*
  - (ii) *for each pair $(q', a) \in M$ there is an a-successor of v whose φ-type is q', and*
  - (iii) *v has no further successors.*
- *Finally, $\mathcal{F} \subseteq Q$ is the set of labels q such that $\varphi \in q(\overline{k \cdot n})$.*

13.15 Lemma. *For every formula $\varphi \in$ MIC, every structure $\mathcal{K}, v$ of rank n, and every $m \geq n$, $\mathcal{L}_\varphi(m)$ accepts $\mathcal{K}, v$ if, and only if, $\mathcal{K}, v \models \varphi$.*

Proof. Let $\mathcal{K}, v$ be an acyclic structure of rank n and let $m \geq n$. By Lemma 13.3, on $\mathcal{K}, v$, the closure ordinal of any fixed-point induction of a sub-formula of φ is at most kn.

Suppose $\mathcal{K}, v \models \varphi$. We construct a labelling f witnessing that $\mathcal{L} := \mathcal{L}_\varphi(m)$ accepts $\mathcal{K}, v$. For every node $u \in \mathcal{K}$ reachable from v let $f(u)$ be the φ-type of $\mathcal{K}, u$. It is an immediate consequence of the definition of φ-types and the labelling systems $\mathcal{L}_\varphi(m)$ that this labelling is consistent with the transition function, i.e is a labelling indeed. As $\mathcal{K}, v \models \varphi$, it follows that $\varphi \in f(v)(\overline{k \cdot n})$ and thus $\mathcal{L}$ accepts $\mathcal{K}, v$.

Towards the converse, let f be a labelling witnessing that $\mathcal{K}, v \models \mathcal{L}$. Let V be the universe of $\mathcal{K}$. We claim that for all $u \in V$, if $q := f(u)$ is the φ-type f assigns to u, then for all $\bar{\imath} \in \{0, \dots, k \cdot m\}^k$ and all $\psi \in \Phi$ we have $\psi \in q(\bar{\imath})$ if, and only if, $(\mathcal{K}, \overline{X}^{\bar{\imath}}), u \models \psi$.

The claim is proved simultaneously for all nodes by induction on the lexicographical ordering on $\bar{\imath}$ and the structure of the formulae $\psi \in q(\bar{\imath})$.

- The case of atomic formulae $p \in \mathcal{P}$ follows immediately from Part (i) of Definition 13.14.
- The case of Boolean connectives is trivial.
- Now suppose $\psi := \langle a \rangle \vartheta$. If u is a leaf, then ψ must be false at u. By definition of the transition function, in particular Part (iii), q is the φ-type of a transition system $\mathcal{S}, v'$, such that v' has no successor. It follows that $\psi \notin q(\bar{\imath})$.

Suppose u is no leaf. Let $M \subseteq Q \times \mathcal{A}$ be such that $(q', a) \in M$ if, and only if, there is an a-successor of u labelled by q'. By definition, q is the φ-type of some transition system $\mathcal{S}, s$ such that for each $(q', a) \in M$, there is an a successor of s in $\mathcal{S}$ whose φ-type is q'. Now, if $\psi \in q(\overline{\imath})$, then there must be a pair $(q', a) \in M$ such that $\vartheta \in q'(\overline{\imath})$. Let u' be an a-successor of u whose label is q'. By induction on the structure of the formulae, this implies that $(\mathcal{K}, \overline{X}^{\overline{\imath}}), u' \models \vartheta$ and thus $(\mathcal{K}, \overline{X}^{\overline{\imath}}), u \models \psi$.

Conversely, if $(\mathcal{K}, \overline{X}^{\overline{\imath}}), u \models \psi$, then there is some a-successor u' of u such that $(\mathcal{K}, \overline{X}^{\overline{\imath}}), u' \models \vartheta$. Let q' be the label of u'. By induction on the structure of the formulae, this implies that $\vartheta \in q'(\overline{\imath})$ and therefore $\psi \in q(\overline{\imath})$.

- The case for $\psi := [a]\vartheta \in \Phi$ is analogous.

- Now, let $\psi := X_j$ be an atom where X_j is one of the fixed-point variables occurring in φ. Suppose X_j is bound in a system S that also binds $X_{r_1}, \dots, X_{r_l}$.

  If $\overline{\imath} = \overline{0}$, then, since $X_j^0 = \varnothing$ for all j, $(\mathcal{K}, \overline{X}^{\overline{\imath}}), u \not\models X_j$ and also, as q is a φ-type of some transition system, $X_j \notin q(\overline{\imath})$.

  If $\overline{\imath} > \overline{0}$ then, since q is the φ-type of some transition system, this implies that there is a tuple $\overline{\imath}' < \overline{\imath}$ which agrees with $\overline{\imath}$ on all positions except $j, r_1, \dots, r_l$, such that $\varphi_j \in q(\overline{\imath}')$, where φ_j is the defining formula of X_j. By induction on the stages, this implies that $(\mathcal{K}, \overline{X}^{\overline{\imath}'}), u \models \varphi_j$ and therefore $(\mathcal{K}, \overline{X}^{\overline{\imath}}), u \models X_j$ by the inflationary semantics of the **ifp**-operator.

  The converse is analogous.

- Finally, suppose $\psi := (\mathbf{ifp}\ X_{r_1} : S)$, where

$$S := \begin{cases} X_{r_1} & \leftarrow \quad \varphi_{r_1}(X_{r_1}, \dots, X_{r_l}) \\ & \vdots \\ X_{r_l} & \leftarrow \quad \varphi_{r_l}(X_{r_1}, \dots, X_{r_l}) \end{cases}$$

  is a system of formulae, and $\psi \in q(\overline{\imath})$. W.l.o.g. assume that $r_1 < r_2 < \dots < r_l$. By the definition of φ-types, for all $1 \le j \le l$, $i_{r_j} = km$, as the variables $X_{r_1}, \dots, X_{r_l}$ are bound by ψ. Further, q is the φ-type of some transition system and therefore there is a sequence of stages $\overline{X}^{\overline{\imath}'}$ such that

  - $i_j = i'_j$, for all $j \ne r_s$, where $1 \le s \le l$,
  - $i'_{r_s} < i_{r_s}$ for all $1 \le s \le l$, and
  - $\varphi_{r_1} \in q(\overline{\imath}')$.

Thus $\bar{\imath}' < \bar{\imath}$ and, by induction on the stages, we get that $(\mathcal{K}, \overline{X}^{\bar{\imath}'}), u \models \varphi_{r_1}$ and therefore $(\mathcal{K}, \overline{X}^{\bar{\imath}}), u \models \psi$.

Again, the converse is analogous.

We have shown that for all nodes u labelled by $f(u) = q$, all $\psi \in \Phi$, and all $\bar{\imath} \in \{0, \dots, k \cdot m\}^k$,

$$\psi \in q(\bar{\imath}) \text{ if, and only if, } (\mathcal{K}, \overline{X}^{\bar{\imath}}), u \models \psi.$$

Since $\mathcal{L}$ accepts $\mathcal{K}, v$ we have that $\varphi \in f(v)(\overline{k \cdot n})$ and therefore $\mathcal{K}, v \models \varphi$. This finishes the proof of the lemma. □

The following theorem follows immediately.

13.16 Theorem. MIC *has exponential labelling index on acyclic structures in the sense that every* MIC *definable class of acyclic transition systems has at most exponential labelling index and there are classes of structures which are definable in* MIC *but have an exponential lower bound on their labelling index.*

Proof. In Lemma 13.15 we have proved that the family of labelling systems in Definition 13.14 defined for a formula φ accepts the class of all transition systems satisfying φ. Clearly, for every n, the size of $\mathcal{L}_\varphi(n)$ is exponential in n. From this, the upper bound follows immediately.

The lower bound follows from the fact that automaticity and labelling index coincide on classes of words. As we have seen, there are languages with an exponential automaticity definable in MIC. □

As ML-formulae can be seen as MIC-formulae without any fixed-point operators, the number of φ-types for a ML-formula φ depends only on φ and is therefore constant. Thus, the proof of the lemma above also yields a direct proof for Corollary 13.12.

13.3.3 Higher Dimensional μ-Calculus

Recall from Chapter 10 the definition of the higher dimensional μ-calculus. By Theorem 10.9, this logic characterises precisely the class of bisimulation invariant properties decidable in polynomial time. Thus, the bisimulation relation itself is definable and therefore the class $\mathcal{C}$ of transition systems $\mathcal{K}, v$ where all successors of the root v are bisimilar. Obviously, the number of pairwise non bisimilar trees of height at most n is non-elementary in n. This is clear for $n = 0$ and, by induction, for $n > 0$ we get that there are $(n-1)$-fold exponentially many trees of height $n-1$ which are pairwise not bisimilar. Thus, as the root of a tree of height at most n may have any set of trees of height at most $n-1$ rooted at its successors, we get that there are n-fold exponentially many trees of height at most n which are pairwise not bisimilar. Thus the class $\mathcal{C}$ has a non-elementary labelling index. This proves the following lemma.

13.17 Lemma. *There are classes of structures definable in L_μ^ω which have a non-elementary labelling index.*

The next corollary follows immediately.

13.18 Corollary. MIC *is a strict subset of L_μ^ω.*

13.4 Monadic Inflationary Fixed-Point Logic

In the previous section we showed that there is a close correspondence between modal fixed-point logics and particular labelling indices. We will show that this correspondence does not extend to monadic fixed-point extensions of first-order logic. In particular we will show that for any elementary function f there is a class of transition systems definable in monadic inflationary fixed-point logic whose labelling index dominates f.

The properties we are going to construct, that are definable in M-IFP and have high labelling index, are based on the use of trees to encode integers in a number of ways of increasing complexity. To be precise, for each natural number k, we inductively define an equivalence relation $\simeq_k$ on trees as follows.

13.19 Definition. *For any two trees $\mathcal{T}$ and $\mathcal{S}$ with root t and s resp., write*

- *$\mathcal{T} \simeq_0 \mathcal{S}$ just in case that $\mathcal{T}$ and $\mathcal{S}$ have the same height and*
- *$\mathcal{T} \simeq_{k+1} \mathcal{S}$ just in case the set of $\simeq_k$-equivalence classes of the subtrees rooted at the children of t is the same as the set of $\simeq_k$-equivalence classes of the subtrees rooted at the children of s.*

By abuse of notation, we will also think of these relations as relations on the nodes of a tree $\mathcal{T}$. In this case, by $u \simeq_k v$ we mean $\mathcal{T}_u \simeq_k \mathcal{T}_v$ where $\mathcal{T}_u$ and $\mathcal{T}_v$ are the trees rooted at u and v respectively.

13.20 Lemma. *The number of distinct $\simeq_k$ equivalence classes of trees of height $n + k$ or less is k-fold exponential in n.*

Proof. The proof is by induction on k. Clearly, the number of $\simeq_0$ equivalence classes of trees of height n or less is n. Furthermore, let N be the set of $\simeq_k$-equivalence classes of trees of height at most $n + k$. For every subset M of N, we can construct a tree $\mathcal{T}_M$ of height at most $n + k + 1$ consisting of a root v and for each class $\mathcal{C}$ in M a tree $\mathcal{T} \in \mathcal{C}$ rooted at a successor of v. For every $M_1, M_2 \subseteq N$ such that $M_1 \neq M_2$, the trees $\mathcal{T}_{M_1}$ and $\mathcal{T}_{M_2}$ are $\simeq_{k+1}$-inequivalent. Thus, there are $2^{|N|}$ different $\simeq_{k+1}$-equivalence classes. □

We use this to prove the next theorem, which shows that there is no elementary upper bound on the labelling index of classes of trees definable in M-IFP.

13.21 Theorem. *For every elementary function f, there is a property with labelling index $\Omega(f)$ definable in* M-IFP.

Proof. We first show by induction on k that the equivalence relation $\simeq_k$ is definable by a formula $\vartheta(u,v) \in$ M-IFP.
Basis. Let $\psi(x)$ be defined as

$$\begin{aligned}\psi(x;u,v) := [\mathbf{ifp}_{X,x} \quad & ((x \neq u \wedge x \neq v \wedge \forall y\,(Exy \rightarrow Xy)) \vee \\ & (x = u \wedge \forall y\,(Euy \rightarrow Xy) \wedge \exists y\,(Evy \wedge \neg Xy)) \vee \\ & (x = v \wedge \forall y\,(Evy \rightarrow Xy) \wedge \exists y\,(Euy \wedge \neg Xy)))](x).\end{aligned}$$

and define $\vartheta_0(u,v) := \forall x\,(x = u \vee x = v \rightarrow \neg\psi(x))$. Clearly, in each stage α, X^α contains all nodes of height less than α other than u and v and one of these just in case they are of different height.
Induction Step. The definition of the relation given in Definition 13.19 actually shows that ϑ_{k+1} is obtained by a *first-order* formula from the relations $\simeq_k$.

By Lemma 13.20 there is a k such that the number of $\simeq_k$-equivalence classes is $\Omega(f)$. Consider the class $\mathcal{C}_k$ of trees defined by $\mathcal{T} \in \mathcal{C}_k$ if, and only if, for any pair u, v of children of the root, $u \simeq_k v$. It is easy to see that this class is bisimulation-closed, M-IFP-definable, and its labelling index is bounded from below by the number of distinct $\simeq_k$ equivalence classes. $\square$

It follows from this that there are bisimulation invariant properties definable in M-IFP that are not definable in MIC. This contrasts with L_μ whose expressive power coincides precisely with the bisimulation invariant fragment of M-LFP. The result also dashes hopes of characterising MIC as the bisimulation-invariant fragment of a natural predicate logic.

13.22 Corollary. MIC *is strictly contained in the bisimulation invariant fragment of* M-IFP*, even if simultaneous inductions are not allowed in* M-IFP.

Chapter 14

Labelling Indices on Arbitrary Transition Systems

In the previous chapter, we considered the labelling index of classes of *acyclic* structures. It was shown that on such structures, the framework of labelling indices provided a uniform method to show inexpressibility results for various fixed-point extensions of modal logic as well as for the bisimulation invariant fragments of first-order based logics. In this chapter, we aim at extending this framework to arbitrary classes of transition systems, i.e. systems that may contain cycles.

As we will see, this extension to cyclic structures requires some changes. Whereas the definition of labelling systems was already given in full generality and extends without changes, there is no longer a straightforward definition of rank on cyclic systems. As transition systems with cycles are not well-founded, the height of the nodes in such a system is no longer defined. Instead we have to come up with a more involved measure to extend the notion of rank from trees to arbitrary systems.

14.1 The Rank of Arbitrary Structures

In order to have a meaningful measure of the growth rate of labelling systems, we need a measure of the size of finite transitions systems that generalises the length of a string and the height of a tree. Consider again the results on labelling indices of classes of trees definable in modal logics. The property of the height of a tree that was of interest for us there was that it bounds the length of any fixed-point induction that can be defined in L_μ or MIC. If there is any hope of extending the results proved for trees, then the new definition of rank on cyclic structures should also be such that it bounds the closure ordinals of formulae in these logics.

As mentioned above, there is no *canonical* candidate for a definition of rank on arbitrary structures. We present two different definitions and show that both methods are equivalent. We can therefore refer to the rank of a structure without being specific about which of the two definitions we actually use.

The first definition is based on the length of the longest path through the transition system.

14.1 Definition (path rank). *The* path rank *of a structure $\mathcal{K}, v$ is defined as the length of the longest non-repeating sequence $u_1, \ldots, u_n$ of nodes in $\mathcal{K}$ such that for all $1 \leq i < n$ the node u_{i+1} is reachable from u_i.*

It is easy to see, that on trees, the path rank is just one plus the height of the tree. We now introduce another combinatorial characterisation of the rank, which makes clear the way in which it generalises the height of a tree or, equivalently, the length of the longest path in an acyclic structure. We first make the following observation, whose proof is trivial.

14.2 Lemma. *If $\mathcal{K}$ is strongly connected, its path rank is equal to the number of nodes in $\mathcal{K}$.*

We use this to define the following characterisation of the rank of a structure.

14.3 Definition. *The* block decomposition *of a structure $\mathcal{K}$ is the acyclic graph $G = (V, E)$ whose nodes are the strongly connected components of $\mathcal{K}$ and where $(s, t) \in E$ if, and only if, for some $u \in s$ and some $v \in t$, there is an action a such that $u \xrightarrow{a} v$.*

For each node s of G, we write $weight(s)$ for the number of nodes u of $\mathcal{K}$ such that $u \in s$.

The rank of a node s of G is defined inductively by

$$rank(s) = weight(s) + \max\{rank(t) \mid (s, t) \in E\}.$$

The block rank *of a rooted finite transition system $\mathcal{K}, v$ is defined as the rank of the block containing v in the block decomposition of $\mathcal{K}$.*

The previous definition offered another measure for the rank of a structure, which was purely combinatorial. The advantage of it is that for any given structure $\mathcal{K}, v$, its block rank can easily be determined.

The following proposition, whose proof is trivial, shows that the two definitions of rank are equivalent.

14.4 Proposition. *The block rank of $\mathcal{K}, v$ is equal to its path rank.*

Henceforth, we refer to the rank of a structure $\mathcal{K}, v$ without being specific about which notion of rank we use. We now aim at establishing that the rank of a structure indeed bounds the closure ordinals of MIC-formulae. For this, it is convenient to first define another notion of rank that is directly based on closure ordinals.

14.5 Definition. *The* induction rank *of a rooted finite transition system $\mathcal{K}, v$ is the largest n such that there is a* MIC*-formula* $(\textbf{ifp}\ X : \psi)$ *for which $v \in X^n$ but $v \notin X^m$ for all $m < n$.*

We can observe the following analogon to Lemma 13.3.

14.6 Lemma. *Let k*MIC *denote the class of formulae in* MIC *where every system of formulae defining a fixed-point induction has at most k rules. Let $\mathcal{K}$ be a structure with distinguished node v and let p be its path rank. Then*

$$\mathrm{cl}_{k\mathrm{MIC}}(\mathcal{K}) \leq k \cdot p,$$

where $\mathrm{cl}_{k\mathrm{MIC}}(\mathcal{K})$ denotes the supremum of all α such that there is a system S of at most k formulae for which α is the least ordinal with $S^\alpha = S^{\alpha+1}$.

Proof. Suppose S is a system of at most k formulae defining variables $X_1, \ldots, X_k$. On every structure $\mathcal{K}, v$ such that there is an induction on S of length n, we construct a non-repeating sequence $u_1 \ldots u_m$ of states[1]in $\mathcal{K}$, where $m := \lceil \frac{n}{k} \rceil$, such that every u_{i+1} is reachable from u_i.

For this, consider the induction stages X_j^i induced by S, where $1 \leq j \leq k$ and $0 \leq i \leq n$. Clearly, for every stage $i > 0$ there must be a state u and a variable X_j such that $u \in X_j^i$ but $u \notin X_j^{i-1}$. We inductively define for every such state u and set X_j a dependency tree $\mathcal{T}(u, X_j)$ as follows. If $i = 1$, the tree consists of a single node labelled by u and X_j. If $i > 1$, then there must be an element v, reachable from u, and a set X_l such that $v \in X_l^{i-1} - X_l^{i-2}$. Otherwise, u would already be contained in X_j at some earlier stage. Now, the dependency tree for u and X_j consists of a node labelled by u and X_j and for each such v a successor which is the root of the dependency tree $\mathcal{T}(v, X_l)$ for v and X_l.

A simple induction shows that the dependency tree for every state u and set X_j such that $u \in X_l^i - X_l^{i-1}$ is of height exactly $i-1$. Now consider the dependency tree $\mathcal{T} := \mathcal{T}(u, X_j)$ for a state u and set X_j such that $u \in X_j^n - X_j^{n-1}$, where n is the closure ordinal of the induction on S. Clearly, no path in $\mathcal{T}$ can contain more than one node labelled by the same pair (v, X_l), for some v and X_l. Therefore, for every state v in $\mathcal{K}$ and every path $\mathcal{P}$ in $\mathcal{T}$, there are at most k nodes labelled by (v, X_l) for some l. Let $\mathcal{P} \subseteq \mathcal{T}$ be a path in $\mathcal{T}$ of maximal length, i.e. length n. As $\mathcal{P}$ cannot contain more than k nodes labelled by the same state v in $\mathcal{K}$, it follows that there are at least $m := \lceil \frac{n}{k} \rceil$ nodes which are labelled by different states from $\mathcal{K}$. Let $(u_1, \ldots, u_m)$ be the sequence of these states in decreasing order with respect to their height in $\mathcal{T}$. By construction of the dependency tree, every u_{i+1} is reachable from u_i. Thus, we have constructed a non-repeating sequence of length m where every node is reachable from its predecessor.

[1]In this proof we refer to the nodes in $\mathcal{K}$ as *states* to distinguish them from the nodes of a tree that will be defined later on.

By definition of the path rank, m must be less than or equal to p. As $n \leq k \cdot \lceil \frac{n}{k} \rceil = k \cdot m \leq k \cdot p$, the lemma is proved. □

While the path rank of a structure $\mathcal{K}, v$ provides a combinatorial measure that bounds its induction rank, it is not an exact characterisation. Nor can we expect it to be exact because it is clear that the induction rank is invariant under bisimulation while the path rank is not. It may therefore be more appropriate to consider the path rank, not of a structure $\mathcal{K}$, but of its quotient under bisimulation $\mathcal{K}_{/\sim}$. With this, we do indeed get the required converse to Lemma 14.6.

14.7 Lemma. *If the path rank of $\mathcal{K}_{/\sim}$ is n, there is a formula $\varphi(X)$, positive in X whose closure ordinal on $\mathcal{K}$ is n.*

Proof. Suppose there is a non-repeating sequence $(u_1, \dots, u_n)$ witnessing that the path rank of $\mathcal{K}_{/\sim}$ is n. Since, by definition of $\mathcal{K}_{/\sim}$, the nodes u_i are pairwise not bisimilar, we can write modal formulae $\varphi_1, \dots, \varphi_n$ such that u_i is the unique element of $\mathcal{K}$ satisfying φ_i.

Now consider the formula

$$\psi := (\mathbf{ifp}\ X : \varphi_1 \vee \bigvee_{i=2}^{n} (\varphi_i \wedge \mathit{somewhere}(X \wedge \varphi_{i-1}))),$$

where $\mathit{somewhere}(\varphi)$ is the formula $(\mathbf{ifp}\ Y : \varphi \vee \Diamond Y)$ as defined in Proposition 11.5. Obviously, the closure ordinal of ψ is n. □

An immediate consequence of the proofs of Lemma 14.6 and 14.7 is that it does not matter whether we define the induction rank of a structure in terms of MIC or L_μ.

14.2 The Labelling Index of Modal Logics

In this section, we aim at extending the bounds on the labelling indices for modal logics from trees to the general setting. We start our investigation with the modal iteration calculus. On trees, we proved an exponential upper bound for the labelling index of classes of trees definable in MIC. Considering again the proof for this result, it becomes clear that at no point it was used that the underlying transition system was a tree. Thus, the proof extends literally to the case of arbitrary transition systems.

14.8 Theorem. *Every class of transition systems that is definable in* MIC *has at most exponential labelling index.*

Clearly, as ML-formulae are just MIC-formulae without fixed points, we get the following corollary.

14.9 Corollary. *Every class of transition systems definable in* ML *has constant automaticity.*

In Section 13.3.3 and 13.4 we proved a non-elementary lower bound for classes of trees definable in M-IFP or L^{ω}_{μ}. Clearly, this also gives a lower bound on arbitrary structures.

14.10 Theorem. (i) *There are classes of transition systems definable in L^{ω}_{μ} which have a non-elementary labelling index.*

(ii) *There is no elementary upper bound for the labelling index of* M-IFP*-definable classes of transition systems.*

Note the difference between the two statements of the Theorem. Whereas for L^{ω}_{μ} there is a *single* formula whose labelling index is not bound by any elementary function, the only statement about M-IFP we can show is that there is no elementary function dominating the labelling index of all M-IFP-formulae. It is an open problem whether the labelling index of any *individual* M-IFP-formula can be bound by an elementary function.

14.3 The Modal μ-Calculus

We now turn towards the labelling index of L_{μ}-definable classes of transition systems. We first show that on classes of arbitrary structures, the constant labelling index for L_{μ} does no longer hold true. For instance, it can easily be seen, using pumping arguments, that to express reachability, constant size labelling systems are not sufficient.

14.11 Lemma. *There is a L_{μ}-definable class $\mathcal{C}$ of structures that has a linear lower bound on its labelling index.*

Proof. Let $\mathcal{C}$ be the class of transition systems such that there is a node labelled by the proposition p and is reachable from the root. We claim that $\mathcal{C}$ has linear labelling index. Clearly, $\mathcal{C}$ can be accepted by a family of labelling systems of linear size. Simply take a label to recognise nodes labelled by p and then, for each $i \in \{1, \dots, n\}$, a label q_i to remember that a node labelled p has been seen within i steps from the current position. Further, take an error label for nodes from which no node labelled p can be reached.

On the other hand, each family of labelling systems accepting $\mathcal{C}$ must have at least linear size. Assume otherwise and suppose that for some $n > 2$ there is a labelling system $\mathcal{L}$ of size less than $n-1$ accepting the class $\mathcal{C}_n$ of structures from $\mathcal{C}$ of rank at most n. Consider the structure $\mathcal{K} := (\{0, \dots, n-1\}, E, P)$ with $E := \{(i, i+1) : 0 \leq i < n-1\}$ and $p := \{n-1\}$. Obviously, $\mathcal{K}, 0 \in \mathcal{C}_n$ and thus $\mathcal{K}, 0$ is accepted by $\mathcal{L}$. As there are less than $n-1$ labels, there must be two different nodes $u < v$ in $\mathcal{K}$ labelled by the same label q in $\mathcal{L}$. But then the same labelling also witnesses that the system

$\mathcal{K}' := (\{0, \dots, v\}, E', P')$, where $E' := \{(i, i+1) : 0 \le i < v\} \cup \{(v, u+1)\}$ and $P' := \varnothing$, would be accepted by $\mathcal{L}$. As $\mathcal{K}', 0 \notin \mathcal{C}$ we get a contradiction. □

This also shows that various ML extensions like LTL, CTL, or CTL* have non-constant labelling index, as they can express reachability.

We now turn towards establishing upper bounds for the labelling index of L_μ-definable classes of transition systems.

The main difference between L_μ and the fixed-point logics considered above is monotonicity. This has a major impact on the definition of labelling systems accepting L_μ definable classes of structures.

Consider the labelling systems for MIC-formulae φ as defined above. In each node u of the structures, we remembered for the sub-formulae of φ every tuple $(i_1, \dots, i_k)$ of induction stages where the sub-formula becomes true at u. As L_μ-formulae are monotone, if a sub-formula is true at a tuple of stages $(i_1, \dots, i_k)$ it will also be true at all higher stages of μ and all lower stages of ν-operators. Thus, if we only had one fixed-point operator, say μX, it would suffice to mark each node u of the structure by the number of the stage at which it is included into the fixed point of X and to give it a special label if it is not included at all. We would thus only have linearly many labels in the labelling system.

But monotonicity also helps if there are more than one fixed-point operator. The reason is that if a formula is true at a node u and a tuple of stages $\bar{\imath}$, then it is also true at u if all or some of its free fixed-point variables are interpreted by their respective fixed points. With this, it turns out to be sufficient to consider in each node u of the transition system only those tuples $\bar{\imath}$ of stages where at most one fixed-point induction has not yet reached its fixed point. As there are only polynomially many such tuples we get a polynomial upper bound on the size of the labelling systems. We now give a detailed proof of this fact, implementing these ideas.

14.12 Definition. *Let $\varphi \in L_\mu$ be a formula in guarded normal form (see Definition 10.1) with fixed-point variables $X_1, \dots, X_k$ that are bound by sub-formulae $\mu X_i.\varphi_i$ or $\nu X_i.\varphi_i$. Let Φ be the set of sub-formulae of φ and let $P \subseteq \mathcal{P}$ be a set of propositions.*

A function $q : \Phi \to \{0, \dots, n+1, \bot\}^k$, where $\bot$ is an abbreviation for $n+2$, is locally P-consistent*, if the following conditions hold for all $\psi \in \Phi$. We write $q(\psi)_i$ to denote the i-th component of the image of a formula ψ.*

- *If $\psi := p$, then for all i, $q(\psi)_i := 0$ if $p \in P$ and $q(\psi) = \bot$ otherwise.*
- *Conversely, if $\psi := \neg p$, then for all i, $q(\psi)_i := \bot$ if $p \in P$ and $q(\psi) = 0$ otherwise.*
- *For all i, $q(\psi_1 \vee \psi_2)_i := \min\{q(\psi_1)_i, q(\psi_2)_i\}$.*

- *For all i, $q(\psi_1 \wedge \psi_2)_i := \max\{q(\psi_1)_i, q(\psi_2)_i\}$.*
- *If $\psi := \lambda_i X_i.\varphi_i$, with $\lambda_i \in \{\mu, \nu\}$, then for all j,*

$$q(\psi)_j := \begin{cases} n+1 & j = i, \lambda_i = \mu \textit{ and } q(\varphi_i)_i \neq \bot \\ 0 & j = i, \lambda_i = \nu \textit{ and } q(\varphi_i)_i = 0 \\ q(\varphi_i) & j < i \\ \bot & \textit{otherwise.} \end{cases}$$

- *If $\psi := X_i$ and X_i is a variable bound by a μ-operator, then for all j,*

$$q(X_i)_j := \begin{cases} q(\varphi_i)_j + 1 & \textit{if } j = i \textit{ and } q(\varphi_i)_i \neq \bot \\ 0 & \textit{if } j > i \textit{ and } q(\varphi_i)_i \neq \bot \\ \bot & \textit{otherwise.} \end{cases}$$

- *If $\psi := X_i$ and X_i is a variable bound by a ν-operator, then for all j,*

$$q(X_i)_j := \begin{cases} 0 & \textit{if } j \geq i \textit{ and } q(\varphi_i)_i \neq \bot \\ \bot & \textit{otherwise.} \end{cases}$$

A function q is locally consistent *if it is locally P-consistent for some $P \subseteq \mathcal{P}$. Let Q be the set of all locally consistent functions $q : \Phi \to \{0, \ldots, n+1, \bot\}^k$ and let $M \subseteq Q \times \mathcal{A}$. A function $q \in Q$ is* globally (M, P)-consistent, *if it is locally P-consistent and further the following conditions holds:*

- *If $\psi := \langle a \rangle \psi'$, then for all i, $q(\psi)_i := \min(\{\bot\} \cup \{q'(\psi')_i : (q', a) \in M\})$.*
- *If $\psi := [a]\psi'$, then for all i, $q(\psi)_i := \max(\{0\} \cup \{q'(\psi')_i : (q', a) \in M\})$.*

The function is globally consistent, if it is globally (M, P)-consistent for some pair (M, P).

Globally consistent functions will play the same role for L_μ as φ-types played for MIC. We first establish some facts about (M, P)-consistent functions.

14.13 Proposition. (*i*) *For every pair (M, P) as in the definition above, there is exactly one function that is globally (M, P)-consistent.*

(*ii*) *For $1 \leq i \leq k$, let φ_i be the sub-formulae of φ defining fixed points, i.e. the formulae $\lambda_i X_i.\varphi_i$ occur in φ where $\lambda_i \in \{\nu, \mu\}$. For all i such that $\lambda_i = \nu$, all formulae $\psi \in \Phi$, and all globally consistent functions f, either $(f(\psi))_i = 0$ or $(f(\psi))_i = \bot$.*

Proof. For Part (i), recall that the formula φ is in guarded normal form. This implies that the definition is not cyclic, i.e. the labels of the formulae φ_i, $\mu X_i.\varphi_i$, and $\nu X_i.\varphi_i$ do not depend on the labels of the defined variable X_i. As all rules are deterministic, the label of each formula must be well defined.

Part (ii) can easily be proved by induction on the structure. $\square$

We are now ready to define the labelling systems for L_μ-formulae.

14.14 Definition. *Let $\varphi \in L_\mu$ be a formula with fixed-point variables $X_1, \dots, X_k$ that are bound by sub-formulae $\mu X_i.\varphi_i$ or $\nu X_i.\varphi_i$. Let Φ be the set of sub-formulae of φ.*

For every $n < \omega$ define the labelling system $\mathcal{L}_\varphi(n)$ as follows.

- *Q is defined as the set of all globally consistent functions $q : \Phi \to \{0, \dots, n+1, \bot\}^k$.*
- *$\mathcal{A}$ is the set of actions and $\mathcal{P}$ the set of proposition symbols in the signature.*
- *For each $M \subseteq Q \times \mathcal{A}$ and $P \subseteq \mathcal{P}$ define $\delta(M, P)$ as the function q that is globally (M, P)-consistent.*
- *Finally, define $\mathcal{F} := \{q \in Q : q(\varphi)_1 \neq \bot\}$.*

Proposition 14.13 above implies that the definition of $\mathcal{L}_\varphi$ is well-defined. We now prove the correctness of the definition. The proof is split into two separate lemmata.

14.15 Lemma. *Let $\mathcal{K}, v$ be a transition system of rank at most n and let φ and $\mathcal{L}_\varphi(n)$ be as in Definition 14.14 above. If $\mathcal{L}_\varphi(n)$ accepts $\mathcal{K}, v$, then $\mathcal{K}, v \models \varphi$.*

Proof. Let f be a $\sim$-consistent labelling witnessing that $\mathcal{L}_\varphi$ accepts $\mathcal{K}, u$. Let $X_1, \dots, X_k$ be the fixed-point variables occurring in φ. We first fix some notation. We write $\tilde{X}_i^m$ for the stage m of the induction on φ_i in which all free variables of φ_i other than X_i are interpreted by their respective fixed points. Consequently, $\tilde{X}_i^\infty$ denotes the fixed point of X_i in the induction on φ_i where all free variables are interpreted by their fixed points.

We claim that for all nodes $u \in V$, all $i \leq k$, and all formulae $\psi \in \Phi$, if $(f(u)(\psi))_i = m \neq \bot$, then

- $\mathcal{K}, u \models \psi(\tilde{X}_i^m)$, if X_i is bound by a μ-operator, and
- $\mathcal{K}, u \models \psi(\tilde{X}_i^\infty)$, if X_i is bound by a ν-operator.

Clearly, this implies the lemma as by definition of acceptance, the root v must be labelled by a label in $\mathcal{F}$ and these labels all map φ to something different from $\bot$.

The claim is proved simultaneously for all nodes by induction on the labels $f(u)(\psi)$ and for each such label by induction on the structure of the formulae. Let $q := f(u)$ be the label assigned to the node u. Further, let P be the set of propositions true at u and let M be the set of pairs (q', a) such that $a \in \mathcal{A}$ and there is an a-successor u' of u with $f(u') = q'$. Now assume $\psi \in \Phi$ is a formula and $q(\psi) = m \neq \bot$.

- Suppose $\psi := p$ and let $i \leq k$. By definition, if q is the label assigned to u by f, and $q(p)_i \neq \bot$, then this implies that $p \in P$ and thus $\mathcal{K}, u \models \psi$.
- The case $\psi := \neg p$ is analogous.
- The case for Boolean connectives and next-modalities are obvious.
- Let $\psi := \lambda X_i.\varphi_i$ for $\lambda \in \{\mu, \nu\}$. Suppose first that $\lambda = \mu$ and consider some $j < i$. By definition of δ, if $q(\psi)_j = m \neq \bot$, then $q(\psi)_j = q(\varphi_i)_j$. By induction on the structure of the formulae, this implies that $(\mathcal{K}, \tilde{X}_j^m), u \models \varphi_i$. Recall the convention that $(\mathcal{K}, \tilde{X}_j^m), u \models \varphi_i$ means that φ_i holds at the node u if X_j is interpreted by its m-th stage and all other free variables are interpreted by their fixed points. In particular, this implies that u occurs in the fixed point of X_i and thus $(\mathcal{K}, \tilde{X}_j^m), u \models \psi$.

  Now suppose $j = i$. By definition, $q(\psi)_i \neq \bot$ implies $q(\psi)_i = n + 1$ and $q(\varphi_i)_i = m \neq \bot$. By induction on the stages, this implies that $(\mathcal{K}, \tilde{X}_i^m), u \models \varphi_i$ and therefore also $(\mathcal{K}, \tilde{X}_i^m), u \models \psi$.

  The case where $\lambda = \nu$ is analogous.
- Finally, suppose $\psi := X_i$. Assume first that X_i is bound by a μ-operator. If $q(\psi) \neq \bot$, then, by definition, $q(\varphi_i)_i = m \neq \bot$ and $q(\psi)_i = m+1$. Thus, by induction on the stages, $(\mathcal{K}, \tilde{X}_i^m), u \models \varphi_i$ and therefore $(\mathcal{K}, \tilde{X}_i^{m+1}), u \models X_i$.

  Now consider some $j > i$. By definition, $q(\psi)_j := 0$. We have to show that $(\mathcal{K}, \tilde{X}_j^0), u \models X_i$. Recall that this means that X_i is true at the node u under the interpretation of X_j by its 0-th stage and where all other fixed point variables, including X_i, are interpreted by their respective fixed points. As proved in the preceding paragraph, u occurs in the fixed point of X_i. From this, the claim follows immediately.

  Now suppose X_i is bound by a ν-operator. By definition, $q(X_i)_j \neq \bot$ implies $q(X_i)_j = 0$. As X_i is bound by a greatest fixed-point operator, X_i^0 contains the whole universe and therefore X_i holds true at u.

This proves the claim and thus the lemma. □

We now turn to the converse direction.

14.16 Lemma. *Let $\mathcal{K}, v$ be a transition system of rank no more than n. Let φ be a formula and $\mathcal{L}_\varphi(n)$ be the corresponding labelling system. Then $\mathcal{K}, v \models \varphi$ implies that $\mathcal{L}_\varphi$ accepts $\mathcal{K}, v$.*

Proof. Let $X_1, \dots, X_k$ be the fixed-point variables occurring in φ and let $X_1^\infty, \dots, X_k^\infty$ be their fixed-points on $\mathcal{K}$, i.e. X_i^∞ is the fixed point of the induction on φ_i where all variables other than X_i occurring free in φ_i are interpreted by their fixed points.

For each node u define a function $q_u : \Phi \to \{0, \dots, n+1, \bot\}$ as follows. Let $\psi \in \Phi$ be a sub-formula. For all i such that ψ is not a sub-formula of φ_i define $q(\psi)_i := \bot$. Otherwise consider the stages of the induction on φ_i where again all free variables of φ_i other than X_i are interpreted by their fixed points. If, for some m, ψ becomes true at u at stage X_i^m, take the smallest such m and define $q(\psi)_i := m$ if φ_i is bound by a μ-operator and define $q(\psi)_i = 0$ if it is bound by a ν-operator. Otherwise define $q(\psi)_i := \bot$.

It is now a simple observation that the function $f : V \to Q$ assigning to each node $u \in V$ the function q_u is a $\sim$-consistent labelling witnessing that $\mathcal{L}_\varphi$ accepts $\mathcal{K}, v$. □

The following theorem is an immediate consequence of the preceding two lemmata.

14.17 Theorem. *Every class of transition systems definable in L_μ has at most polynomial labelling index.*

Together with Lemma 14.11, the theorem implies that the μ-calculus has a polynomial labelling index in the sense that every L_μ definable property has polynomial labelling index and there are L_μ-definable properties with a linear lower bound on the labelling index. Note, however, that there still is a gap between the polynomial upper and the linear lower bound. Whether the labelling index of any L_μ-formula can be bound by a fixed polynomial, is linear for instance, or whether for any polynomial there is a formula in L_μ whose labelling index exceeds the polynomial, is still an open problem.

Another consequence of the proof is that if a L_μ-formula does not use any μ-operators, the class of structures defined by it has constant labelling index.

14.4 Labelling Index and Complexity

In this section we study the relationship between the labelling index of a class of structures and its complexity in terms of conventional complexity measures such as time and space. We begin by contrasting the labelling index with the usual notion of computational complexity in terms of machine time measured as a function of the size of the structure. We demonstrate that the two measures are not really comparable by exhibiting a class of structures that is decidable in polynomial time but has non-elementary labelling index and an NP-complete problem that has exponential labelling index.

The first of these is the class of finite trees $\mathcal{T}$ such that if t_u and t_v are subtrees rooted at a successor of the root, then $t_u \sim t_v$. As shown in Section 13.3.3, there is no elementary bound on the automaticity of this class, but as it is definable in L^{ω}_{μ}, it is decidable in time polynomial in the *size* of the tree. This yields the following result.

14.18 Proposition. *There is a polynomial time decidable class of transition systems with non-elementary labelling index.*

In contrast, we can construct an NP-complete problem of much lower labelling index. We obtain this by encoding propositional satisfiability as a class of structures $\mathcal{S}$ closed under bisimulation, and demonstrate that it is accepted by an exponential family of labelling systems.

14.19 Theorem. *There are* NP-*complete problems with exponential labelling index.*

Proof. Let φ be a propositional formula formed from the propositional variables $V_1, \dots, V_n$ using the Boolean operations $\wedge$, $\vee$ and $\neg$. We define an encoding of φ as a transition system $\mathcal{T}_\varphi$ in the vocabulary with a single action a and propositional vocabulary $\{\wedge, \vee, \neg, V, \text{Count}\}$. $\mathcal{T}_\varphi$ is inductively defined as follows.

- If φ is a variable V_i, then $\mathcal{T}_\varphi$ consists of a root labelled V which is connected to a chain of length i of nodes labelled Count. Further, there is an edge from the root to itself.
- If φ is $\psi_1 \wedge \psi_2$, $\mathcal{T}_\varphi$ consists of a root labelled $\wedge$ with two successors which are roots of $\mathcal{T}_{\psi_1}$ and $\mathcal{T}_{\psi_2}$. The rule for $\psi_1 \vee \psi_2$ is similar.
- If φ is $\neg\psi$, $\mathcal{T}_\varphi$ consists of a root labelled $\neg$ with a single successor which is the root of $\mathcal{T}_\psi$.

Now, the class of structures $\mathcal{S}$ is defined as

$$\mathcal{S} := \{\mathcal{K} : \mathcal{K} \sim \mathcal{T}_\varphi \text{ for a satisfiable formula } \varphi\}.$$

It is immediate from the definition that $\mathcal{S}$ is bisimulation closed and NP-completeness follows from the obvious reduction. We now demonstrate an exponential family of labelling systems that accepts $\mathcal{S}$.

Let TA_n denote the set of possible truth assignments to the variables $\{V_1, \dots, V_n\}$. Obviously, the size of TA_n is 2^n. We define a labelling system $\mathcal{L}_n$ as follows. The set Q of states is defined as $\{0,1\} \times TA_n \ \cup \ \{C_i : 1 \leq i \leq n\} \ \cup \ \{\text{Fail}\}$.

Let $M \subseteq Q \times \mathcal{A}$ and $P \subseteq \mathcal{P}$ be given. The transition function $\delta(M, P)$ is defined as follows.

- If $P := \{\text{Count}\}$, then $\delta(M, P) = C_{i+1}$, provided that for all $(q, a) \in M$, $q = C_i$.

- Let $P := \{V\}$. Suppose there is some i such that there is a pair $(C_i, a) \in M$ and for no $j \neq i$ a pair (C_j, a) occurs in M. Let $T \in TA_n$ be a truth assignment and let t be the truth value assigned to V_i by T. If there is some pair $(q, a) \in M$ such that $q := (t, T)$ and for no pair $(q', a) \in M$, $q' := (t', T')$ with $t \neq t'$ or $T \neq T'$, then $\delta(M, P) := (t, T)$.

- If $P := \{\wedge\}$, then $\delta(M, P) := (1, T)$ if for all $(q, a) \in M$, $q = (1, T)$. If there is a pair $(q, a) \in M$ with $q := (0, T)$ and for all $(q, a) \in M$, either $q := (0, T)$ or $q := (1, T)$, then $\delta(M, P) := (0, T)$.

- The rule for $P := \{\vee\}$ is analogous.

- Finally, if $P := \{\neg\}$, then $\delta(M, P) := (1, T)$ provided that for all $(q, a) \in M$, $q := (0, T)$, and $\delta(M, P) := (0, T)$ provided that for all $(q, a) \in M$, $q := (1, T)$.

- In all other cases, $\delta(M, P) = \{\text{Fail}\}$.

Finally, the set $\mathcal{F}$ of accepting states is the set $\{1\} \times TA_n$. It is now immediate from the construction that $\mathcal{L}_n$ accepts a transition system $\mathcal{K}$ of rank at most n if, and only if, $\mathcal{K} \in \mathcal{S}$. □

In Definitions 13.6 and 13.7, we imposed restrictions on labelling systems to guarantee that the class of structures defined is closed under bisimulation. An alternative is to allow arbitrary labelling systems, but change the notion of acceptance of a structure to guarantee that the class of structures is closed. For instance, we could say that $\mathcal{K}, v \models \mathcal{L}$ if, and only if, there is a labelling f of the universe V which labels any pair of bisimilar points by the same state. It turns out that this makes labelling systems potentially more powerful. Indeed, we can then construct a single labelling system accepting the NP-complete class of structures defined in Theorem 14.19. On the other hand, it is open whether the exponential bound in Theorem 14.19 can be lowered.

14.5 The Trace-Equivalence Problem

In this section, we apply our methods to a particular problem that is of interest to verification – the *trace-equivalence* problem. We determine exactly the labelling index of a number of variations of the problem and thereby derive results about their expressibility in various modal fixed-point logics.

Recall the definition of the trace equivalence problem as presented in Definition 12.9.

14.20 Definition (Trace equivalence problem). *Let Σ be an alphabet, i.e. a non-empty set of proposition symbols. Further, let $\mathcal{F}$ be a proposition symbol not in Σ.*

Let $\mathcal{K}, v$ be a finite rooted transition system. The set $\mathcal{T}(\mathcal{K})$ of traces in $\mathcal{K}$ *is defined as the set of words $w \in \Sigma^*$ such that there is a path labelled by w in $\mathcal{K}$ from v to a node in $\mathcal{F}^{\mathcal{K}}$. Two transition systems $\mathcal{K}, v$ and $\mathcal{K}', v'$ are* trace equivalent*, if $\mathcal{T}(\mathcal{K}) = \mathcal{T}(\mathcal{K}')$.*

The trace equivalence problem *($\mathcal{TE}$) is defined as the decision problem of the class*

$$\mathcal{TE}_\Sigma := \left\{ \mathcal{K}, v : \begin{array}{l} \mathcal{K}, v \textit{ is finite and for all } u, w \textit{ such that } v \to u \textit{ and} \\ v \to w, \mathcal{K}, u \textit{ and } \mathcal{K}, w \textit{ are trace equivalent} \end{array} \right\}.$$

The unary trace equivalence problem *is defined as the trace equivalence problem over unary alphabets, i.e. alphabets Σ with $|\Sigma| = 1$.*

The acyclic trace equivalence problem *is defined as the subclass of $\mathcal{TE}$ where all structures are acyclic.*

We now aim at establishing the labelling index of several variants of the trace equivalence problem. We start with the simplest case, the unary trace equivalence problem over acyclic structures.

14.21 Theorem. *On acyclic structures, unary trace equivalence has exponential labelling index.*

Proof. Let $\mathcal{T}$ be an instance of the unary acyclic trace equivalence problem and let $\mathcal{T}_u$ and $\mathcal{T}_v$ be two subtrees rooted at successors of $\mathcal{T}$. Let n be the height of $\mathcal{T}$. It is easily seen that if $\mathcal{T}_u$ and $\mathcal{T}_v$ are not trace equivalent, then there must be some $l \leq n$ such that in one of $\mathcal{T}_u, \mathcal{T}_v$ there is a path of length l from the root to a node labelled by $\mathcal{F}$ but not in the other.

Now, for each tree $\mathcal{T}_u$ rooted at a successor of the root of $\mathcal{T}$ define $TR(\mathcal{T}_u) := \{n :$ there is a path of length $l \leq n$ from u to a node labelled by $\mathcal{F}\}$. Thus, $\mathcal{T} \in \mathcal{TE}_\Sigma$ if, and only if, for every pair u and v of successors of the root, $TR(\mathcal{T}_u) = TR(\mathcal{T}_v)$.

To establish an upper bound for the labelling index, consider the family of labelling systems $\mathcal{L}_n$ defined as follows. Let $\mathcal{A} := \{E\}$ be the set of actions and $\mathcal{P} := \{\mathcal{F}\}$ be the set of propositions. For every $n < \omega$ define $\mathcal{L}_n := (Q_n, \mathcal{A}, \mathcal{P}, \delta_n, F_n)$, where

- $Q_n := \text{Pow}(\{a\}^{\leq n}) \times \{e, n\}$,
- $F_n := \{(P, x) \in Q : x = e\}$, and
- for every $M := \{(P_1, x_1), \ldots, (P_k, x_k)\} \subseteq Q \times \mathcal{A}$ and $P \subseteq \mathcal{P}$ define

  $$\delta(M, P)_n := (\{a^l : a^{l-1} \in P_i \text{ for some } i\} \cup \{a : \text{ if } \mathcal{F} \in P\}, x),$$

  where $x = e$ if $P_i = P_j$ for all i, j and $x = n$ otherwise.

It is now easily seen that the root of a tree $\mathcal{T}$ of height at most n gets a label (P, e) in the labelling system $\mathcal{L}_n$ if, and only if, all its successors are trace equivalent. As $|Q_n| = 2 \cdot 2^n$, it follows that there is an exponential upper bound for the labelling index of the unary acyclic trace equivalence problem.

Towards establishing a lower bound, suppose that for some n there is a labelling system $\mathcal{L}$ with less than 2^{n-1} states accepting all trees $\mathcal{T} \in \mathcal{TE}$ of height at most n. As there are 2^{n-1} trees of height at most $n-1$ which are pairwise not trace equivalent, it follows that there are two trees $\mathcal{T}_1$ and $\mathcal{T}_2$ which are not trace equivalent but whose root is labelled by the same state in $\mathcal{L}$. Thus, the tree $\mathcal{T}$ consisting of a root with two copies of $\mathcal{T}_1$ as successors is accepted by $\mathcal{L}$ if, and only if, the tree $\mathcal{T}'$ consisting of a root with one copy of $\mathcal{T}_1$ and one copy of $\mathcal{T}_2$ as successors is accepted by $\mathcal{L}$. As $\mathcal{T} \in \mathcal{TE}$ but $\mathcal{T}' \notin \mathcal{TE}$ we get a contradiction. □

We now consider the binary trace equivalence problem over acyclic structures. Again for any pair of trees of height n which are not trace equivalent there is a string of length at most n witnessing this. But now, as the alphabet is binary, there are 2^n different strings of length at most n. Thus, the same proof as above only with the modification that the state set of the labelling system $\mathcal{L}_n$ is now $Q_n := \text{Pow}(\{a, b\}^{\leq n})$ establishes the double exponential upper and lower bound for the binary acyclic trace equivalence problem.

14.22 Theorem. *On acyclic structures, binary trace equivalence has a double exponential labelling index.*

Finally, we turn towards the labelling index of the unary trace equivalence problem over arbitrary structures. In particular, we aim at proving a double exponential upper and lower bound for this problem.

14.23 Theorem. *On arbitrary structures, unary trace equivalence has a double exponential labelling index.*

Proof. Essentially, the theorem is proved along the same line as the Theorems 14.21 and 14.22 above. The only difference is that there no longer is a linear bound on the length of the string witnessing that two structures are not trace equivalent. Instead, we show that if two structures of rank at most n are not trace equivalent, then there must be a string of length at most 2^n separating them. The result is based on a construction from [SM73].

In the proof of Theorem 6.1 in that paper, Stockmeyer and Meyer give a LOGSPACE-reduction from 3CNF[2] to the trace equivalence problem for unary structures, or, equivalently, the language equivalence problem for automata over unary alphabets. The idea is to code truth assignments for propositional variables $X_1, \ldots, X_n$ by numbers and define for every formula φ in 3CNF an automaton that accepts a word of length k if, and only if, k codes a truth assignment that satisfies φ.

Clearly, for any truth assignment T to the variables $X_1, \ldots, X_n$ there is a propositional formula φ_T that is uniquely satisfied by T and whose size is linearly bounded by n. By the construction we get an automaton that accepts a word of length k if, and only if, k encodes T. As the reduction is in LOGSPACE, the size of the automaton is polynomially bounded by the size of φ_T and thus by n. Therefore, the rank of any such automaton viewed as a graph is also polynomially bounded by n.

Now, for any set $\mathcal{S}$ of truth assignments to the variables $X_1, \ldots, X_n$, define the automaton $\mathcal{A}_\mathcal{S}$ consisting of a root and for each assignment T in $\mathcal{S}$ a copy of the automaton $\mathcal{A}_T$ rooted at a successor of the root. By Proposition 14.4, the rank of $\mathcal{A}_\mathcal{S}$ is also polynomially bounded in n. Now, by construction of $\mathcal{A}_\mathcal{S}$, the automaton accepts a string of length $k+1$ if, and only if, k encodes a truth assignment in $\mathcal{S}$.

Clearly, there are 2^{2^n} sets of truth assignments to n variables and therefore 2^{2^n} automata of rank $n+1$ which are pairwise not trace equivalent. Now, the proof of the double exponential lower and upper bound on the labelling of the trace equivalence problem for unary structures follows along the same line as the proofs of the Theorems 14.21 and 14.22. □

Combining the results in the preceding theorems with the bounds on the labelling indices proved above, we immediately get the following corollary.

14.24 Corollary. (*i*) *Unary trace equivalence on acyclic structures is not definable in L_μ.*

(*ii*) *Binary trace equivalence on acyclic structures as well as unary trace equivalence on arbitrary structures are not definable in* MIC.

As unary trace equivalence on acyclic structures is definable in MIC, this provides another example of a property separating MIC and L_μ. Moreover, we have proved in Theorem 12.10 that unary trace equivalence on arbitrary structures is definable in MPC. Thus, the double exponential lower bound for this problem immediately implies a separation of MIC and MPC.

[2]3CNF is the class of propositional formulae in conjunctive normal form where every disjunct consists of at most 3 literals.

14.25 Corollary. MIC *is strictly contained in* MPC.

Finally, we compare the results against the computational complexity of these variants of the trace equivalence problem. The problem on acyclic structures (whether unary or binary) is decidable in polynomial time. The unary trace equivalence problem on arbitrary structures is co-NP-complete, while the general trace equivalence problem is Pspace-complete. Thus, the following corollary is immediate.

14.26 Corollary. MPC *is not contained in* L_μ^ω*, and thus not in* IFP*, provided that* NP *is different from* Ptime*.*

Part III

Constraint Databases

Chapter 15

Constraint Databases

Constraint databases have been introduced by Kanellakis, Kuper, and Revesz [KKR90, KKR95] in 1990 as a possible model for infinite databases. As a motivating example, consider a database storing simple geometrical objects, like rectangles or circles. In a relational database, a rectangle would typically be stored in a relation containing the coordinates of two diagonal corners. Similarly, a circle would be stored by the coordinates of its centre and the radius. Now, a user who wants to know whether two particular objects in the database intersect, first must know the type of the objects, i.e. rectangles or circles, then has to query the coordinates, and finally use some algorithm to determine whether they intersect.

It would be more convenient if the database system allowed the user to work with the objects as if they were rectangles or circles, i.e. infinite sets of points in the plane. The details of how they are actually stored in the database should be hidden. Further, the user should be allowed to use build in predicates like addition or multiplication, so that querying the database for the intersection of some stored object with a specified area becomes possible.

With this in mind, Kanellakis, Kuper, and Revesz introduced the constraint database model as a general database model where such requirements are met.

15.1 The Constraint Database Model

In the constraint database model, databases are expansions of a fixed structure $\mathfrak{A}$, the so-called *context structure*, by a finite set of potentially infinite database relations. The context structure, e.g. $\mathfrak{A} := (\mathbb{R}, <, +, \cdot)$, provides the built in predicates on its universe. Clearly, although the active domain might be infinite, the relations have to be finitely representable so that they can be manipulated by a computer system. In the constraint database framework, first-order logic is used as formalism to represent infinite relations. This is made precise in the following definition.

15.1 Definition. *Let τ be a signature and $\mathfrak{A} := (A, \tau)$ be an arbitrary τ-structure with universe A. A k-ary relation $R \subseteq A^k$ is* finitely representable over $\mathfrak{A}$ *if there is a quantifier-free first-order formula $\varphi_R(\overline{x}) \in \mathrm{FO}[\tau]$, with k free variables, such that for all $\overline{a} \in A^k$*

$$\mathfrak{A} \models \varphi[\overline{a}] \qquad \textit{if, and only if,} \qquad \overline{a} \in R.$$

The formula φ_R is called the finite representation of R over $\mathfrak{A}$*. Finitely representable relations are also called constraint relations. Note that the formula φ_R is allowed to use parameters from A.*

One might be tempted to ask why only quantifier-free formulae are allowed as representations. We will come back to this question at the end of Section 15.2 where we have all the terminology at hand to answer it. We now give a precise definition of the constraint database model.

15.2 Definition. *Let σ and τ be disjoint signatures such that σ is relational and let $\mathfrak{A} := (A, \tau)$ be an arbitrary τ-structure with universe A. A* σ-database over $\mathfrak{A}$ *is a σ-expansion $\mathfrak{B}$ of $\mathfrak{A}$, in terms $\mathfrak{B} := (\mathfrak{A}, \sigma)$.*

The database is called finite*, if the interpretation of all relations from σ in $\mathfrak{B}$ is finite. It is called* finitely representable *or* σ-constraint database *if for all relations $R \in \sigma$ the interpretation $R^{\mathfrak{B}}$ in $\mathfrak{B}$ is finitely representable over $\mathfrak{A}$. In this case, a set $\Phi := \{\varphi_R : R \in \sigma\}$, where the formulae φ_R are finite representations of $R^{\mathfrak{B}}$, is called a representation of $\mathfrak{B}$.*

If a concrete representation Φ of $\mathfrak{B}$ is important we sometimes write $\mathfrak{B} := (\mathfrak{A}, \Phi)$ to define $\mathfrak{B}$. The structure $\mathfrak{A}$ is called the context *or* background structure.

Finally, the class of constraint databases over $\mathfrak{A}$ is denoted as $\mathrm{CDB}(\mathfrak{A})$ *and* $\mathrm{CDR}(\mathfrak{A})$ *denotes the class of constraint relations over $\mathfrak{A}$.*

Note that the representations of constraint relations are not necessarily unique, i.e. a relation can be represented by many different but equivalent formulae. Another subtle point in the definition of finite representations is the use of parameters. This becomes clear if we consider uncountable universes, the reals for instance. Obviously, a formula with arbitrary, i.e. transcendental parameters is no longer a finite representation. Thus, with such context structures, the set of elements allowed for parameters is usually restricted. This will be further explored later on.

The choice of the context structure depends on but also determines the application area in which the databases may be used. We now list some context structures typically considered in the constraint database framework.

- Dense order constraint databases are constraint databases over the real line $(\mathbb{R}, <)$. This class of databases is very well studied and enjoys a number of desirable properties, mainly because many questions about dense order constraint databases can be reduced to questions about finite databases via a back-and-forth translation of databases and queries using suitable interpretations. See [BST96, GK99, Kre99b] for instance.

- Another important context structure is the real ordered group $(\mathbb{R}, <, +)$. The relations representable in this database model are precisely the semi-linear sets. Linear constraint databases have been studied intensively in the literature, as they provide an adequate model for spatial databases and geographical information systems.

  Two different linear constraint database models have been studied. In the first model, the set of parameters allowed in the formulae is restricted to the integers, or the rationals, which gives the same set of representable relations. This is sometimes referred to as the ***Z**-linear database model.* In the second model, real algebraic parameters are allowed. It is more general than the first and consequently also more complicated. As rational coefficients are sufficient for the applications we have in mind, we only consider the first model and agree that linear constraint relations are represented by formulae over $(\mathbb{R}, <, +)$ with integer coefficients only. This will be discussed in more depth in Chapter 16.

- Polynomial constraint databases are constraint databases over the ordered field of reals $(\mathbb{R}, <, +, \cdot)$. In this database model, arbitrary semi-algebraic sets can be represented, making it a powerful model for all sorts of applications involving spatial information. However, contrary to the linear database model, the (practical) complexity of algorithms manipulating semi-algebraic sets is usually too high. Therefore, most current applications of spatial databases or geographical information systems use linear approximations to the spatial data and therefore effectively work in a linear database model.

Figure 15.1 shows an example of a rather simple database in the polynomial constraint database model. Note that effectively a constraint database is an infinite structure and has to be seen as such. The details of finite representations are hidden by the database management system (DBMS) implementing a constraint database model. Thus, the user works on an infinite database and the DBMS has to take care of translating the user actions on the database to actions operating on the representations. This issue will be of particular importance in the next section where we consider constraint queries.

In practical applications, constraint databases are often many-sorted. For instance, a geographical information system usually contains *spatial information* in form of an area map but also non-spatial or *thematic information*, for instance, names of buildings or streets, or the population

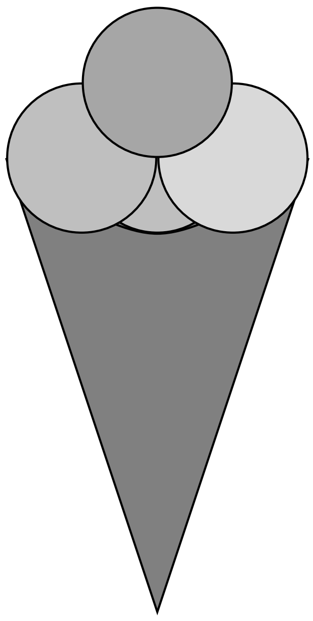


Figure 15.1: Constraint database defined by $\varphi(x,y) := (x-2)^2+(y-6)^2 \leq 1 \vee (x-1)^2+(y-6)^2 \leq 1 \vee (x-3)^2+(y-6)^2 \leq 1 \vee (x-2)^2+(y-7)^2 \leq 1 \vee (y \geq 6-3x \wedge y \leq 6 \wedge y \geq -6+3x)$

of an electoral district. Thus, the database relations will contain tuples $(x, y; z_1, \dots, z_k)$ where only x and y range over the infinite universe, e.g. $\mathbb{R}$, and the variables $\overline{z}$ contain the thematic information and range over a finite or at least countable set.

For theoretical research, this makes no difference as the tuples can also be seen as tuples in $\mathbb{R}^{k+2}$. In practise however, the complexity of many algorithms manipulating spatial databases depend heavily on the dimension of the involved relations. Here, knowing that the spatial part is only binary may help to speed up query processing and database manipulation significantly.

In this work, we only consider theoretical aspects of constraint databases, and therefore do not take thematic information into account. Instead, we focus on single-sorted databases where all involved variables range of the infinite universe.

15.2 Constraint Queries

In this section we address the problem of querying constraint databases. As already noted above, the details of finite representations are hidden by the database management system. Consequently, queries on such a database system can be formulated as if working on an infinite database. Therefore, queries have to be considered on two different levels of abstraction, namely on the conceptual level as functions from structures to relations, and on the physical level as functions from representations of constraint databases to representations of constraint relations.

Obviously, the two notions of queries do not necessarily coincide. Therefore, one usually imposes two restrictions to constraint queries, namely *consistency* and *closure*. Consistency means that the result of a query on two different representations of the same database always yields the same relation, although not necessarily the same representation of it. As we only consider queries defined by logical formulae, this criterion is trivially met and we agree that all queries have to be consistent in this sense. However, the second criterion, closure, will be important later on and is therefore formally defined in the next definition.

15.3 Definition. *Let $\mathfrak{A}$ be a structure and σ a relational signature. A* database query *Q is a mapping from constraint databases over $\mathfrak{A}$ to relations over $\mathfrak{A}$. This mapping may be partial, i.e. undefined for some databases. The query is* closed, *if for any constraint database $\mathfrak{B} \in \mathrm{CDB}(\mathfrak{A})$ such that the result of Q on $\mathcal{B}$ is defined, $Q(\mathfrak{B})$ is representable over $\mathfrak{A}$. A* constraint query *is a closed database query.*

By definition, queries are abstract functions from structures to relations. In practise, however, they are usually specified by means of a query language. We will consider various query languages which all have in common that they are based on first-order logic. Therefore we start our exploration on query languages with FO.

15.4 Definition. *Let $\mathfrak{A}$ be a context structure and σ a relational signature. A first-order formula $\varphi(\overline{x}) \in \mathrm{FO}[\tau \cup \sigma]$ with k free variables $\overline{x}$ defines the k-ary query*

$$\begin{array}{rccl} Q_\varphi : & \mathrm{CDB}(\mathfrak{A}) & \longrightarrow & \mathrm{CDR}(\mathfrak{A}) \\ & \mathfrak{B} & \longmapsto & \{\overline{a} \in A^k : \mathfrak{B} \models \varphi[\overline{a}]\}. \end{array}$$

Often, we write $\varphi(\mathfrak{B})$ instead of $Q_\varphi(\mathfrak{B})$.

There are context structures where first-order logic does not provide a closed query language. For instance, on databases over $(\mathbb{N}, <)$ the formula $\varphi(x, y) := \exists^{=1} z\, x < z < y$ expresses that x and y have distance 2. It is easily seen that there is no quantifier-free formula over $(\mathbb{N}, <)$ expressing this. We now introduce a property of context structures that guarantees closure of first-order queries.

15.5 Definition. *A complete theory T over a signature τ* admits quantifier elimination *if for every first-order formula $\varphi(\overline{x}) \in \mathrm{FO}[\tau]$ there is a quantifier-free formula $\varphi'(\overline{x}) \in \mathrm{FO}[\tau]$ such that $T \models \forall \overline{x}(\varphi \leftrightarrow \varphi')$.*

A structure $\mathfrak{A}$ admits quantifier elimination, if its theory $\mathrm{Th}(\mathfrak{A})$ *does. The quantifier elimination is called* effective, *if there is an effective method to compute an equivalent quantifier-free formula from a given formula φ.*

The following proposition shows that precisely in constraint databases over structures admitting quantifier elimination first-order logic provides a closed query language.

15.6 Proposition. *Let $\mathfrak{A}$ be a τ-structure. $\mathfrak{A}$ admits quantifier elimination if, and only if, every first-order formula is closed on the class of constraint databases over $\mathfrak{A}$.*

Proof. Suppose $\mathfrak{A}$ admits quantifier elimination. Let $\mathfrak{B} := (\mathfrak{A}, \Phi) \in \mathrm{CDB}(\mathfrak{A})$ be a σ-constraint database, where Φ is the set of formulae φ_R representing the database relations $R \in \sigma$ in $\mathfrak{B}$.

For any given $\varphi \in \mathrm{FO}[\tau \cup \sigma]$ consider the formula φ' obtained from φ by replacing every atom $R\overline{u}$, for $R \in \sigma$, by $\varphi_R(\overline{x}/\overline{u})$. It is easily seen that $\mathfrak{B} \models \forall\overline{x}(\varphi \leftrightarrow \varphi')$. As φ' is a pure $\mathrm{FO}[\tau]$ formula, there is a quantifier-free formula $\psi \in \mathrm{FO}[\tau]$ equivalent to φ'. Thus, ψ is a quantifier-free representation of $Q_\varphi(\mathfrak{B})$. The opposite direction is trivial. $\Box$

As the proposition shows, quantifier elimination is a necessary and sufficient condition for first-order logic to be closed on a class of constraint databases. Further requirements that are usually imposed on a context structure are that the quantifier-elimination procedure is effective and also that the context structure itself is recursive, so that containment of a tuple in a relation is decidable.

As proved above, on recursive context structures with effective quantifier elimination, first-order logic yields a decidable and closed query language. To measure the complexity of its evaluation problem, we first have to agree on a measure of size for the databases. Clearly, algorithms operating on constraint databases get the finite representation as input. As there is no canonical representation associated with a constraint database, we agree that whenever we speak about such a database we have a particular representation in mind. The size of the database is then measured in terms of this representation.

15.7 Definition. *The size $|\mathfrak{B}|$ of a constraint database $\mathfrak{B}$ is defined as the sum of the lengths of all formulae occurring in the representation of $\mathfrak{B}$.*

With this definition of size, we can now turn towards establishing complexity bounds for problems related to constraint queries. There are various different methods to measure the complexity of evaluation problems.

15.8 Definition. *Let $\mathfrak{A}$ be a context structure and $\mathcal{L}$ be a query language over $\mathfrak{A}$.*

- *The* evaluation problem *for a query formula $\varphi \in \mathcal{L}$ and a database $\mathfrak{B}$ over $\mathfrak{A}$ is defined as the problem of finding a finite representation of $Q_\varphi(\mathfrak{B})$.*
- *The* recognition problem *for φ and $\mathfrak{B}$ is defined as the problem of checking whether a given tuple $\overline{a}$ is in $Q_\varphi(\mathfrak{B})$.*

Recall that in Chapter 5, we only considered the evaluation problem for logics on finite structures. This was because for finite relational databases, the two problems are essentially equivalent. If the recognition problem can be solved, then the evaluation problem can also be solved, simply by applying the recognition problem to all possible tuples. However, in the context of constraint databases, this approach obviously fails as the relations are no longer finite. Solving the evaluation problem boils down to computing a representation of the answer – something that is not always possible for query languages which are not closed. On the other hand, solving the recognition problem corresponds to evaluating boolean queries for which closure is trivially given.

15.9 Definition. *Let $\mathfrak{A}$ be a context structure and $\mathcal{L}$ be a query language. The complexity of the evaluation problem can be measured in three different ways.*

- *For a fixed query formula $\varphi \in \mathcal{L}$, the* data complexity *of the query Q_φ is defined as the amount of resources (e.g. time, space, or number of processors) needed to evaluate the function that takes a representation Φ of a database $\mathfrak{B}$ to a representation of the answer relation $Q_\varphi(\mathfrak{B})$.*
- *For a fixed constraint database $\mathfrak{B}$ over $\mathfrak{A}$, the* query complexity of $\mathfrak{B}$ *is defined as the amount of resources needed to evaluate the function taking a query formula φ to the representation of the answer relation $Q_\varphi(\mathfrak{B})$.*
- *If both, the database and the query, are part of the input, we speak of the* combined complexity. *It is defined as the complexity of the function taking the pair $(Q_\varphi, \mathfrak{B})$ to $Q_\varphi(\mathfrak{B})$.*

In the sequel, we will mostly consider the data complexity of the evaluation problem and only rarely comment on the other complexity measures.

Why only quantifier-free representations? In Definition 15.1, finite representations of constraint relations were defined as quantifier-free formulae. One effect of this was, that the class of possible context structures had to be restricted to structures admitting effective quantifier elimination.

Another possibility is to allow arbitrary first-order formulae as representations and in fact there are authors who treat constraint databases in this way. With this definition, first-order query evaluation becomes trivial: We only have to substitute in the query the atoms involving database relations by the formulae defining them in the database. As first-order logic is an important query language, it sounds promising that query evaluation is so efficient. However, this approach raises some issues.

One problem is, that although evaluating a query is simple, displaying the result in a reasonable way might not. This is trivially seen when considering constraint databases over the arithmetic and boolean first-order queries. As explained above, evaluating such a formula, i.e. computing a representation of the answer, is trivial. However, giving the output in form of a *yes/no* answer instead of by a complicated formula describing it, is undecidable.

But even for decidable context structures, the simple approach is not satisfactory. Consider for instance polynomial constraint databases. Clearly, the user wants to view the result of a query on such a database as a semi-algebraic point set on his monitor and not as a first-order formula describing this set. But displaying a the set of points represented by an arbitrary first-order formula is a complex and non-trivial task and essentially boils down to computing a quantifier-free formulae equivalent to it. In this sense, nothing is gained by full first-order representations.

Chapter 16

The Linear Constraint Database Model

Linear constraint databases are constraint databases over the real ordered group $(\mathbb{R}, <, +)$. With their ability to store semi-linear sets, they form an appropriate model for spatial databases. An important application for such kind of databases are *Geographical Information Systems* (GIS) which are used to store geographical information, i.e. information about objects on or below the earth's surface. Typically, such a system contains *spatial information*, i.e. relations over the reals, in form of a map with additional non-spatial or *thematic information* about the objects in the map. The thematic information often comes from a finite or at least countable domain. For instance, local administrations provide maps of their town which list the historical usage of the various sites in the town. Potential purchasers of properties can then use these maps to asses the probability of having the site polluted with residual waste, e.g. oil pollution caused by former industrial usage.

Geographical information systems have numerous applications in geology, environmental sciences, or geography and become more and more important in these areas. As the (practical) complexity of algorithms manipulating semi-algebraic sets is too high and the accuracy achieved by linear approximations to such kind of data is usually sufficient, geographical information systems often use semi-linear sets to store spatial information.

In this chapter we study the linear constraint database model. We first present some structural properties of relations definable in this model. We then consider first-order logic as a linear constraint query language and demonstrate its deficiency in expressing several interesting queries. This leads to the definition of an extension of first-order logic in which these queries can be formalised. It will later be used as the basis for introducing fixed-point logics capable of defining precisely those closed queries on linear constraint databases that computable in polynomial time. The chapter is closed with the presentation of a second extension of first-order logic, called *PFOL*, which has been introduced by Vandeurzen et al. in [VGG98, Van99]. For this logic, it has been shown that there is a finite representation of semi-linear sets in terms of finite point sets such that the encoding and decoding can be defined in PFOL. This will frequently be used in the subsequent chapters.

16.1 Linear Constraint Databases

We first give a precise definition of linear constraint databases.

16.1 Definition. *The class of* linear constraint databases *is defined as the class of constraint databases over the context structure* $\mathfrak{A} := (\mathbb{R}, <, +)$*. The formulae representing linear constraint relations are allowed to use integer parameters.*

By definition, linear constraint relations are represented by quantifier-free formulae over the signature $\{=, <, +\}$. W.l.o.g. we can assume that the formulae representing the relations are in disjunctive normal form, i.e. of the form $\bigvee_i \bigwedge_j \varphi_{ij}$, where the φ_{ij} are atoms $p(\overline{x}) > 0, p(\overline{x}) = 0$ or $p(\overline{x}) < 0$ for polynomials $p(\overline{x}) := a_0 + \sum_{l=1}^{k} a_i x_i$. Here, the a_i are parameters from $\mathbb{N}$. As already noted above, allowing arbitrary real numbers as parameters is senseless, as then the formulae are no longer finite representations. Therefore, the set of elements from which the parameters may be chosen must be restricted to numbers which have a finite presentation.

In the literature, two different models of linear constraint databases have been considered. In the first model, sometimes called ***A**-linear constraint database model*, the parameters a_i may be arbitrary real algebraic coefficients whereas in the second model, sometimes called ***Z**-linear constraint database model*, only integer coefficients are allowed.

The restriction to integer coefficients seems rather restrictive. However, allowing rational coefficients in the formulae yields the same class of representable relations, as every equality $a_0 + \sum_{i=1}^{k} a_i x_i = 0$ with rational coefficients a_i can be transformed to an equality with integer coefficients by multiplication with the least common denominator.

Throughout the remaining chapters, we will only consider linear constraint databases with integer coefficients, as rational slopes are sufficient for almost all applications of linear constraint databases. Further, we are only interested in the spatial information stored in a database and do not consider the additional thematic information. Finally, as common in the literature on spatial constraint databases, we only consider databases with one spatial relation S. This simplifies the notation, but all results reported below extend to databases with more than one spatial relation.

16.2 Semi-Linear Sets

16.2.1 Structural Properties

In this section, we present some structural properties of linear constraint relations. It is obvious that the class of linear constraint relations coincides with the well known class of semi-linear sets, inductively defined by the following rules.

- For every linear inequality $p(\overline{x}) > 0$, the set $\{\overline{a} \in \mathbb{R}^k : p(\overline{a}) > 0\}$ is a semi-linear set.
- If $R, R' \subseteq \mathbb{R}^k$ are semi-linear, then so is $R \cap R'$, $R \cup R'$, and $\mathbb{R}^k - R$.

We recall some standard notions from geometry that will be used later on.

16.2 Definition. *Let $P \subseteq \mathbb{R}^d$ be a set of points.*

- *The* affine support *or* affine hull *of P is defined as the smallest affine subspace of $\mathbb{R}^d$ containing P.*
- *The dimension of P is defined as the maximal value d such that P contains a d-dimensional open hypercube. The dimension of the empty set is defined as -1.*
- *The* convex hull of P *is defined as*

$$\text{conv}(P) := \{x : \exists p_1, \ldots, p_n \in P, n \in \mathbb{N}, \exists a_1, \ldots, a_n \in \mathbb{R}^{\geq 0},\\ \Sigma a_i p_i = x,\ \Sigma a_i = 1\}.$$

  We define the open convex hull of P *as the interior of* conv(P) *with respect to its affine support. It is formally defined as*

$$\text{openconv}(P) := \{x : \exists p_1, \ldots, p_n \in P, n \in \mathbb{N}, \exists a_1, \ldots, a_n \in \mathbb{R},\\ a_i > 0,\ \Sigma a_i p_i = x,\ \Sigma a_i = 1\}.$$

- *A* polyhedron in $\mathbb{R}^d$ *is the intersection of finitely many open or closed halfspaces*[1]*. It is* bounded*, if it is entirely contained in some d-dimensional hypercube of edge length $l \in \mathbb{R}^{\geq 0}$. A bounded polyhedron is called a* polytope.

Recall from above that a linear constraint relation S is defined by a quantifier-free formula. W.l.o.g. we can think of this formula as being in disjunctive normal form, i.e. of the form $\varphi_S := \bigvee_i \bigwedge_j \varphi_{ij}$, where each φ_{ij}

[1] Usually, polyhedra in $\mathbb{R}^d$ are defined as the intersection of finitely many *closed* halfspaces in $\mathbb{R}^d$. For our purposes it is more convenient to allow the intersection with *open* halfspaces also.

is a linear inequality, defining a halfspace, or a linear equation, defining a hyperplane.

Thus, each conjunct in $\varphi_i := \bigwedge \varphi_{ij}$ defines a polyhedron and the relation S consists of a union of polyhedra.

In the following paragraphs we will study decompositions of semi-linear sets into a finite number of subsets satisfying some regularity conditions. For this, we often speak about the interior of subsets or of subsets being open. This is always meant with respect to the sets affine support. We agree on the following proviso.

Proviso. Whenever we speak about the interior of a set or a set being open, this is always with respect to its affine support. □

16.2.2 Arrangements

Let S be a semi-linear subset of $\mathbb{R}^d$. We present a method to decompose the space $\mathbb{R}^d$ into finitely many partitions, i.e. convex, connected, and disjoint sets, such that the set $S \subseteq \mathbb{R}^k$ can be composed from a subset of these partitions. The presentation follows [Ede87]. See also [GO97, Zie95] for details.

Let φ_S be the formula defining the set S. Let $\mathfrak{G}(S) := \{g_1, \dots, g_n\}$ be the set of atoms, i.e. linear (in)equalities, occurring in φ_S. Consider the set of hyperplanes

$$\mathfrak{H}(S) := \{h : \begin{array}{l} h \text{ is an equation from } \mathfrak{G}(S) \text{ or there is some in-} \\ \text{equality } g \in \mathfrak{G}(S) \text{ and } h \text{ is obtained by replacing} \\ \text{in } g \text{ the inequality by equality.} \end{array} \}$$

See Figure 16.1 and 16.2 for a spatial relation and the corresponding set of hyperplanes.

Note that two atoms $p(\overline{x}) > 0$ and $p(\overline{x}) < 0$ occurring in φ_S correspond to the same equality $p(\overline{x}) = 0$ in $\mathfrak{H}(S)$. Thus, the size $m := |\mathfrak{H}(S)|$ of $\mathfrak{H}(S)$ might be less than the number of different atoms in φ_S.

For every $h := \Sigma a_i x_i = b \in \mathfrak{H}(S)$, we define the set of points being *above, on*, or *below* h, so that a point $p := (p_1, \dots, p_d)$ is above h if $\Sigma a_i p_i > b$, on h if $\Sigma a_i p_i = b$, and below h if $\Sigma a_i p_i < b$. Let h^- denote the set of points below h and h^+ the set of points above h. Clearly, any two points on the same side of all hyperplanes in $\mathfrak{H}(S)$ are either both contained in or both outside of S. To see this, recall that the hyperplanes in $\mathfrak{H}(S)$ arise from the atoms in φ_S. Points on the same side of a hyperplane h cannot be separated by the corresponding atom. Clearly, if the points cannot be separated by the atoms in φ_S, then the boolean combination of the atoms cannot separate them either.

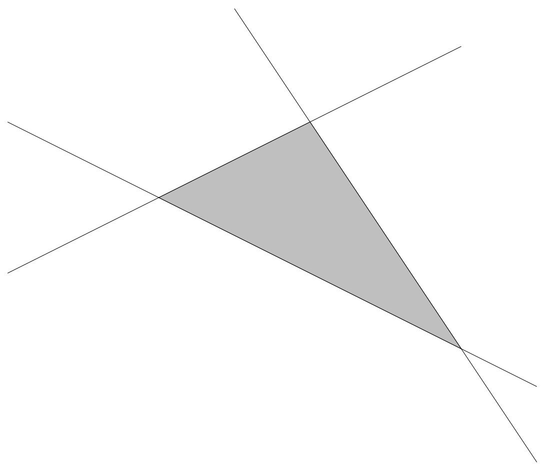


Figure 16.1: Example of a spatial relation S.

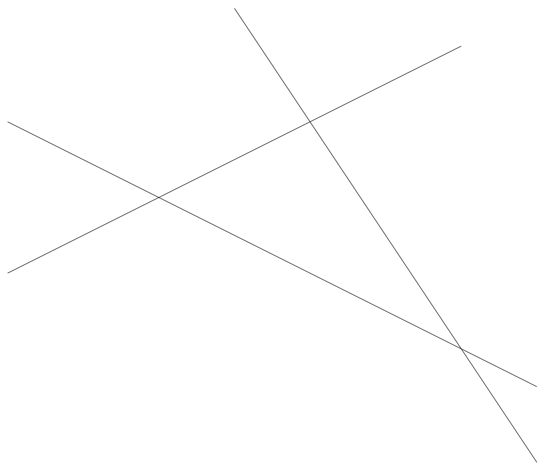


Figure 16.2: Set of hyperplanes induced by S.

Let p be a point in $\mathbb{R}^d$. The *position* $v_i(p)$ with respect to $h_i \in \mathfrak{H}(S)$ is defined as

$$v_i(p) := \begin{cases} +1 & \text{if } p \in h_i^+, \\ 0 & \text{if } p \in h_i, \\ -1 & \text{if } p \in h_i^-. \end{cases}$$

The position of a point with respect to $\mathfrak{H}(S)$ is determined by the vector $(v_1(p), \ldots, v_m(p))$, called the *position vector of* p, and any two points sharing the same position vector are inseparable by φ_S.

We call a set containing all points sharing the same position vector a *face* and the dissection of $\mathbb{R}^d$ into faces induced by the set $\mathfrak{H}(S)$ of hyperplanes an *arrangement* $\mathcal{A}(S)$. This dissection of $\mathbb{R}^d$ into faces is a partition of $\mathbb{R}^d$ with the property that every face is either contained in or disjoint to S. See Figure 16.3 for the decomposition of the database shown above.

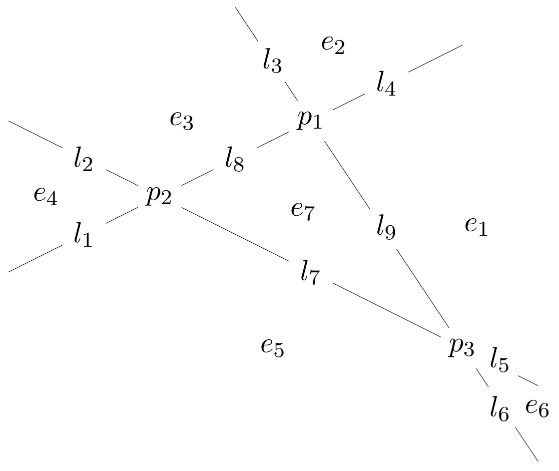


Figure 16.3: Arrangement $\mathcal{A}(S)$.

The *dimension of a face* in the decomposition is defined as the dimension of its affine support. Thus, in the example above, there are seven two-dimensional faces e_1 to e_7, nine one-dimensional faces l_1 to l_9, and three zero-dimensional faces p_1 to p_3. As usual, we call zero-dimensional faces *vertices*.

Two faces f and g are *incident*, if f is of dimension one less than g and it is contained in the boundary of g or, conversely, g is of dimension one less than f and it is contained in the boundary of f.

Typically, arrangements are stored in a data structure like the *incidence graph*. The incidence graph contains a vertex for every face in the arrangement as well as two additional vertices, one representing a virtual (-1)-dimensional face, denoted by $\varnothing$, which is incident to every 0-dimensional face, and one vertex representing a $(d+1)$-dimensional face, written as $\mathcal{A}(S)$, incident to every d-dimensional face. We call the last two vertices *improper* and the other vertices *proper*. Each proper vertex v stores the position vector of the points contained in the corresponding face f as well as two lists of directed edges, one containing edges pointing at the vertices whose face is incident to f and one containing edges pointing at the vertices whose face f is incident to. Figure 16.4 shows the incidence graph for the part of the arrangement of Figure 16.3 containing the faces around p_2.

As every vertex stores the position vector of the corresponding face, a conjunction of atoms defining the face can easily be obtained from $\mathfrak{H}(S)$ and the incidence graph for $\mathcal{A}(S)$. Also the incidence relation can be efficiently

$$\begin{tikzcd}
 & & & \mathcal{A}(S) \arrow[dll, leftrightarrow] \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] \arrow[drr, leftrightarrow] & & & \\
\cdots & e_5 \arrow[d, leftrightarrow] \arrow[drrr, leftrightarrow] & e_4 \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] & & e_3 \arrow[dll, leftrightarrow] \arrow[dr, leftrightarrow] & e_7 \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] & \cdots \\
\cdots & l_1 \arrow[drr, leftrightarrow] & l_2 \arrow[dr, leftrightarrow] & & l_7 \arrow[dl, leftrightarrow] & l_8 \arrow[dll, leftrightarrow] & \cdots \\
 & & \cdots & p_2 \arrow[d, leftrightarrow] & \cdots & & \\
 & & & \varnothing & & &
\end{tikzcd}$$

Figure 16.4: Part of the incidence graph for $\mathcal{A}(S)$.

decided.

It is known, that an arrangement for a set of n hyperplanes in $\mathbb{R}^d$ can be computed in time $\mathcal{O}(n^d)$. See Theorem 7.6 in [Ede87]. Besides the known sequential algorithms, attempts have been made to define parallel algorithms for computing arrangements [ABB96, Goo93], leading to algorithms running in parallel time $\mathcal{O}(\log n)$ using a polynomial number of processors on a CREW PRAM (See [Goo93]).

Since the number of hyperplanes in $\mathfrak{H}(S)$ is always less than or equal to the number of atoms in the representation of the database, the following theorem is immediate.

16.3 Theorem. *Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a linear constraint database. The arrangement $\mathcal{A}(S)$ can be computed in polynomial time with respect to the size of the representation of $\mathfrak{B}$.*

16.3 First-Order Logic as Linear Constraint Query Language

In this section we consider first-order logic as query language for linear constraint databases. The linear constraint database model is among the most important constraint models considered so far in the literature. Consequently, a lot is already known about the properties and weaknesses of first-order queries on linear constraint databases. As for finite databases, first-order logic fails to express queries relying on recursion, connectivity for instance. However, whereas on finite databases most queries of daily use are expressible in FO, or, equivalently, SQL, the situation is different in the constraint framework. It has been shown that also queries like defining the convex closure of points, defining their affine support, testing whether points are collinear, and many others are not expressible in FO. See [KLP00, chapter 4 and 9] and references therein for results on the expressive power of FO.

Besides first-order queries using order and addition, several authors considered queries which may also use multiplication. Therefore they refer to FO$[<,+]$ as FO+Lin and to FO$[<,+,\cdot]$ as FO+Poly. As we do not consider multiplication at all it is not necessary to make this distinction.

We first mention some queries not definable in first-order logic.

16.4 Theorem. *Let $\mathfrak{B} := (\mathfrak{A}, S)$ be a constraint database such that S contains only finitely many points. The following queries on linear constraint databases are not definable in first-order logic.*

(1) *Define the convex hull of the points in S.*

(2) *Define the affine support of the points in S.*

Proof. We prove (1) first by showing that with the definability of convex closure multiplication becomes definable also. The definition of multiplication by convex closure is demonstrated in Figure 16.5.

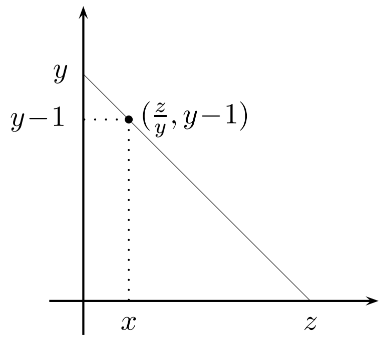


Figure 16.5: Defining multiplication by convex closure.

The relation *mult*(x, y, z), true for x, y, and z if $x \cdot y = z$, can be defined as follows. W.l.o.g. we assume that x, y, and z are positive. Consider the points $(0, y)$ and $(z, 0)$ in the plane. The convex closure of these points is the line segment as shown in Figure 16.5. The point on the line segment having $(y-1)$ as second coordinate has the first coordinate $\frac{z}{y}$. Now, if $(x, y-1) \in \text{conv}\{(0, y), (z, 0)\}$, then $x = \frac{z}{y}$ and $x \cdot y = z$.

Thus, if the convex closure was definable in first-order logic, then every semi-algebraic set would be definable in FO with order and addition and thus be semi-linear. As it is known that there are semi-algebraic sets which are not semi-linear, we get a contradiction.

The same proof also establishes (2) as the convex hull is a subset of the affine support. $\square$

The theorem shows that plain first-order logic is too weak to define various important queries on linear constraint databases. We now consider an extension of FO which allows the definition of the convex hull of finite point sets. As the proof of Theorem 16.4 shows, we can not hope for full definability of convex hulls.

16.3.1 Extending First-Order Logic by Convex Hulls

In this section we define an extension of first-order logic on semi-linear databases such that with each definable finite set of points also its convex hull becomes definable.

16.5 Definition. *Let $\overline{x}_i, \overline{x}'_i, \overline{y}$, and $\overline{z}$ be sequences of distinct variables such that $|\overline{x}_i| = |\overline{x}'_i| = |\overline{y}| = l$ for some l. The logic FO(conv) is defined as the extension of first-order logic by the following rules:*

(i) If $\varphi \in$ FO(conv) is a formula with free variables $\{\overline{x}_1, \dots, \overline{x}_k, \overline{z}\}$ then $\psi := [\mathrm{conv}_{\overline{x}_1,\dots,\overline{x}_k}\varphi](\overline{y}, \overline{z})$ is also a formula, with free variables $\{\overline{y}, \overline{z}\}$.

(ii) If $\varphi \in$ FO(conv) is a formula with free variables $\{\overline{x}_1, \overline{x}'_1, \dots, \overline{x}_k, \overline{x}'_k, \overline{z}\}$ then $\psi := [\mathrm{uconv}_{\overline{x}_1,\overline{x}'_1,\dots,\overline{x}_k,\overline{x}'_k}\varphi](\overline{y}, \overline{z})$ is also a formula, with free variables $\{\overline{y}, \overline{z}\}$.

The semantics of the additional operators is defined as follows. Let $\mathfrak{B}$ be the input database and $\psi := [\mathrm{conv}_{\overline{x}_1,\dots,\overline{x}_k}\varphi](\overline{y}, \overline{z})$ be a formula in *FO(conv).* Let $\varphi^{\mathfrak{B}}$ be the result of evaluating the formula φ in $\mathfrak{B}$. If $\varphi^{\mathfrak{B}}$ is infinite, then $\psi^{\mathfrak{B}} := \varnothing$. Otherwise,

$$\psi^{\mathfrak{B}} := \{(\overline{a}, \overline{b}) : \overline{a} \in \bigcup \{\mathrm{conv}\{\overline{a}_1, \dots, \overline{a}_k\} : \mathfrak{B} \models \varphi[\overline{a}_1, \dots, \overline{a}_k, \overline{b}]\}\},$$

where $\mathrm{conv}\{\overline{a}_1, \dots, \overline{a}_k\}$ denotes the interior (with respect to the affine support) of the convex closure of $\{\overline{a}_1, \dots, \overline{a}_k\}$.

The semantics of the uconv operator is defined similarly. The motivation for it is, that every set defined by the conv operator is bounded. To overcome this restriction, the uconv operator is designed to handle "points at infinity". Let $\psi := [\mathrm{uconv}_{\overline{x}_1,\overline{x}'_1,\dots,\overline{x}_k,\overline{x}'_k}\varphi](\overline{y}, \overline{z})$ be a formula and let $\mathfrak{B}$ be the input database. Again, if $\varphi^{\mathfrak{B}}$ is infinite, then $\psi^{\mathfrak{B}} := \varnothing$. Otherwise,

$$\varphi^{\mathfrak{B}} := \{(\overline{a}, \overline{b}) : \begin{array}{c} \text{there are } \overline{a}_1, \overline{a}'_1, \dots, \overline{a}_k, \overline{a}'_k \text{ such that} \\ \mathfrak{B} \models \varphi[\overline{a}_1, \overline{a}'_1, \dots, \overline{a}_k, \overline{a}'_k, \overline{b}] \text{ and } \overline{a} \in \mathrm{conv}(\bigcup_{i=1}^{k} \mathrm{line}(\overline{a}_i, \overline{a}'_i)) \end{array}\},$$

where $\mathit{line}(\overline{a}_i, \overline{a}'_i) := \{\overline{x} : (\exists b \in \mathbb{R}^{\geq 0}) \text{ such that } \overline{x} = \overline{a}_i + b(\overline{a}'_i - \overline{a}_i)\}$ defines the half line with origin $\overline{a}_i$ going through $\overline{a}'_i$.

Intuitively, each pair $(\overline{a}_i, \overline{a}'_i)$ represents two points, the point $\overline{a}_i$ and the point "reached" when starting at $\overline{a}_i$ and going in the direction of $\overline{a}'_i$ to infinite distance. Now uconv returns the union of the open convex closure (with respect to its affine support) of the $2k$ points represented by each tuple $((\overline{a}_{i,1}, \overline{a}'_{i,1}), \ldots, (\overline{a}_{i,k}, \overline{a}'_{i,k}))$.

Note that the condition on the formula φ to define a finite set is purely semantical. Since finiteness of a semi-linear set is first-order definable - e.g. by the formula $\mathit{finite}(\varphi)$ stating that there are $\varepsilon, \delta > 0$ such that the Manhattan-distance between any two points in $\varphi^{\mathfrak{B}}$ is greater than ε and each point in $\varphi^{\mathfrak{B}}$ is contained in a hypercube of edge length δ - this condition can also be ensured on a syntactical level, giving the language an effective syntax.

We now present an example of a query definable in *FO(conv)*.

16.6 Example. *Let $\varphi(x)$ be a formula defining a finite set. The query*

$$\mathit{mult}_\varphi(x, y, z) := \{(a, b, c) : a \text{ satisfies } \varphi \text{ and } a \cdot b = c\}$$

is definable in FO(conv). We give an explicit formula only for the case where $x, y, z \geq 0$. The other cases are analogous.

Let x_1, x'_1 and x_2, x'_2 be pairs of variables and let $\psi(x_1, x'_1, x_2, x'_2; x) := (x_1, x'_1) = (0, 0) \wedge (x_2, x'_2) = (1, x)$ be a formula defining for each x the pair of points $(0, 0)$ and $(1, x)$ in $\mathbb{R}^2$. Then

$$\psi(x, y, z) := [\mathrm{uconv}_{x_1, x'_1, x_2, x'_2} \varphi(x) \wedge \psi](y, z; x)$$

defines the query mult_φ. The uconv operator defines - for the parameter x - the half line with origin $(0, 0)$ and slope x. Thus the point (y, z) is on this line if, and only if, $x \cdot y = z$.

16.7 Note. *The example shows that atoms of the form $x \cdot_\varphi y = z$ can be expressed in FO(conv), with the semantics that x satisfies φ and $x \cdot y = z$, where φ is required to define a finite set. From now on, we allow atoms of this form in FO(conv)-formulae.*

We first show that *FO(conv)* is a closed linear constraint query language. We also establish an upper bound on the complexity of the query evaluation problem.

16.8 Theorem. *FO(conv) is closed and has* PTIME *data-complexity.*

Proof. Closure of the conv-operator follows from the finiteness of the sets, of which the convex closure is taken. PTIME data-complexity can easily be established by induction on the structure of the queries. $\square$

Extensions of first-order logic on linear constraint databases have extensively been studied in the literature. In the next section we introduce another such extension, called *PFOL*, for which several interesting queries have been shown to be expressible.

16.3.2 Extending First-Order Logic by Multiplication

In this section we present a query language called *PFOL*, an extension of FO+Lin by a restricted form of multiplication. The language has been introduced by Vandeurzen et al. in [VGG98]. See also [Van99] for a detailed study of its properties.

In PFOL, there are two different kinds of variables, *real variables* and so-called *product variables*. A PFOL program consists of a sequence of formulae $(\varphi_1(x), \ldots, \varphi_k(x); \varphi(\overline{x}))$, where each φ_i is required to define a finite set $D_i \subset \mathbb{R}$. The formulae are allowed to use multiplication between variables $x \cdot p$, but at least p must be a product variable. All product variables must be bound at some point by a quantifier of the following form. In φ_i the quantifiers for the product variables are of the form $\exists p \in D_j$, for some $j < i$. In φ all D_i may be used to guard the variable.

Essentially, the formulae φ_i successively define a sequence of finite sets and then at least one factor of every multiplication in φ is restricted to one of these sets. In the original definition of PFOL there were also terms $t = \sqrt{|p|}$. We do not take this term building rule into account and allow only multiplication. Note that the square root operator strictly increases the expressive power of the language. Therefore we call the language considered here *restricted PFOL.*

It is known that convex closure can be defined in restricted PFOL. Conversely, in Example 16.6 we already saw that multiplication with one factor bounded by a finite set can be defined in *FO(conv).* Thus, the proof of the following theorem is immediate.

16.9 Theorem. *Restricted PFOL = FO(conv).*

The previous theorem implies that the results on expressiveness proved for PFOL carry over to *FO(conv).*

In the next section we will show how finite representations of semi-linear sets can be defined in PFOL. This result, which has been obtained by Vandeurzen et al. in [VGG98], will frequently be used throughout the remaining chapters.

16.3.3 Finite Representations of Semi-Linear Sets

In this section, we aim at defining a representation of semi-linear sets S by a finite set of points, such that the set S can be recovered from its representation. Further, encoding and decoding is PFOL-definable.

Recall again the discussion of arrangements of semi-linear sets $S \subseteq \mathbb{R}^d$ in Section 16.2.2. An arrangement $\mathcal{A}(S)$ of S is a partition of $\mathbb{R}^d$ into finitely many regions, the *faces*, such that for each face R either $R \subseteq S$ or $R \cap S = \varnothing$. By construction, an arrangement consists of faces of different dimension. Besides the faces of maximal dimension, i.e. the dimension of S, there are also faces of lower dimension. In particular, the arrangement contains zero-dimensional faces, i.e. individual points.

Consider again Figure 16.3 which shows the arrangement of the example in Section 16.2.2. Obviously, the face e_7 can be obtained from the zero-dimensional faces p_1, p_2, and p_3 by taking the interior of their convex hull.

This is a general phenomenon of arrangements, i.e. every bounded face can be defined by the convex hull of points in the arrangement. However, faces can be fairly complex themselves and there is not necessarily a fixed number k, depending only on the dimension d of the underlying space $\mathbb{R}^d$, such that any face in the arrangement of a semi-linear set $S \subseteq \mathbb{R}^d$ can be obtained from the open convex hull of no more than k points in the arrangement.

Here we are mainly interested in a decomposition of semi-linear sets $S \subseteq \mathbb{R}^d$ into a finite set $S^{enc} \subseteq \mathbb{R}^k$ of points, such that S can be recovered from S^{enc} and both, the encoding of S into S^{enc} and the decoding of S^{enc} into S is definable in restricted PFOL.

It has been shown in [Van99] that such a decomposition exists indeed.

16.10 Theorem. *For every $d < \omega$, there are PFOL-formulae* code *and* decode *such that on every semi-linear set $S \subseteq \mathbb{R}^d$* code *defines a finite relation $S^{\mathrm{enc}} \subseteq \mathbb{R}^{(d+1)^2}$ and on every finite set $D \subseteq \mathbb{R}^{(d+1)^2}$* decode *defines a set $T \subseteq \mathbb{R}^d$ such that*

$$(\mathrm{decode} \circ \mathrm{code})(S) = S.$$

A detailed construction of the decomposition and the formulae defining it can be found in [VGG98] and also in [Van99, Section 5.3.1].

Chapter 17

Complete Query Languages

In this chapter we consider the fixed-point logics introduced in Part I as candidates for linear constraint query languages. The main result of this section is to show that transitive-closure logic is expressive complete on the class of linear constraint databases in the sense that every partially computable query is definable in TC.

However, care has to be taken on what the partially computable query defined by a formula in TC is meant to be. Whereas on finite databases the result of a TC-formula is always computable, the precise method used to compute it is irrelevant for the query defined by this formula. On linear constraint databases, however, it is easily seen that the arithmetic is TC-interpretable in a linear constraint database. Thus, the result of a TC-query no longer needs to be computable. Therefore, the partially computable query defined by such a formula depends on the precise operational semantics, i.e. evaluation method, given to TC as with some evaluation schemata the result of a particular query might be computable whereas others may fail to do so.

Formally, the standard *model-theoretical semantics* for a fixed-point logic $\mathcal{L}$ as presented in Part I can be defined as a function

$$\text{mod} : \varphi \in \mathcal{L} \longmapsto (\text{mod}_\varphi : (\mathfrak{A}, \sigma) \longrightarrow \text{Pow}(A^*))$$

taking formulae $\varphi \in \mathcal{L}$ to functions mod_φ which map a constraint database $\mathfrak{B} := (\mathfrak{A}, \sigma)$ to the set of tuples satisfying φ on $\mathfrak{B}$. In the constraint database setting, the problem arises that the functions mod_φ may not be computable or the result may not be representable over $\mathfrak{A}$. For instance, on any linear constraint database $\mathfrak{B}$ the formula

$$\varphi(n) := [\mathbf{dtc}_{x,y}\, y = x + 1](0, n)$$

defines the set of natural numbers. As this set is not representable over $\mathfrak{A} := (\mathbb{R}, <, +)$, this shows that already deterministic transitive-closure logic yields a query language which is not closed in the sense of Definition 15.3.

Further, it is clear that multiplication and thus full arithmetic is definable by DTC-formulae over linear constraint databases. This shows that even for DTC, the functions mod_φ above may not even be arithmetical and the results may not be finitely representable.

As the example demonstrates, on linear constraint databases there are queries in DTC whose result according to the model-theoretical semantics is not computable, independent of the particular evaluation scheme used. However, there might also be queries such that their result can be computed by one evaluation scheme but not by another. For instance, consider the query $[\mathbf{tc}_{x,y} y = x + 1](0, 10)$. A naive evaluation scheme would begin the evaluation by first computing the transitive closure of the formula $y = x+1$. This done, it would check whether the pair $(0, 10)$ occurs in it. As the transitive closure of $y = x+1$ is infinite, the evaluation procedure would fail to compute the correct answer. A more clever evaluation scheme, however, would start at 0 and keep adding 1 to it until 10 is reached.

Thus, in the constraint database setting, statements like TC capturing the class of partially computable queries can only be made with reference to a fixed operational semantics.

Therefore we equip the fixed-point logics with an operational semantics that assigns to every formula of the logic a partially recursive function taking constraint databases to constraint relations.

17.1 Operational Semantics for Fixed-Point Logics

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

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

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

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

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

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

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

- *If* $\varphi \in \mathrm{FO}$, *define* $\mathrm{op}_\varphi(\mathfrak{B}) := \varphi'$, *where* φ' *is obtained from* φ *by first substituting each occurrence of a database relation symbol by the formula defining the relation in* $\mathfrak{B}$ *and then eliminating the quantifiers using the quantifier elimination method fixed above.*

- *If* $\varphi := \varphi_1 \wedge \varphi_2$, *define* $\mathrm{op}_\varphi(\mathfrak{B})$ *as* $((\mathrm{op}_{\varphi_1}(\mathfrak{B})) \wedge (\mathrm{op}_{\varphi_2}(\mathfrak{B})))$. *For other the boolean connectives, the function* op_φ *is analogously defined.*

- *If* $\varphi := \exists x \varphi_1$, *define* $\mathrm{op}_\varphi(\mathfrak{B})$ *as the result of applying the quantifier-elimination method fixed above to* $\exists x(\mathrm{op}_{\varphi_1}(\mathfrak{B}))$.

- *Now suppose that* φ *is of the form* $\varphi := [\mathbf{tc}_{\overline{x},\overline{y}}\, \psi(\overline{x}, \overline{y})](\overline{u}, \overline{v})$. *Recall that we allowed the use of integers as constants in the formulae. Let* I *be the indices of constants among* $\overline{u} := u_1, \ldots, u_n$, *i.e.,* u_i *is a constant if, and only if,* $i \in I$. *We inductively define formulae* σ_i, $i \in \omega$, *as follows.*

  (*i*) $\sigma_0 := \mathrm{op}_\psi(\mathfrak{B})(\overline{x}, \overline{y}) \wedge \bigwedge_{i \in I} u_i = x_i$.

  (*ii*) $\sigma_{i+1} := \sigma_i(\overline{x}, \overline{y}) \vee \exists \overline{x}' \left(\sigma_i(\overline{x}, \overline{x}') \wedge \mathrm{op}_\psi(\mathfrak{B})(\overline{x}', \overline{y})\right)$.

  If there is no $j \in \omega$ *such that* σ_j *and* σ_{j+1} *are equivalent in* $\mathfrak{B}$, *then* $\mathrm{op}_\varphi(\mathfrak{B})$ *is undefined. Otherwise let* i *be the smallest natural number such that* σ_i *and* σ_{i+1} *are equivalent and define* $\mathrm{op}_\varphi(\mathfrak{B})$ *as the result of applying the quantifier elimination method to the formula*

$$\exists \overline{x} \exists \overline{y} \left((\overline{u} = \overline{x}) \wedge (\overline{y} = \overline{v}) \wedge \sigma_i(\overline{x}, \overline{y})\right).$$

Finally, define the operational semantics $\mathrm{op}_{\mathrm{TC}}$ *for* TC *as the function taking formulae* φ *to* op_φ.

Observe the difference in the way the formula σ_0 is defined between this operational semantics and the standard way to define the model-theoretical semantics as outlined in Definition 4.1. The conjunct $\bigwedge_{i \in I}(u_i = x_i)$ reduces the computation of the transitive closure to the computation of all tuples which are reachable from tuples with some fixed components. Thus, by letting constants occur in the tuple $\overline{u}$ one gets some control over the process of building up the transitive closure. However limited this control might seem, we will show below that it is enough to allow the definition of arbitrary partially computable queries over databases from CDB, whereas it seems unlikely that this is also possible without this modification.

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

17.3 Proposition. *The operational semantics of Definition 17.2 is consistent with the model-theoretical semantics of* TC*.*

17.2 Expressive Completeness of Transitive Closure Logic

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

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

We show now that all partially computable functions on constraint databases over $(\mathbb{R}, <, +)$ can be defined in TC. The proof runs along the following line. We first show that the logic PFOL as defined in Section 16.3.2 above is contained in TC. This enables us to use the results on PFOL-definable finite representations discussed in Section 16.3.3. We then show that the run of Turing-machines on such finite representations can be simulated in TC. The main issue addressed in the proof is not to show that transitive-closure logic is expressive enough but to guarantee that the various **tc**-operators used in the formulae are actually terminating in cases where the query to be simulated is defined on a database.

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

To show that PFOL $\subseteq$ TC it suffices to prove that atoms $x \cdot p = y$ are definable in TC by a formula whose evaluation always terminates.

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

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

The formula φ_{im} is defined as

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

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

We now give the definition of the formula *mult* which makes use of a formula φ_{nd} defined below. Let *mult* be defined as

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

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

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

$$\begin{aligned} \varphi_{nd}(p,n,d) := \ & \varphi_p(p) \land d \cdot^{\varphi_p}_{i} p = n \ \land \\ & [\mathbf{tc}_{x,y,x',y'} \neg x = y \cdot^{\varphi_p}_{i} p \land \gamma(x,y,x',y')\ ](1,0,n,d), \end{aligned}$$

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

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

The proof of the following lemma is now straightforward.

17.5 Lemma. *Every PFOL formula, where all product variables are bound by formulae defining sets of rationals, is equivalent to a formula in* TC *whose evaluation always terminates.*

Recall from Section 16.3.3 the definition of an encoding of semi-linear sets by finite point sets. In Theorem 16.10, we showed that there are PFOL queries *code* and *decode*, such that for a given databases $\mathfrak{B} := ((\mathbb{R},<,+), S^{\mathfrak{B}})$, where S is k-ary, *code* defines a finite set $S^{\text{enc}} \subseteq \mathbb{R}^{(k+1)^2}$ of $(k+1)$-tuples of points in $\mathbb{R}^{k+1}$, and $S^{\mathfrak{B}}$ can be recovered from this finite encoding S^{enc} by a formula *decode*, i.e. $S = \{\overline{a} : (\mathbb{R},<,+) \models \text{decode}(S^{\text{enc}})\}$.

As, by definition, all parameters occurring in the formulae defining the database relations are rational, and therefore also all points in the encoding have rational coordinates, Lemma 17.5 implies that such a finite encoding of the database relation can be defined in TC.

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

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

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

The run of M will be simulated by the formula φ_M. We use bold face letters $\mathbf{q_0}, \mathbf{q_f}, \dots, \mathbf{a}, \mathbf{m}$ to denote fixed constants of the Turing-machine, e.g. states $\mathbf{q_0}$, symbols $\mathbf{a}$ of the alphabet, or $\mathbf{m} \in \{-1, 0, 1\}$.

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

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

(1) $\mathit{init}(x'_l, x'_r, t', s')$ becomes true for the tuple x'_l, x'_r, t', s' coding the input configuration, i.e. $x'_l = 0$, x'_r codes the input, $s' = \mathbf{q_0}$, i.e. the machine is in the initial state, and, finally, $t' = 0$.

(2) $\mathrm{compute}(x_l, x_r, t, s; x'_l, x'_r, t', s')$ becomes true for a pair of tuples, if the x'_l, x'_r, t', s' codes the successor configuration of the configuration coded in x_l, x_r, t, s.

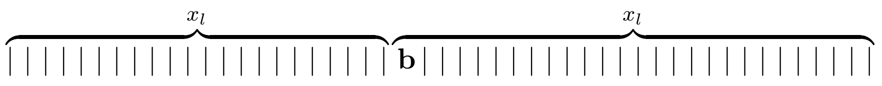
The conjunct $s \neq \mathbf{q_f}$ is needed to terminate the evaluation of the formula once the machine reaches the final state q_f.

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

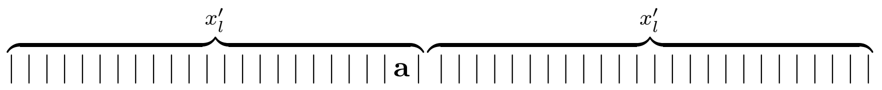
(i) *move-right* is defined as

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

Consider the following situation:



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



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

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

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

(iii) *move-left* is defined as

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

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

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

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

The formula init is defined as

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

with start being defined as

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

Here the formulae *max*, *min*, and *succ* are defined with respect to the lexicographical ordering of the points in the encoding S' and the formula $\text{append}(x', p', x)$ defines x' to code the tape inscription obtained from the inscription coded in x with the bit representation of p' being appended to the end. It is defined as

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

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

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

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

Now, the proof of the following lemma is straightforward.

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

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

Thus we have shown the following theorem.

17.7 Theorem. *Under the operational semantics defined above, the linear constraint queries definable in transitive-closure logic are precisely the partially computable queries on constraint databases over* $(\mathbb{R}, <, +)$.

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

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

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

In the following sections we consider other fixed-point logics and the classes of queries definable in them. We begin with stratified and least fixed-point logic.

17.3 Completeness of SFP and LFP

Clearly, SFP is more expressive than TC. Thus, Theorem 17.7 generalises to SFP and LFP in the sense that each partially computable query can be defined in these logics. However, a bit care has to be taken on whether the formulae terminate in the cases where the query is computable.

Let $\varphi := [\mathbf{tc}_{\overline{x},\overline{y}}\psi(\overline{x},\overline{y})](\overline{u},\overline{v})$ be a TC-formula. Then φ can inductively be translated to the equivalent SFP-formula $\varphi^* := [\mathbf{sfp}_{R,\overline{x},\overline{y}}\psi^*(\overline{x},\overline{y}) \vee \exists\overline{z}\, R\overline{x}\overline{z} \wedge \psi^*(\overline{z},\overline{y})](\overline{u},\overline{v})$. However, under the standard operational semantics, this formula might not terminate although, given the operational semantics above, the TC-formula does. To avoid this we recursively translate φ to $\varphi^* := [\mathbf{sfp}_{R,\overline{x},\overline{y}}(\overline{x} = \overline{u} \wedge \psi^*(\overline{x}, \overline{y})) \vee \exists \overline{z}(R\overline{x}\overline{z} \wedge \psi^*(\overline{z}, \overline{y}))](\overline{u}, \overline{v})$. This closely resembles the operational semantics we used for TC and thus guarantees termination of the formulae.

17.4 Existential Fixed-Point Logic

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

17.5 Dense Linear Orders

In this section we consider dense linear order databases, e.g. constraint databases over $(\mathbb{R}, <, +)$. Fixed-point logics on this class of databases have been studied in [BST99, GK99, Kre99b] where it is shown that questions about fixed-point queries on dense order databases can be reduced to the corresponding questions on finite databases. In [GK99] it is shown that for each dense linear order database $\mathfrak{B}$ there is a finite ordered database inv($\mathfrak{B}$) with universe B, called the *invariant of* $\mathfrak{B}$, such that

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

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

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

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

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

17.9 Theorem. *Stratified fixed-point logic and least fixed-point logic have the same expressive power on the class of finitely representable structures over the real line* $(\mathbb{R}, <)$. *Further,* TC *is less expressive than* SFP *provided that* NLogspace *is different from* Ptime.

Chapter 18

Tractable Query Languages

In the previous chapter, we have seen that transitive-closure logic provides an expressive complete query language for linear constraint databases. In this chapter, now, we present two tractable yet still expressive query languages. They are obtained by extending the logic *FO(conv)* defined in Section 16.3.1 by least fixed-point constructs. The resulting logics can be shown to express precisely those closed linear constraint queries which are computable in PTIME, i.e. the logics capture PTIME on the class of linear constraint databases.

18.1 The Logic *RegLFP(conv)*

Let $S \subseteq \mathbb{R}^d$ be a semi-linear set. Recall the definition of arrangements $\mathcal{A}(S)$ from Section 16.2.2. Arrangements are dissections of $\mathbb{R}^d$ into finitely many disjoint regions, i.e. connected subsets of $\mathbb{R}^d$, such that for each region R either $R \cap S = \varnothing$ or $R \subseteq S$. Thus, the input relation S can be written as a finite union of regions in its arrangement. Further, it was shown that for any fixed dimension, arrangements of semi-linear sets can be computed in polynomial time.

In this section we consider a query language which has access to the set of regions in an arrangement of the input database. This gives the logic a limited access to the representation of the database and, with it, increases its expressive power. Precisely, we consider a fixed-point logic where the fixed-point induction is defined over the finite set of regions. The semantics of the logic is defined in terms of certain two-sorted structures, called *region extensions*. Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a database and let $\mathcal{A}(S)$ be the set of regions in an arrangement of S. The logic then has separate variables and quantifiers for the reals and the set of regions.

We now give the formal definitions.

18.1 Definition (Region extensions). *Let* $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ *be a linear constraint database, where* S *is a* d*-ary relation, and let* $\mathcal{A}(S)$ *be the set of* *regions in an arrangement of S. The structure $\mathfrak{B}$ gives rise to a two-sorted structure $\mathfrak{B}^{Reg} := ((\mathbb{R}, <, +), S; Reg, adj)$, called the* region extension of $\mathfrak{B}$*, with sorts $\mathbb{R}$ and $Reg := \mathcal{A}(S)$ and the adjacency relation $adj \subseteq Reg \times Reg$, where two regions are adjacent if there is a point p in one of them such that every ε-neighbourhood of p has a non-empty intersection with the other region.*

The dimension of a region is defined as the dimension of its affine support, i.e. the dimension of the smallest affine subspace it is contained in. It is known that the number of regions in the arrangement is bounded polynomially in the size of the representation of S. As arrangements can be computed in polynomial time, the region extension of a given database is polynomial time computable also.

We now define the logic *RegLFP(conv)* which is *FO(conv)* augmented with a least fixed-point operator on the set of regions in the region extension. There are three different types of variables, so-called *element-*, *region-*, and *set variables*. Element variables will be interpreted by real numbers and region variables by regions in the region extension of the database. Set variables are equipped with a pair $(k, l) \in \mathbb{N}^2$ of arities. A set variable of arity (k, l) is interpreted by subsets $M \subseteq Reg^k \times \mathbb{R}^l$, such that for every $\overline{R} \in Reg^k$ the set $\{\overline{x} : (\overline{R}, \overline{x}) \in M\}$ is finitely representable.

18.2 Definition. *The logic RegLFP(conv) is defined on region extensions as the extension of FO(conv) by a least fixed-point operator. Precisely, the logic extends FO(conv) by the following rules.*

- *If R is a region variable and $\overline{x}$ is a sequence of element variables, then $R\overline{x}$ is a formula.*
- *If φ is a formula and R a region variable, then $\exists R\varphi$ is also a formula.*
- *If M is a set variable of arity (k, l), $\overline{R} := R_1, \ldots, R_k$ is a sequence of region variables, and $\overline{x} := x_1, \ldots, x_l$ is a sequence of element variables, then $M\overline{R}\overline{x}$ is a formula.*
- *Let $\overline{x} := x_1, \ldots, x_l$ be a tuple of element variables, $\overline{R} := R_1, \ldots, R_k$ be a tuple of region variables, and M a free (k, l)-ary set variable. If $\varphi(M, \overline{R}, \overline{x})$ is a formula such that M occurs only positively in φ, then $[\mathbf{tlfp}_{M,\overline{R},\overline{x}}\, \varphi](\overline{R}, \overline{x})$ is a formula.*
- *If $\varphi(x)$ is a formula and x, y, z are element variables, then $x \cdot_\varphi y = z$ is a formula.*

RegLFP(conv)-queries are defined by RegLFP(conv)-formulae without free region or set variables.

The semantics of the new rules is defined as follows. An atom $R\overline{x}$ states that the point $\overline{x}$ is contained in the region R. An atom $M\overline{R}\overline{x}$ is satisfied if the tuple $(\overline{R}, \overline{x})$ is contained in M. A formula $\exists R\varphi$ is true if there is a region $R \in \mathit{Reg}$ satisfying φ. The semantics of conv, uconv, and $\cdot_\varphi$ is defined as in the previous section.

Finally, let M be a (k, l)-ary set variable, $\overline{R} := R_1, \dots, R_k$ be a sequence of region variables, $\overline{x} := x_1, \dots, x_l$ be a sequence of element variables, and $\varphi(M, \overline{R}, \overline{x})$ be a formula positive in M. The result of the formula $\psi := [\mathbf{tlfp}_{M,\overline{R},\overline{x}}\varphi](\overline{R}, \overline{x})$ evaluated in a database $\mathfrak{B}$ is defined as the least fixed-point of the function f_φ defined as

$$\begin{aligned} f_\varphi : \mathrm{Pow}(\mathit{Reg}^k \times \mathbb{R}^l) &\longrightarrow \mathrm{Pow}(\mathit{Reg}^k \times \mathbb{R}^l) \\ M &\longmapsto \{(\overline{R}, \overline{a}) \in \mathit{Reg}^k \times \mathbb{R}^l : (\exists \overline{y}\,(\overline{R}, \overline{y}) \in M \wedge (\overline{R}, \overline{a}) \in M) \vee \\ &\qquad (\neg\exists \overline{y}(\overline{R}, \overline{y}) \in M \wedge \mathfrak{B} \models \varphi(M, \overline{R}, \overline{a}))\}. \end{aligned}$$

Clearly, the function is monotone and thus its least fixed-point exists and can be computed inductively in time polynomially in the number of regions and thus also in the size of the database.

Intuitively, we can think of the stages of the fixed-point induction as a set of tuples $\overline{R}$ of regions, where for each $\overline{R}$ there is a formula $\varphi_{\overline{R}}(\overline{x})$ attached to it defining a set of points in $\mathbb{R}^l$. Once a tuple of regions is contained in some stage of the fixed-point induction, the formula attached to it cannot be changed anymore. This is ensured by the first disjunct in the definition of f_φ.

Note that the definition of the logic does not use any features particular to arrangements. All that is needed is a partition of the underlying space $\mathbb{R}^d$ into a set of regions which respect the database relation S. Further, this set of regions should be computable in polynomial time in the size of the database. Therefore, in practical applications, a decomposition into regions that is better suited for the particular application area can be used. For instance, a CAD system might use a decomposition where the regions consist of the intersections of lines and the part of the lines in between.

We now give an example motivating the definition of the fixed-point operator.

18.3 Example. *Consider a road map with cities and the roads connecting them. Typically, such maps contain information about the distance between any two adjacent cities directly connected by a road. In this set-up, a useful decomposition of the input into regions is to have a region for each city, one for each section of a road between two cities, and regions for the other parts of the map.*

Suppose we plan to travel from one city to another and want to know the distance between these two cities. Let the constants s, t denote the regions of the source and target city and assume a formula $\mathrm{dist}(C, C', d)$*, stating that*

C and C' are regions of adjacent cities and d is the distance between both. Then the formula

$$\begin{aligned}\varphi(x) := [\mathbf{tlfp}_{M,C_1,C_2,d}\ & \mathrm{dist}(C_1,C_2,d) \ \vee\ (\exists C\,\exists d_1\,\exists d_2\ M(C_1,C,d_1)\ \wedge \\ & \mathrm{dist}(C,C_2,d_2)\ \wedge\ d = d_1 + d_2)\ ](s,t,x)\end{aligned}$$

defines the distance between the two cities. Here the real variable d in the fixed-point induction is used to sum up the distances.

We proceed by establishing an upper bound on the data complexity of the evaluation problem for *RegLFP(conv)*-formulae. Let $\psi := [\mathbf{tlfp}_{M,\overline{R},\overline{x}}\varphi]$ be a formula. Intuitively, we take a semi-naive evaluation strategy. Each tuple $\overline{R}$ of regions in the interpretation of the set variable M has a quantifier-free formula $\varphi_{\overline{R}}(\overline{x})$ attached to it, with free element variables $\overline{x}$. Once such a tuple $\overline{R}$ is added to M, the attached formula cannot be changed in the successive steps. This is enforced by the first disjunct in the definition of f_φ. By induction on the structure of the formula we get that for each tuple $\overline{R}$ of regions, the sub-formula φ can be evaluated in polynomial time. As there are only polynomially many regions and in every stage of the fixed point induction at least on tuple of regions must be added to M, it follows that the fixed point of φ can be computed in polynomial time. This establishes the following theorem.

18.4 Theorem. *RegLFP(conv) is closed and has* PTIME *data complexity.*

The theorem gives an upper bound for the evaluation complexity of *RegLFP(conv)*. We now turn towards establishing lower bounds. Precisely, we aim at showing that the logic is expressive enough to capture all polynomial time computable queries on linear constraint databases. The proof of this follows the usual line by showing that the run of a Turing-machine on a linear constraint database can be simulated by a formula of the logic, provided that the running time of the machine is polynomially bounded. In the simulation, positions on the Turing-tape and time steps will be represented by tuples of region variables. However, there are cases where there are not enough regions to represent every position on the Turing-tape used by the Turing-machine. Therefore we have to adopt the following restriction.

18.5 Definition. *A linear constraint database $\mathfrak{B}$ has the* small coordinate property *if the absolute values of the coordinates of points contained in a 0-dimensional region are bounded by $2^{O(n)}$, where n is the number of regions in the region extension of $\mathfrak{B}$.*

An example of a database violating this definition is one that contains only very few regions but which are far away from the origin $\bar{0}$. In such a case there are not enough regions to define a binary representation of coordinates of points in the database. However, usually this restriction is never a problem.

18.6 Theorem. *RegLFP(conv) captures* PTIME *on the class of linear constraint databases having the small coordinate property.*

Proof. We take the usual approach to show capturing results by coding the runs of Turing-machines. See [EF99] and [Imm98] for details. In the approach taken in finite model theory, configurations of Turing-machines are coded by sets of elements from the finite universe. A fixed-point induction is then used to build up the sequence of configurations constituting the run of the Turing-machine. As in *RegLFP(conv)* the fixed-point induction is defined over sets of regions, we use set and region variables to code configurations of Turing-machines. These variables will only range over point regions, i.e. regions of dimension 0. Clearly, this can be enforced by the formulae but to ease notation we will never do this explicitly. Instead, we agree that unless said otherwise, all region variables range over the set of point regions. As such regions only contain a single point, the lexicographical order on the coordinates of these points induces a total order on the set of point regions. Thus, this set is isomorphic to $(\{0, \ldots, n-1\}, <)$ for some n. We use this to encode configurations as common in finite model theory.

Let Q be a PTIME-query and let M be a Turing-machine computing Q. There is a fixed $k \in \mathbb{N}$, such that the running time of M can be bounded by n^k, where n is the number of point regions in the input structure. We aim at defining a *RegLFP(conv)*-formula φ_M coding the run of the machine. In the sequel, capital letters denote set variables and small letter denote region variables. Element variables are not used in the proof. The configurations of the machine are encoded as follows. Essentially, there are set variables $X_q(t_1, \ldots, t_k)$ for each $q \in Q$, stating that the machine is in state q at time $\overline{t}$, a variable $Y_a(\overline{t}, p_1, \ldots, p_k)$ for each $a \in \Sigma$ stating that at time $\overline{t}$ the position $\overline{p}$ on the Turing-tape contains an a and, finally, a variable $Z(\overline{t}, \overline{p})$ stating that at time $\overline{t}$ the head position is $\overline{p}$.

Since we model the run of the Turing-machine by a least fixed-point computation, we have to encode the configurations of the machine in one single relation only. Thus we use a $2k+2$-ary relation $C(r, a, \overline{p}, \overline{t})$ interpreted as follows. C contains all tuples $(r, a, \overline{p}, \overline{t})$ such that

- $r = 0$, a denotes a state $q \in Q$ of M and $X_q\overline{t}$,
- $r = 1$, a denotes a symbol of the alphabet and $Y_a(\overline{t}, \overline{p})$, or
- $r = 2$, $a = 0$ and $Z(\overline{t}, \overline{p})$ holds true.

Essentially, the formula φ_M consists of three parts: the first defines the start configuration, the second the run of the machine, and the third decodes the output.

An input database $\mathfrak{B} := (\mathbb{R}, <, +, S)$ is coded on the Turing-tape as follows. The relation S is encoded by storing the regions of the region extensions of $\mathfrak{B}$ and specifying, which regions are subsets of S. Any bounded region can be identified by the coordinates of its vertices. Thus the bounded part of the relation S is stored by first writing down the coordinates of the points in the point regions and then specifying for each $1 \leq i \leq d+1$ and each i-tuple of points whether their open convex hull is contained in the relation. The following figure demonstrates this.

$$\underbrace{\overbrace{|p_1^1|p_2^1|\dots|p_d^1|}^{\text{1st point}}\,|\dots\overbrace{|p_1^m|p_2^m|\dots|p_d^m|}^{\text{m-th point}}}_{\text{Point set}}\#\underbrace{|d_1^0|d_2^0|\dots|d_m^0|}_{\text{Dimension 0}}\dots\underbrace{|d_1^d|d_2^d|\dots|d_{m^{d+1}}^d|}_{\text{Dimension } d}$$

An entry p_j^i denotes the bit representation of the j-th coordinate of the i-th point. Each coordinate consumes n bits on the tape. Here, the small coordinate property is needed as otherwise it is not guaranteed that there are enough bits, i.e. regions, to store the representation. The points on the tape are ordered according to the order induced by the lexicographical ordering of the coordinates. An entry d_j^i equals 1 if the open convex hull of the j-th i-tuple of points is entirely contained in the input relation. Otherwise it is 0. The unbounded regions can be coded similarly. Instead of the vertices identifying the bounded regions, we use half lines delimiting the unbounded regions. These are represented by two points as indicated by the uconv-operator.

Using this representation we can easily define φ_{Start}. The only subtle part is the encoding of the bit representation of the point's coordinates. Given an integer a, a formula $\psi(R)$ which is satisfied by the i-th region if, and only if, the i-th bit of a is true can be defined using a least fixed-point induction enumerating all regions and tagging them with 1 if the corresponding bit is 1.

The second formula, $\varphi_{Compute}(r, a, \overline{t}, \overline{p})$, coding the run of the Turing machine, can be defined as in finite model theory. For each time step $\overline{t}$, $\varphi_{Compute}(r, a, \overline{t}, \overline{p})$ defines the configuration after the $\overline{t}$-th computation step. See [Grä03] and [EF99] for details.

We assume that the output of the Turing-machine is given in the same form as the encoding of the input. A formula decoding this output can easily be defined using an application of the **tlfp** operator to decode the bit representation of the point's coordinates and then using the conv and uconv operator to compute the interior of the region represented by a tuple of points. □

18.2 Finitary Fixed-Point Logic

The query language introduced in the previous section depends on a specific decomposition of the input database. Thus, its usability relies on the existence of a decomposition which can easily be understood by the user.

Although this is the case in some application areas, it will be a problem in others. In this section we present a way to overcome this dependency on a specific, intuitive decomposition.

We already mentioned that finiteness of a semi-linear set is first-order definable. Thus, we may use a formula finite(φ), which, given a formula φ, evaluates to true if, and only if, the set defined by φ is finite.

18.7 Definition. Finitary Fixed-Point Logic (FFP) *is defined as the extension of FO(conv) by the following fixed-point construct. If* $\overline{x} := x_1, \dots, x_k$ *and* $\overline{z} := z_1, \dots, z_l$ *are sequences of first-order variables,* R *is a* $(k+l)$*-ary second-order variable,* $\varphi(\overline{x})$ *and* $\psi(R, \overline{x}, \overline{z})$ *are formulae, such that* R *does not occur in* φ *and only positively in* ψ*, then* $[\mathbf{flfp}_{R,\overline{x}}(\varphi, \psi)](\overline{u}, \overline{v})$ *is a formula with free variables* $\{\overline{u}, \overline{v}\}$*, where* $\overline{u}, \overline{v}$ *are sequences of variables of arity* k *and* l*.*

The semantics of a formula $\chi := [\mathbf{flfp}_{R,\overline{x}}(\varphi, \psi)](\overline{u}, \overline{v})$ is defined as the least fixed-point of the function

$$\begin{aligned} f_\chi : \; & Pow(\mathbb{R}^{k+l}) \longrightarrow Pow(\mathbb{R}^{k+l}) \\ & R \longmapsto \{(\overline{x}, \overline{z}) : \; (\exists \overline{y}\ (\overline{x}, \overline{y}) \in R \land (\overline{x}, \overline{z}) \in R) \lor \\ & \qquad\qquad ((\neg \exists \overline{y}\, (\overline{x}, \overline{y}) \in R) \land \mathfrak{B} \models \begin{pmatrix} \text{finite}(\varphi) \land \varphi(\overline{x}) \land \\ \psi(R, \overline{x}, \overline{z}))) \end{pmatrix} \}, \end{aligned}$$

where $\mathfrak{B}$ is the input database. Intuitively, the formula φ serves as a guard, ensuring that the fixed-point induction runs over the finite set defined by φ only. As before, the variables $\overline{z}$ can be used to attach some information to a tuple $\overline{x}$ contained in an induction stage.

Regarding the expressive power of this language, one can easily show that the languages *RegLFP(conv)* and FFP are equivalent. The expressive power of this construction relies on the fact that an encoding of the input database by a finite set of points can be defined in *RegFO(conv)*. Using this encoding one can replace the fixed-point induction on the regions by a fixed-point induction on the finite representation.

Thus, FFP captures PTIME on the class of linear constraint databases.

18.8 Theorem. (*i*) *FFP captures* PTIME *on the class of linear constraint databases having the small coordinate property.*

(*ii*) *RegLFP(conv) and FFP have the same expressive power on the class of linear constraint databases.*

Bibliography

[AB87] V. Arvind and S. Biswas. Expressibility of first order logic with a nondeterministic inductive operator. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science*, volume 247 of *Lecture Notes in Computer Science (LNCS)*, pages 323 – 335, 1987.

[ABB96] R. Anderson, P. Beame, and E. Brisson. Parallel algorithms for arrangements. *Algorithmica*, 15:104 – 125, 1996.

[Acz77] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic*, pages 739 –782. North-Holland, 1977.

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

[AN01] A. Arnold and D. Niwiński. *Rudiments of μ-calculus.* North Holland, 2001.

[AV89] S. Abiteboul and V. Vianu. Fixpoint extensions of first-order logic and datalog-like languages. In *Proc. 4th IEEE Symp. on Logic in Computer Science (LICS)*, pages 71–79, 1989.

[AV91] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proc. of the 23rd ACM Symp. on the Theory of Computing*, 1991.

[AvBN98] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

[AVV92] S. Abiteboul, M. Vardi, and V. Vianu. Fixpoint logics, relational machines, and computational complexity. In *Proc. 7th IEEE Symp. on Structure in Complexity Theory*, pages 156 – 168, 1992.

[AVV97] S. Abiteboul, M. Vardi, and V. Vianu. Fixpoint logics, relational machines, and computational complexity. *Journal of the ACM*, 44(1):30–56, 1997. An extended abstract appeared in [AVV92].

[BDLW96] M. Benedikt, G. Dong, L. Libkin, and L. Wong. Relational expressive power of constraint query languages. In *Proc. 15th ACM Symp. on Principles of Database Systems*, pages 5–16, 1996.

[BDLW98] M. Benedikt, G. Dong, L. Libkin, and L. Wong. Relational expressive power of constraint query languages. *Journal of the ACM*, 45:1 – 34, 1998.

[BL00] M. Benedikt and L. Libkin. Expressive power: The finite case. In G.M. Kuper, L. Libkin, and J. Paredaens, editors, *Constraint Databases*, chapter 3, pages 55 – 87. Springer Verlag, 2000.

[Bra98a] J. Bradfield. The model μ-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195:133–153, 1998.

[Bra98b] J. Bradfield. Simplifying the modal mu-calculus alternation hierarchy. In *Proc. of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1373 of *Lecture Notes in Computer Science (LNCS)*, pages 39–49, 1998.

[BS01] 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, North-Holland, 2001.

[BST96] O. Belegradek, A. Stolboushkin, and M. Taitslin. On order-generic queries. Technical Report 96-01, DIMACS, 1996.

[BST97] O. Belegradek, A. Stolboushkin, and M. Taitslin. Generic queries over quasi-o-minimal domains. In *Proceedings of LFCS 97*, volume 1234 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 1997.

[BST99] O. Belegradek, A. Stolboushkin, and M. Taitslin. Extended order-generic queries. *Annals of Pure and Applied Logic*, 97(1–3):85 – 125, 1999.

[Büc60] J. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* MIT Press, 1999.

[Cod70] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13:377–387, 1970.

[Dah87] E. Dahlhaus. Skolem normal forms concerning the least fixed point. In E. Börger, editor, *Computation Theory and Logic*, volume 270 of *Lecture Notes in Computer Science (LNCS)*, pages 20–36. Springer-Verlag, 1987.

[Daw93] A. Dawar. *Feasible Computation Through Model Theory.* PhD thesis, University of Pennsylvania, 1993.

[DG02] A. Dawar and Y. Gurevich. Fixed-point logics. *Bulletin of Symbolic Logic*, 8(1):65–88, 2002.

[DGK01] A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logic. In *Proc. of the 10th Conf. on Computer Science Logic (CSL)*, volume 2142 of *Lecture Notes in Computer Science (LNCS)*, pages 277–291. Springer Verlag, 2001.

[DK02] A. Dawar and S. Kreutzer. Generalising automaticity to modal properties of finite structures. In *Proc. 22nd Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2556 of *Lecture Notes in Computer Science (LNCS)*, pages 109–120. Springer Verlag, 2002.

[DLW95] A. Dawar, S. Lindell, and S. Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation*, 119:160–175, 1995.

[Don70] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406 – 451, 1970.

[DR03] A. Dawar and D. Richerby. Fixed-point logics with nondeterministic choice. *Journal of Logic and Computation*, 13(4):503 – 530, 2003.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* EATCS Monographs on Theoretical Computer Science. Springer, 1987.

[EF99] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 2nd edition, 1999.

[EFT94] H.D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Springer, 1994.

[EJ88] A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 328–337, 1988.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings*, volume 7, 1974.

[Flu95] J. Flum. On the (infinite) model theory of fixed-point logics. Preprint, 1995.

[GdBG97] M. Gyssens, J. Van den Bussche, and D. Van Gucht. Complete geometrical query languages. In *Proc. 16th ACM Symp. on Principles of Database Systems*, pages 62–67, 1997.

[Gee01] F. Geerts. *Geometric and Algorithmic Aspects of Topological Queries to Spatial Databases.* PhD thesis, Limburgs Universitair Centrum, Diepenbeek, 2001.

[GH98] F. Gire and H.K. Hoang. An extension of fixpoint logic with a symmetry-based choice construct. *Information and Computation*, 144:40 – 65, 1998.

[GHO02] E. Grädel, C. Hirsch, and M. Otto. Back and Forth Between Guarded and Modal Logics. *ACM Transactions on Computational Logics*, 3(3):418 – 463, 2002.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and company, New York, 1979. ISBN 0-7167-1044-7.

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

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

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

[GO97] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press, 1997.

[Goo93] M. T. Goodrich. Constructing arrangements optimal in parallel. *Discrete & Computational Geometry*, 9:371 – 385, 1993.

[Grä02] E. Grädel. Guarded fixed point logic and the monadic theory of trees. *Theoretical Computer Science*, 288:129 – 152, 2002.

[Grä03] E. Grädel. Finite model theory and descriptive complexity. In *Finite Model Theory and Its Applications.* Springer-Verlag, 2003. To appear. See http://www-mgi.informatik.rwth-aachen.de/Publications/pub/graedel/Gr-FMTbook.ps.

[Gro94] M. Grohe. *The Structure of Fixed-Point Logics.* PhD thesis, Albert-Ludwigs Universität Freiburg i. Br., 1994.

[Gro96] M. Grohe. Arity hierarchies. *Annals of Pure and Applied Logic*, 82:103 – 163, 1996.

[Gro97] M. Grohe. Existential least fixed-point logic and its relatives. *Journal of Logic and Computation*, 7:205–228, 1997.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[GS97] S. Grumbach and J. Su. Finitely representable databases. *Journal of Computer and System Sciences*, 55:273–298, 1997.

[GTW02] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games – A Guide to Current Research.* Springer Verlag, 2002.

[GW99] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *Proc. 14th IEEE Symp. on Logic in Computer Science*, 1999.

[HK75a] L.A. Harrington and A.S. Kechris. On monotone first-order inductive definitions. Unpublished manuscript, 1975.

[HK75b] L.A. Harrington and A.S. Kechris. On monotone vs. non-monotone induction. Unpublished manuscript, 1975.

[HM74] L.A. Harrington and Y.N. Moschovakis. On positive induction vs. non-monotone induction. Unpublished manuscript, 1974.

[Hod97] W. Hodges. *A shorter model theory.* Cambridge University Press, 1997.

[Imh96a] H. Imhof. Computational aspects of arity hierarchies. In *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL*, volume 1258 of *Lecture Notes in Computer Science (LNCS)*, pages 211–225, 1996.

[Imh96b] H. Imhof. *Fixed-Point Logics and Generalized Quantifiers in Descriptive Complexity.* PhD thesis, Albert-Ludwig Universität, Freiburg i. Br., Germany, 1996.

[Imm81] N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22:384 – 406, 1981.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986. Extended abstract in Proc. 14th ACM Symp. on Theory of Computing, pages 147-152, 1982.

[Imm87a] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760 – 778, 1987.

[Imm87b] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760 – 778, 1987. An extended abstract appeared in *Proc. of the 15th Annual ACM Symp. on Theory of Computing (STOC), pages 347-354, 1983.*

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17(5):935 – 938, 1988. Abstract appeared in Third Structure in Complexity Theory Symp. (1988), 112-115.

[Imm98] N. Immerman. *Descriptive complexity.* Graduate Texts in Computer Science. Springer, 1998.

[Jec97] T. J. Jech. *Set theory.* Perspectives in Mathematical Logic. Springer Verlag, Berlin, 2 edition, 1997.

[Jur98] M. Jurdzinski. Deciding the winner in parity games is in UP $\cap$ Co-UP. *Information Processing Letters*, 68:119–124, 1998.

[JW96] 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*, number 1119 in Lecture Notes in Computer Science, pages 263–277. Springer-Verlag, 1996.

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

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

[Kle55] S. C. Kleene. Arithmetical predicates and function quantifiers. *Transactions of the American Mathematical Society*, 79:312 – 340, 1955.

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

[Kol91] P. Kolaitis. The expressive power of stratified logic programs. *Information and Computation*, 90:50–66, 1991.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Koz88] D. Kozen. A finite model theorem for the propositional μ-calculus. *Studia Logica*, 47:233 – 241, 1988.

[KP84] D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In E. Clarke and D. Kozen, editors, *Proc. Workshop on Logics of Programs*, volume 164 of *Lecture Notes in Computer Science (LNCS)*, pages 313 – 326. Springer Verlag, 1984.

[KPSV96] B. Kuijpers, J. Paredaens, M. Smits, and J. Van den Bussche. Termination properties of spatial datalog programs. In *Logic in Databases*, number 1154 in LNCS, pages 101 – 116, 1996.

[Kre99a] S. Kreutzer. Descriptive complexity theory for constraint databases, 1999. Diplomarbeit an der RWTH Aachen.

[Kre99b] S. Kreutzer. *Descriptive Complexity Theory for Constraint Databases.* Diploma thesis, RWTH Aachen, 1999.

[Kre00] S. Kreutzer. Fixed-point query languages for linear constraint databases. In *Proceedings of the 19th ACM Symp. on Principles of Database Systems (PODS), 2000*, pages 116–125. ACM press, 2000.

[Kre01a] S. Kreutzer. Operational semantics for fixed-point logics on constraint databases. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, volume 2250 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 470 – 484. Springer, 2001.

[Kre01b] S. Kreutzer. Query languages for constraint databases: First-order logic, fixed-points, and convex hulls. In *Proceedings of the 8th International Conference on Database Theory (ICDT)*, number 1973 in Lecture Notes in Computer Science (LNCS), pages 248–262. Springer, 2001.

[Kre02a] S. Kreutzer. Expressive equivalence of least and inflationary fixed-point logic. In *Proc. of the 17th Symp. on Logic in Computer Science (LICS)*, pages 403 – 413, 2002.

[Kre02b] S. Kreutzer. Partial fixed-point logic on infinite structures. In *Annual Conference of the European Association for Computer*

Science Logic (CSL), volume 2471 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2002.

[KV00] B. Kuijpers and D. Van Gucht. Genericity in spatial databases. In G.M. Kuper, L.Libkin, and J.Paredans, editors, *Constraint Databases*, chapter 12, pages 293 – 303. Springer Verlag, 2000.

[KVW00] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47:312–360, 2000.

[LS01] C. Lautemann and N. Schweikardt. An Ehrenfeucht-Fraïssé approach to collapse results for first-order queries over embedded databases. In *18th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2010 of *Lecture Notes in Computer Science (LNCS)*, pages 455 – 466. Springer Verlag, 2001.

[Mos74a] Y.N. Moschovakis. *Elementary Induction on Abstract Structures.* North Holland, 1974. ISBN 0 7204 2280 9.

[Mos74b] Y.N. Moschovakis. On non-monotone inductive definability. *Fundamentae Mathematica*, 82:39–83, 1974.

[Mos94] Y.N. Moschovakis. *Notes on Set Theory.* Springer Verlag, 1994. ISBN 3-540-94180-0.

[Niw86] D. Niwiński. On fixed point clones. In *Proc. 13th International Colloq. on Automata, Languages, and Programming (ICALP)*, number 226 in LNCS, pages 464–473, 1986.

[Ott99] M. Otto. Bisimulation-invariant Ptime and higher-dimensional mu-calculus. *Theoretical Computer Science*, 224:237–265, 1999.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[PVV94] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 279–288, 1994.

[Ros97] E. Rosen. Modal logic over finite structures. *Journal of Logic, Language and Information*, 6:427–439, 1997.

[SB96] J. Shallit and Y. Breitbart. Automaticity I: Properties of a measure of descriptional complexity. *Journal of Computer and System Sciences*, 53:10–25, 1996.

[Sch01] N. Schweikardt. The natural order-generic collapse for ω-representable databases over the rational and the real ordered group. In *10th Annual Conference of the Eurpean Association for Computer Science Logic (CSL)*, volume 2142 of *Lecture Notes in Computer Science (LNCS)*, pages 130 – 144. Springer Verlag, 2001.

[Sch02] N. Schweickardt. *On the Expressive Poer of First-Order Logic with Built-In Predicates.* PhD thesis, Johannes Gutenberg - Universität Mainz, 2002.

[SE89] R. Streett and A. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81:249–264, 1989.

[SM73] L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *Proc. of the 5th ACM Symp. on Theory of Computing*, pages 1 – 9, 1973.

[Tho91] Wolfgang Thomas. On logics, tilings, and automata. In J. Leach et al., editor, *Automata, Languages, and Programming*, Lecture Notes in Computer Science Nr. 510, pages 441–453. Springer-Verlag, 1991.

[Tho94] W. Thomas. Finite-state recognizability and logic: from words to graphs. In *13th World Computer Congress 94*, volume 1, pages 499–506. Elsevier Science, 1994.

[TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata with an application to a decision problem of second order logic. *Mathematical System Theory*, 2:57 – 82, 1968.

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

[Var82] M. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on the Theory of Computing*, pages 137–146, 1982.

[vB76] J. van Benthem. *Modal correspondence theory.* PhD thesis, University of Amsterdam, 1976.

[vB83] J. van Benthem. *Modal Logic and Classical Logic.* Bibliopolis, Napoli, 1983.

[VGG98] L. Vandeurzen, M. Gyssens, and D. Van Gucht. An expressive language for linear spatial database queries. In *Proc. 17th ACM Symp. on Principles of Database Systems*, pages 109–118, 1998.

[Zie95] Günter M. Ziegler. *Lectures on Polytopes.* Number 152 in Graduate texts in mathematics. Springer, 1995.

Symbols

Index