Model-Checking Finite Structures

Lecture Notes

Sylvain Schmitz

Université Paris Cité, CNRS, IRIF, France

2020 *Mathematics Subject Classification.* Primary Logic

Key words and phrases. logic

Contents

Overview

These lecture notes are dedicated to the course *Model-Checking Finite Structures* that takes place in 2025–2028 as part of the *Master in Mathematical Logic and Foundations of Computer Science* at Université Paris Cité.

Rather than a general course on topics in finite model theory (Libkin, 2004; Toruńczyk, 2022), the course focuses on one single topic: the *model-checking problem*. The goal here is to design algorithms that, given as input a first-order sentence φ and a finite relational structure $\mathfrak{A}$, answer whether $\mathfrak{A} \models \varphi$.

This particular focus is mainly motivated by its applications for evaluating queries in database systems (Abiteboul, Hull, and Vianu, 1995; Arenas et al., 2022). However, the course does not enter the details of this application; instead, this particular lens is an excuse to touch upon a number of topics related to finite model theory and complexity theory, notably algorithmic meta-theorems and in particular Courcelle's Theorem, but also circuit complexity, parameterised complexity, monadic second-order logic, tree automata and languages, etc., as presented for instance in chapters 6–7 of (Libkin, 2004), chapters 10–12 of (Flum and Grohe, 2006), chapters 14 and 18–23 of Arenas et al. (2022), Part III of (Downey and Fellows, 2013), the whole of (Kreutzer, 2009), or Chapter 3 of (Comon et al., 2008).

Main References. The main references used in the preparation of these lecture notes are the following.

Arenas, Marcelo, Pablo Barceló, Leonid Libkin, Wim Martens, and Andreas Pieris (2022). *Database Theory. Querying Data.* Preliminary version. (cit. on pp. xi, 2, 3, 13, 15, 28, 36, 45, 59, 97)

Comon, Hubert, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi (2008). *Tree Automata. Techniques and Applications.* (cit. on p. xi)

Courcelle, Bruno and Joost Engelfriet (2012). *Graph Structure and Monadic Second Order Logic. A Language-Theoretic Approach.* Encyclopedia of mathematics and its applications 138. Cambridge University Press. (cit. on p. 11)

Flum, Jörg and Martin Grohe (2006). *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. Springer. (cit. on pp. xi, 39, 40, 45, 46, 59, 89, 93)

Libkin, Leonid (2004). *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. Springer. (cit. on pp. xi, 11, 13, 45, 46)

CHAPTER 1

Preliminaries

1.1. Signatures and Structures

1.1.1. Signatures. A *signature* (aka a *language* or a *vocabulary*) is a set $\sigma = \mathcal{P} \uplus \mathcal{F}$ composed of two disjoint sets $\mathcal{P}$ of *relation symbols* (aka *predicate symbols*) and $\mathcal{F}$of *function symbols*. Each symbol in σ comes with an *arity* in $\mathbb{N}$, which we will denote through a function $\mathsf{ar}\colon \sigma \to \mathbb{N}$ or through a superscript between parentheses, so that for instance $R^{(3)}$ denotes a relation symbol of arity $\mathsf{ar}(R) = 3$. We will restrict the arity of relation symbols to be positive: $\mathsf{ar}(R) > 0$ for all $R \in \sigma$; in other words, there are no proposition symbols.

As usual in a finite model-theoretic setting, we are mostly interested in signatures σ that are *finite* and *relational*. The latter means that there are no function symbols in σ (i.e., $\mathcal{F} = \emptyset$).[1] We write $\mathsf{ar}(\sigma)) \stackrel{\text{def}}{=} \max_{R\in\sigma} \mathsf{ar}(R)$ for the maximal arity of a symbol in a finite relational signature σ.

A counterpart to relational signatures are *algebraic signatures*, where $\mathcal{P} = \emptyset$, which will be used in **??**.

Example 1.1 (Database schemas)**.** A great source of examples and motivations for finite model theory, and the model-checking problem in particular, stems from relational databases.
A notion that closely matches that of a signature in this context is the one of a *database schema*, which in SQL is declared through **CREATE TABLE** directives. If we abstract away from all the features of actual SQL schemas (e.g., types, keys, foreign keys, etc.), a database schema is essentially a list of relation symbols $R^{(r)}$, and for each such symbol, an r-tuple of attribute names. For instance, consider the SQL declarations

```
CREATE TABLE City (
  cid     VARCHAR PRIMARY KEY,
  cname   VARCHAR NOT NULL,
  country VARCHAR NOT NULL
);
CREATE TABLE Person (
  pid     INTEGER PRIMARY KEY,
  name    VARCHAR NOT NULL,
```

[1] When convenient we might allow some additional *constant symbols* (i.e., function symbols with arity zero) but these can typically be encoded through unary relation symbols.

```
    cid      VARCHAR REFERENCES City
  );
  CREATE TABLE Profession (
    pid      INTEGER NOT NULL REFERENCES Person,
    prname   VARCHAR NOT NULL
  );
```

When ignoring all the information about types, keys, etc., these table declarations correspond to the following 'named' database schema (taken from Arenas et al., 2022, Example 3.2)

- `City[cid,cname,country]` to store information about cities, each with an identifier (`cid`), a name (`cid`), and a country code (`country`),
- `Person[pid,pname,cid]` to store identifiers of persons (`pid`), their names (`pname`), and their city of birth (`cid`), and
- `Profession[pid,prname]` to store the professions of people by relating their identifiers (`pid`) with profession names (`pname`).

Finally, when also dropping the names of the attributes, we get a relational signature $\sigma = \{\texttt{Person}^{(3)}, \texttt{Profession}^{(2)}, \texttt{City}^{(3)}\}$. (As we will see in Example 1.7, this signature will be enriched with additional constant symbols.)

1.1.2. Structures. A *structure* $\mathfrak{A} = (A, (R^{\mathfrak{A}})_{R\in\mathcal{P}}, (f^{\mathfrak{A}})_{f\in\mathcal{F}})$ over a signature $\sigma = \mathcal{P} \uplus \mathcal{F}$ is a tuple that contains a non-empty set A called its *domain*, along with

- a relation $R^{\mathfrak{A}} \subseteq A^{\mathsf{ar}(R)}$ for every relation symbol R of σ and
- a function $f^{\mathfrak{A}} \colon A^{\mathsf{ar}(f)} \to A$ for every function symbol f of σ; for a constant function symbol c, it is convenient to think of $c^{\mathfrak{A}}$ as an element of A.

A structure $\mathfrak{A}$ is *finite* if its domain $|A|$ is finite; as we will exclusively work with finite signatures, a finite structure will indeed have a finite representation. A *finite relational structure* is a finite structure over some finite relational signature.

Example 1.2 (Relational structure)**.** Consider the relational signature $\sigma \stackrel{\text{def}}{=} \{R^{(2)}, S^{(2)}\}$ with two binary relational symbols. Figure 1.1 depicts a finite structure $\mathfrak{A}$ over σ with domain $A \stackrel{\text{def}}{=} \{a, b, c\}$, $R^{\mathfrak{A}} \stackrel{\text{def}}{=} \{(a, a), (a, b)\}$, and $S^{\mathfrak{A}} \stackrel{\text{def}}{=} \{(b, b), (b, c)\}$, where R edges are in solid green and S edges in dashed grey.

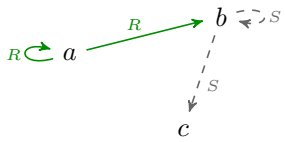


FIGURE 1.1. A finite relational structure.

Example 1.3 (Relational databases)**.** Just like database schemas are the counterparts to relational signatures, *database instances* are the counterparts of finite relational structures. To illustrate the notion, let us first consider how the structure of Example 1.2 could be arranged as a database, i.e., a set of 'tables' listing all the tuples in each relation:

R	
a	a
a	b

S	
b	b
b	c

As mentioned in Example 1.1, actual databases give names to their columns—their attributes—, as in the following example (ibid., Figure 3.1)

City		
cid	cname	country
MPH	Memphis	USA
DLT	Duluth	USA
ST	Stone Town	TZA

Person		
pid	pname	cid
1	Aretha	MPH
2	Billie	
3	Bob	DLT
4	Freddie	ST

Profession	
pid	prname
1	singer
1	songwriter
1	actor
2	singer
3	singer
3	songwriter
3	author
4	singer
4	songwriter

These tables can be seen as a finite structure $\mathfrak{A}$ over the relational signature $\sigma = \{\texttt{Person}^{(3)}, \texttt{Profession}^{(2)}, \texttt{City}^{(3)}\}$ from Example 1.1. We first fix an infinite *data domain* $\mathbb{D}$ that will contain all the data values we might ever wish to store. Then

- we define the domain of the structure as the subset $A \stackrel{\text{def}}{=} \{1, 2, \dots, \text{MPH}, \text{DLT}, \text{ST}, \dots\} \subseteq \mathbb{D}$ of the data domain with one element per different data value in the database, and
- we set up the relations between these elements, so that for instance $(4, \text{Freddie}, \text{ST}) \in \texttt{Person}^{\mathfrak{A}}$.

There is however a value missing for the `cid` attribute of 'Billie' in our database. In SQL, this is called a **NULL**, and we could for instance assume that we have an infinity of 'null' symbols at our disposal in $\mathbb{D}$ that we may treat as data values.

1.1.2.1. *Homomorphisms.* A *homomorphism* between two structures $\mathfrak{A}$ and $\mathfrak{B}$ over the same signature σ is a function $h\colon A \to B$ that preserves all the functions and relations: for all $m \in \mathbb{N}$ and $a_1, \dots, a_m \in A$

- $(a_1, \dots, a_m) \in R^{\mathfrak{A}}$ implies $(h(a_1), \dots, h(a_m)) \in R^{\mathfrak{B}}$ for each relation symbol $R^{(m)} \in \sigma$, and
- $h\big(f^{\mathfrak{A}}(a_1, \dots, a_m)\big) = f^{\mathfrak{B}}(h(a_1), \dots, h(a_m))$ for each function symbol $f^{(m)} \in \sigma$.

A bijective homomorphism is called an *isomorphism* if its inverse is also a homomorphism. An injective homomorphism is called an *embedding* if it is *strong*, which requires that for all $m \in \mathbb{N}$ and $a_1, \dots, a_m \in A$, $(h(a_1), \dots, h(a_{\mathsf{ar}(R)})) \in R^{\mathfrak{B}}$ implies $(a_1, \dots, a_{\mathsf{ar}(R)}) \in R^{\mathfrak{A}}$ for each relation symbol $R^{(m)} \in \sigma$. If $\mathfrak{A}$ embeds into $\mathfrak{B}$ through h, then $\mathfrak{A}$ is isomorphic to the *substructure* of $\mathfrak{B}$ induced by $h(A)$.

We shall write

$\mathfrak{A} \to \mathfrak{B}$: if there exists a *homomorphism* between $\mathfrak{A}$ and $\mathfrak{B}$,

$\mathfrak{A} \hookrightarrow \mathfrak{B}$: if there exists an *injective homomorphism* between $\mathfrak{A}$ and $\mathfrak{B}$,
$\mathfrak{A} \subseteq \mathfrak{B}$: if there exists an *embedding* between $\mathfrak{A}$ and $\mathfrak{B}$,
$\mathfrak{A} \twoheadrightarrow \mathfrak{B}$: if there exists a *surjective homomorphism* between $\mathfrak{A}$ and $\mathfrak{B}$, and
$\mathfrak{A} \cong \mathfrak{B}$: if there exists an *isomorphism* between $\mathfrak{A}$ and $\mathfrak{B}$.

Example 1.4 (Homomorphisms)**.** TODO

1.1.2.2. *Classes of Structures.* A *class of structures* is a class (in the set-theoretic sense) $\mathscr{C}$ of structures, which is closed under isomorphisms. We write

- Struct for the class of all structures over all finite relational signatures, and
- Fin for the class of all finite relational structures.

For a relational signature σ and a class of structures $\mathscr{C}$, we write $\mathscr{C}[\sigma]$ for the class of all structures over σ within $\mathscr{C}$; hence for instance $\mathsf{Fin}[\sigma]$ is the class of all finite structures over σ, and $\mathsf{Fin} = \bigcup_{\sigma \text{ finite relational}} \mathsf{Fin}[\sigma]$.

For instance, we define Graph as the class of all finite, simple, non-empty graphs over the signature of graphs $\sigma \stackrel{\text{def}}{=} \{E^{(2)}\}$. Hence $\mathsf{Graph} \subseteq \mathsf{Fin}[E^{(2)}] \subseteq \mathsf{Fin}$; graphs are precisely the finite structures $\mathfrak{G}$ over the signature $\{E^{(2)}\}$ such that $E^{\mathfrak{G}}$ an irreflexive and symmetric relation. We will use standard notations for graphs $G = (V, E)$, for instance $\{v, v\}$ for edges in E, $V(G) \stackrel{\text{def}}{=} V$ and $E(G) \stackrel{\text{def}}{=} E$ for its sets of vertices and edges.

Example 1.5 (Graphs)**.** The graph depicted in Figure 1.2a is a structure (V, E) with domain $V \stackrel{\text{def}}{=} \{a, b, c\}$ and relation $E \stackrel{\text{def}}{=} \{(a, b), (b, a), (a, c), (c, a)\}$. This is indeed a *simple graph*: there are no self-loops and the edge relation is symmetric, in which case we will rather represent it as in Figure 1.2b.

(A) Depicted as a relational structure.

(B) Depicted as a graph.

FIGURE 1.2. A simple finite graph.

1.2. First-Order Logic

1.2.1. Syntax. Let us fix an infinite countable set $\mathcal{X}$ of *first-order variables*. The sets of *first-order terms* and *first-order formulæ* over a signature σ are defined through the abstract syntax

$$t ::= x \mid f(t_1, \ldots, t_{\mathsf{ar}(f)}) \qquad \text{(terms)}$$
$$\varphi ::= R(t_1, \ldots, t_{\mathsf{ar}(R)}) \mid t_1 = t_2 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \qquad \text{(formulæ)}$$

where x ranges over X, f over $\mathcal{F}$, R over $\mathcal{P}$, and the t_i's over terms; thus in the case of relational signatures, terms are either variables or constant symbols. We write respectively $\mathsf{T}(\mathcal{F},\mathcal{X})$ and $\mathsf{FO}[\sigma]$ for the sets of terms and of formulæ over σ and $\mathcal{X}$. We denote by FO the set of all formulæ over all the finite relational signatures σ, so that $\mathsf{FO} = \bigcup_{\sigma \text{ finite relational}} \mathsf{FO}[\sigma]$.

We write $\bar{x} \subseteq \mathcal{X}$ to denote a set of variables. We assume X itself to have some fixed linear ordering, so that if $\bar{x}$ is finite it can also be construed as a tuple in $\mathcal{X}^{|\bar{x}|}$ relative to that ordering.

1.2.1.1. *Extended Syntax.* We will routinely use the usual syntactic sugar:

$$\top \stackrel{\text{def}}{=} x = x \text{ for some } x \in \mathcal{X} \qquad \text{(true)}$$

$$\bot \stackrel{\text{def}}{=} \neg\top \qquad \text{(false)}$$

$$\varphi \vee \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \wedge \neg\psi) \qquad \text{(disjunction)}$$

$$\varphi \to \psi \stackrel{\text{def}}{=} \neg\varphi \vee \psi \qquad \text{(implication)}$$

$$\varphi \leftrightarrow \psi \stackrel{\text{def}}{=} (\varphi \to \psi) \wedge (\psi \to \varphi) \qquad \text{(equivalence)}$$

$$t_1 \neq t_2 \stackrel{\text{def}}{=} \neg(t_1 = t_2) \qquad \text{(disequality)}$$

$$\forall x.\varphi \stackrel{\text{def}}{=} \neg(\exists x.\neg\varphi) \qquad \text{(universal quantification)}$$

$$\exists \bar{x} \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_k \text{ if } \bar{x} = \{x_1,\dots,x_k\} \qquad \text{(existential tuple quantification)}$$

$$\forall \bar{x} \stackrel{\text{def}}{=} \forall x_1 \cdots \forall x_k \text{ if } \bar{x} = \{x_1,\dots,x_k\} \qquad \text{(universal tuple quantification)}$$

$$\bigvee_{i\in I} \varphi_i \stackrel{\text{def}}{=} \varphi_{i_1} \vee \cdots \vee \varphi_{i_n} \text{ if } I = \{i_1,\dots,i_n\} \qquad \text{(finite disjunction)}$$

$$\bigwedge_{i\in I} \varphi_i \stackrel{\text{def}}{=} \varphi_{i_1} \wedge \cdots \wedge \varphi_{i_n} \text{ if } I = \{i_1,\dots,i_n\} \qquad \text{(finite conjunction)}$$

1.2.1.2. *Free and Bound Variables.* Intuitively, an occurrence of a variable x in a formula φ is *bound* if, going up in the syntax tree of φ, we meet a quantifier $\exists x$ (or, using the extended syntax, $\forall x$, $\exists \bar{x}$, or $\forall \bar{x}$ with $x \in \bar{x}$); if not, it is *free*. A variable is bound if all its occurrences are bound; it is free otherwise. Formally, the set $\mathsf{free}(\varphi)$ of free variables in φ is defined by induction by

$$\mathsf{free}(x) \stackrel{\text{def}}{=} \{x\} \qquad \mathsf{free}(t_1 = t_2) \stackrel{\text{def}}{=} \mathsf{free}(t_1) \cup \mathsf{free}(t_2)$$

$$\mathsf{free}(f(t_1,\dots,t_{\mathsf{ar}(f)})) \stackrel{\text{def}}{=} \bigcup_{1\le i\le \mathsf{ar}(f)} \mathsf{free}(t_i) \qquad \mathsf{free}(R(t_1,\dots,t_{\mathsf{ar}(R)})) \stackrel{\text{def}}{=} \bigcup_{1\le i\le \mathsf{ar}(R)} \mathsf{free}(t_i)$$

$$\mathsf{free}(\neg\varphi) \stackrel{\text{def}}{=} \mathsf{free}(\varphi) \qquad \mathsf{free}(\varphi \wedge \psi) \stackrel{\text{def}}{=} \mathsf{free}(\varphi) \cup \mathsf{free}(\psi)$$

$$\mathsf{free}(\exists x.\varphi) \stackrel{\text{def}}{=} \mathsf{free}(\varphi) \setminus \{x\}$$

For a set $S \subseteq \mathsf{FO}$ of formulæ, we write $\mathsf{free}(S) \stackrel{\text{def}}{=} \bigcup_{\varphi\in S} \mathsf{free}(\varphi)$ for its set of free variables.

We take $\varphi(\bar{x})$ to mean that φ is a first-order formula with $\mathsf{free}(\varphi) = \bar{x}$. A term without free variables is called *ground* and we denote the set of all ground terms over $\mathcal{F}$ by $T(\mathcal{F})$ (aka the *free algebra* over $\mathcal{F}$). A ground term necessarily has constant symbols on all the leaves of its syntax tree. A formula without free variables is called a *sentence*.

If $\varphi(\bar{x})$ is a formula and $\bar{t}$ is a tuple of terms in $(T(\mathcal{F},\mathcal{X}))^{\bar{x}}$, we write $\varphi(\bar{t})$ to denote the result of the substitution of each free occurrence within φ of a variable in $\bar{x}$ by its corresponding term in $\bar{t}$.

1.2.2. Semantics. Consider a signature $\sigma = \mathcal{P} \uplus \mathcal{F}$ and a structure $\mathfrak{A}$ over σ. A (first-order) *valuation* in $\mathfrak{A}$ of a set of variables $\bar{x} \subseteq \mathcal{X}$ is a function $\bar{a} \in A^{\bar{x}}$; using the underlying linear ordering over $\mathcal{X}$, when $\bar{x}$ is finite we can also see a valuation as a tuple in $A^{|\bar{x}|}$ of elements from A. For $b \in A$ and $y \in \mathcal{X}$, we write $\bar{a}[b/y]$ for the valuation in $A^{\bar{x}\cup\{y\}}$ such that $(\bar{a}[b/y])(y) = b$ and $(\bar{a}[b/y])(x) = \bar{a}(x)$ for all $x \neq y$ in $\bar{x}$.

The semantics $[\![t]\!]^{\mathfrak{A}}_{\bar{a}}$ of a term $t \in T(\mathcal{F},\mathcal{X})$ with $\mathsf{free}(t) \subseteq \bar{x}$ in such a valuation $\bar{a} \in A^{\bar{x}}$ in the structure $\mathfrak{A}$ is the element in A defined inductively by interpreting the term in $\mathfrak{A}$:

$$[\![x]\!]^{\mathfrak{A}}_{\bar{a}} \stackrel{\text{def}}{=} \bar{a}(x)\ , \qquad [\![f(t_1,\dots,t_{\mathsf{ar}(f)})]\!]^{\mathfrak{A}}_{\bar{a}} \stackrel{\text{def}}{=} f^{\mathfrak{A}}\big([\![t_1]\!]^{\mathfrak{A}}_{\bar{a}},\dots,[\![t_{\mathsf{ar}(f)}]\!]^{\mathfrak{A}}_{\bar{a}}\big)\ .$$

We say that a structure $\mathfrak{A}$ and a valuation $\bar{a} \in A^{\bar{x}}$ *satisfy* a first-order formula $\varphi \in \mathsf{FO}[\sigma]$ with free variables $\mathsf{free}(\varphi) \subseteq \bar{x}$, denoted $\mathfrak{A}, \bar{a} \models \varphi$, in the following inductive cases

$$\begin{aligned}
&\mathfrak{A}, \bar{a} \models R(t_1,\dots,t_{\mathsf{ar}(R)}) && \text{if } ([\![t_1]\!]^{\mathfrak{A}}_{\bar{a}},\dots,[\![t_{\mathsf{ar}(f)}]\!]^{\mathfrak{A}}_{\bar{a}}) \in R^{\mathfrak{A}}\ ,\\
&\mathfrak{A}, \bar{a} \models t_1 = t_2 && \text{if } [\![t_1]\!]^{\mathfrak{A}}_{\bar{a}} = [\![t_1]\!]^{\mathfrak{A}}_{\bar{a}}\ ,\\
&\mathfrak{A}, \bar{a} \models \neg\varphi && \text{if } \mathfrak{A}, \bar{a} \not\models \neg\varphi\ ,\\
&\mathfrak{A}, \bar{a} \models \varphi \wedge \psi && \text{if } \mathfrak{A}, \bar{a} \models \varphi \text{ and } \mathfrak{A}, \bar{a} \models \psi\ ,\\
&\mathfrak{A}, \bar{a} \models \exists x.\varphi && \text{if } \exists b \in A \text{ such that } \mathfrak{A}, \bar{a}[b/x] \models \varphi\ .
\end{aligned}$$

This definition extends to a set $S \subseteq \mathsf{FO}[\sigma]$ with $\mathsf{free}(S) \subseteq \bar{x}$ by saying that $\mathfrak{A}, \bar{a} \models S$ if $\mathfrak{A}, \bar{a} \models \varphi$ for all $\varphi \in S$.

The more usual notation in model theory textbooks is to write $\mathfrak{A} \models \varphi(\bar{a})$ for a formula $\varphi(\bar{x})$ and a valuation $\bar{a} \in A^{\bar{x}}$ if $\mathfrak{A}, \bar{a} \models \varphi$ in the above definition.[2] If φ is a sentence and $\mathfrak{A}, \varepsilon \models \varphi$ for the empty valuation ε, we write more simply $\mathfrak{A} \models \varphi$ and call $\mathfrak{A}$ a *model* of φ. This extends to a set $T \subseteq \mathsf{FO}[\sigma]$ of sentences (aka a *theory*), by writing $\mathfrak{A} \models T$ if $\mathfrak{A} \models \varphi$ for all $\varphi \in T$, in which case we also say that $\mathfrak{A}$ is a *model* of T.

Example 1.6 (Evaluation of a formula)**.** Consider the formula $\varphi(y,z) \stackrel{\text{def}}{=} \exists x.R(x,y) \wedge \neg S(y,z)$ and the finite relational structure $\mathfrak{A}$ of Example 1.2. We can see that $\mathfrak{A} \models \varphi(b,a)$ by the inductive evaluation depicted in Figure 1.3.

1.2.2.1. *Evaluating Queries.* The *evaluation* of a formula $\varphi(\bar{x})$ on a structure $\mathfrak{A}$ is the set

$$\varphi(\mathfrak{A}) \stackrel{\text{def}}{=} \{\bar{a} \in A^{\bar{x}} \mid \mathfrak{A} \models \varphi(\bar{a})\}\ .$$

If φ is a sentence, then $\varphi(\mathfrak{A})$ is either the empty set $\emptyset$ or the empty valuation ε, and can be treated as the Boolean value false or true.

Example 1.7 (SQL Queries)**.** In a database context, a formula is called a *query* and a sentence a *Boolean query*. Let us illustrate this with a few SQL queries and attempt to translate them as first-order formulæ.

[2] Note the difference: if $\mathfrak{A}, \bar{a} \models \varphi$ then $\bar{a} \in A^{\bar{x}}$ for some $\bar{x} \supseteq \mathsf{free}(\varphi)$, whereas if $\mathfrak{A} \models \varphi(\bar{a})$ then $\bar{a} \in A^{\mathsf{free}(\varphi)}$.

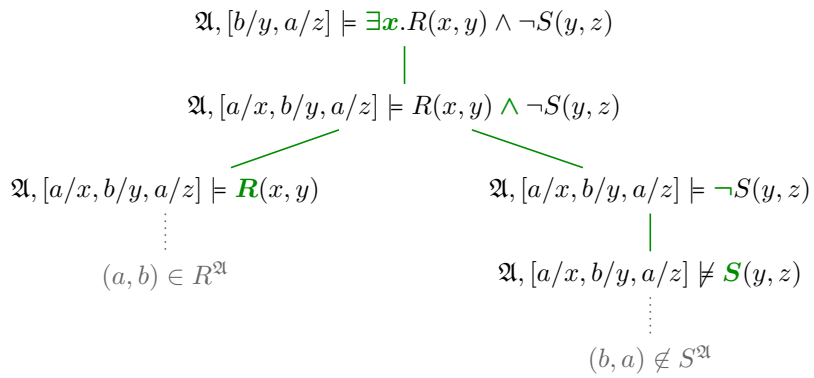


FIGURE 1.3. The evaluation of the formula φ of Example 1.6 on the finite relational structure of Example 1.2 with valuation $[b/y, a/z]$. The node of the syntax tree of φ we are working on at each inductive step is highlighted in green.

Consider the database of Example 1.3. One might ask for all the personal identifiers and names of people who have two different professions with a SQL query

```
SELECT DISTINCT Person.pid, Person.name
  FROM Profession Pr1, Profession Pr2, Person
 WHERE Pr1.pid = Person.pid AND Pr2.pid = Person.pid
       AND Pr1.prname <> Pr2.prname;
```

The answer for this query would be the pairs $(1, \text{Aretha})$, $(3, \text{Bob})$, and $(4, \text{Freddie})$. This can be written as a first-order formula with two free variables

$$\begin{aligned} \varphi_{1.1}(\mathit{pid}, \mathit{pname}) \stackrel{\text{def}}{=} \exists \mathit{cid} \exists \mathit{prname}_1 \exists \mathit{prname}_2 . \text{Person}(\mathit{pid}, \mathit{pname}, \mathit{cid}) \\ \wedge \text{Profession}(\mathit{pid}, \mathit{prname}_1) \\ \wedge \text{Profession}(\mathit{pid}, \mathit{prname}_2) \\ \wedge \mathit{prname}_1 \neq \mathit{prname}_2 \ . \end{aligned} \tag{1.1}$$

Then the evaluation of this formula on the relational structure $\mathfrak{A}$ defined in Example 1.3 for the database does indeed yield the set

$$\varphi_{1.1}(\mathfrak{A}) = \{(1, \text{Aretha}), (3, \text{Bob}), (4, \text{Freddie})\} \ .$$

As another example, we can find the professions that every person in the database exerts with the SQL query

```
SELECT DISTINCT Pr1.prname
  FROM Profession Pr1
 WHERE NOT EXISTS
  (SELECT P1.name
     FROM Person P1
    WHERE NOT EXISTS
     (SELECT P2.pid
```

```
        FROM Person P2 NATURAL JOIN Profession Pr2
       WHERE P1.pid = P2.pid AND Pr2.prname = Pr1.prname));
```

This can be written as a first-order formula with one free variable

$$\varphi_{1.2}(\mathit{prname}) \stackrel{\text{def}}{=} \forall \mathit{pid} \forall \mathit{name} \forall \mathit{cid}.\mathrm{Person}(\mathit{pid}, \mathit{name}, \mathit{cid}) \\ \rightarrow \mathrm{Profession}(\mathit{pid}, \mathit{prname}) \ , \tag{1.2}$$

whose evaluation the database yields the set

$$\varphi_{1.2}(\mathfrak{A}) = \{\text{singer}\} \ .$$

As you can see, it can be rather cumbersome to express negations in SQL; indeed, database theory often focuses on queries without negations, see Chapter 4.

Finally, consider asking for the names of the people in the database who are actors or percussionists, as in the SQL query

```
SELECT DISTINCT name
  FROM Person NATURAL JOIN Profession
 WHERE prname = 'actor' OR prname = 'percussionist';
```

Now, this is where the treatment of database queries requires a bit of fiddling with the usual definitions of first-order logic over finite relational structures: we need to have access to two constant symbols $\text{actor}^{(0)}$ and $\text{percussionist}^{(0)}$, in which case the SQL query can be expressed by the formula

$$\varphi_{1.3}(\mathit{name}) \stackrel{\text{def}}{=} \exists \mathit{pid} \exists \mathit{cid}.\mathrm{Person}(\mathit{pid}, \mathit{name}, \mathit{cid}) \\ \wedge \big(\mathrm{Profession}(\mathit{pid}, \text{actor}) \\ \vee \mathrm{Profession}(\mathit{pid}, \text{percussionist})\big) \ . \tag{1.3}$$

In order to be able to write such formulæ, we actually work with an infinite signature $\sigma \cup \mathbb{D}$, where the elements of $\mathbb{D}$ are treated as constant symbols. However, not all the elements of our infinite set $\mathbb{D}$ appear in our structure $\mathfrak{A}$: more precisely, only the elements in $A \subseteq \mathbb{D}$ appear, and this is called the *active domain* of the database.

- The constant symbols in the active domain are interpreted as themselves: for instance, $\text{actor}^{\mathfrak{A}} \stackrel{\text{def}}{=} \text{actor}$ is one such constant symbol from the active domain.
- Let us write $\mathbb{D}(\varphi)$ for the constant symbols from $\mathbb{D}$ that appear in φ. We still need some way of interpreting those constants outside the active domain, like 'percussionist' in (1.3). For those constants in $\mathbb{D}(\varphi) \setminus A$, we simply add new structure elements and interpret them as themselves. These new elements are unrelated from all the elements in the active domain A.

This means that we actually perform the evaluation of (1.3) on a structure $\mathfrak{A}+\text{percussionist}$, resulting in the correct answer

$$\varphi_{1.3}(\mathfrak{A} + \text{percussionist}) = \{\text{Aretha}\} \ .$$

1.2.2.2. *Semantic Properties.* The semantic properties typically defined for first-order formulæ can be relativised to a class of structures $\mathscr{C}$ (for instance Fin) rather than the class Struct of all structures.

A formula $\varphi \in \mathsf{FO}[\sigma]$ is *satisfiable* in a class of structures $\mathscr{C}$ if there exists a structure $\mathfrak{A} \in \mathscr{C}[\sigma]$ and a valuation $\bar{a} \in A^{\mathsf{free}(\varphi)}$ such that $\mathfrak{A}, \bar{a} \models \varphi$. More generally, a set

$S \subseteq \mathsf{FO}[\sigma]$ of formulæ is *satisfiable* in $\mathscr{C}$ if there exists a structure $\mathfrak{A} \in \mathscr{C}[\sigma]$ and a valuation $\bar{a} \in A^{\mathsf{free}(S)}$ such that $\mathfrak{A}, \bar{a} \models S$.

A formula $\varphi \in \mathsf{FO}[\sigma]$ is *valid* over $\mathscr{C}$ if for all structure $\mathfrak{A} \in \mathscr{C}[\sigma]$ and all valuations $\bar{a} \in A^{\mathsf{free}(\varphi)}$, $\mathfrak{A} \models \varphi(\bar{a})$. Assuming that $\mathscr{C}[\sigma]$ is not empty, a valid formula is in particular satisfiable.

A formula $\varphi \in \mathsf{FO}[\sigma]$ is a (logical) *consequence* over $\mathscr{C}$ of a set $S \subseteq \mathsf{FO}[\sigma]$ of formulæ if, for all structure $\mathfrak{A} \in \mathscr{C}[\sigma]$ and valuations $\bar{a} \in A^{\mathsf{free}(\varphi) \cup \mathsf{free}(S)}$, if $\mathfrak{A}, \bar{a} \models S$, then $\mathfrak{A}, \bar{a} \models \varphi$. If such is the case, we write $S \models_{\mathscr{C}} \varphi$; if $S = \{\psi\}$ we write more simply $\psi \models_{\mathscr{C}} \varphi$. Observe that S is unsatisfiable over $\mathscr{C}$ if and only if $S \models_{\mathscr{C}} \bot$ and that φ is valid over $\mathscr{C}$ if and only if $\emptyset \models_{\mathscr{C}} \varphi$. If both $\varphi \models_{\mathscr{C}} \psi$ and $\psi \models_{\mathscr{C}} \varphi$, then we say that φ and ψ are (logically) *equivalent* over $\mathscr{C}$.

If $\varphi \in \mathsf{FO}[\sigma]$ is a sentence or $T \subseteq \mathsf{FO}[\sigma]$ a set of sentences, we denote their sets of models over $\mathscr{C}$ by

$$\mathsf{Mod}_{\mathscr{C}}(\varphi) \stackrel{\text{def}}{=} \{\mathfrak{A} \in \mathscr{C}[\sigma] \mid \mathfrak{A} \models \varphi\}\ , \quad \mathsf{Mod}_{\mathscr{C}}(T) \stackrel{\text{def}}{=} \bigcap_{\varphi \in T} [\![\varphi]\!]_{\mathscr{C}} = \{\mathfrak{A} \in \mathscr{C}[\sigma] \mid \mathfrak{A} \models T\}\ .$$

Note that $\mathsf{Mod}_{\mathscr{C}}(T)$ is itself a class of structures (see exercise 1.1).[3]

When the class of structures is $\mathscr{C} = \mathsf{Struct}$, we simply say that φ (or S) is satisfiable, valid, a logical consequence, etc., and drop the various '$\mathscr{C}$' subscripts in the notations.

1.2.3. Syntactic Fragments.

By restricting the syntactic constructs in formulæ, we obtain a variety of fragments that (might) constrain the expressiveness of the logic.

Negative Normal Forms. An *atomic formula* is a formula of the form $R(t_1, \ldots, t_{\mathsf{ar}(R)})$ or $t_1 = t_2$. A *literal* is an atomic formula or its negation. Any first-order formula can be put in *negative normal form*, which restricts negations to occur inside litterals, and this form is typically preferred when defining fragments of first-order logic. In other words, these formulæ are defined through the abstract syntax

$$\begin{aligned} \alpha &::= R(t_1, \ldots, t_{\mathsf{ar}(R)}) \mid t_1 = t_2 && \text{(atomic formulæ)} \\ \ell &::= \alpha \mid \neg\alpha && \text{(litterals)} \\ \varphi &::= \ell \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \forall x.\varphi && \text{(formulæ)} \end{aligned}$$

The Alternation Hierarchy. *Quantifier-free* formulæ (in negative normal form) are defined through the abstract syntax

$$\psi ::= \ell \mid \psi \vee \psi \mid \psi \wedge \psi \qquad \text{(QF formulæ)}$$

We write QF for the set of all quantifier-free formulæ. *Existential* formulæ are then defined by

$$\varphi ::= \psi \in \mathsf{QF} \mid \exists x.\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \qquad (\Sigma_1 \text{ formulæ})$$

and similarly, *universal* formulæ are defined by

$$\varphi' ::= \psi \in \mathsf{QF} \mid \forall x.\varphi' \mid \varphi' \vee \varphi' \mid \varphi' \wedge \varphi' \qquad (\Pi_1 \text{ formulæ})$$

[3] Conversely, for a class of structures $\mathscr{C}[\sigma]$ over a single signature σ, its *theory* is the set of sentences that satisfy all the structures in $\mathscr{C}[\sigma]$, i.e., $\mathsf{Th}(\mathscr{C}[\sigma]) \stackrel{\text{def}}{=} \{\varphi \in \mathsf{FO}[\sigma] \mid \mathsf{free}(\varphi) = \emptyset \text{ and } \forall \mathfrak{A} \in \mathscr{C}[\sigma]\ .\ \mathfrak{A} \models \varphi\}$. This definition can also be relativised, this time to various fragments of first-order logic.

Letting $\Sigma_0 = \Pi_0 = \mathsf{QF}$, the previous idea gives rise the *alternation hierarchy* of first-order logic, defined through

$$\varphi ::= \psi_{\in \Pi_i} \mid \exists x.\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \qquad (\Sigma_{i+1} \text{ formulæ})$$
$$\varphi' ::= \psi_{\in \Sigma_i} \mid \forall x.\varphi' \mid \varphi' \vee \varphi' \mid \varphi' \wedge \varphi' \qquad (\Pi_{i+1} \text{ formulæ})$$

Any formula in Σ_i can be equivalently written in *prenex normal form* as $\exists \bar{x}_1 \forall \bar{x}_2 \cdots \exists \bar{x}_i.\psi$ where $\psi \in \mathsf{QF}$ if i is odd, or as $\exists \bar{x}_1 \forall \bar{x}_2 \cdots \forall \bar{x}_i.\psi$ with $\psi \in \mathsf{QF}$ if i is even (this allows some of the 'quantifier blocks' $\exists \bar{x}_j$ or $\forall \bar{x}_j$ to be empty). Similarly, a Π_i formula can be written in prenex normal form as $\forall \bar{x}_1 \exists \bar{x}_2 \cdots \forall \bar{x}_i.\psi$ where $\psi \in \mathsf{QF}$ if i is odd, or as $\forall \bar{x}_1 \exists \bar{x}_2 \cdots \exists \bar{x}_i.\psi$ with $\psi \in \mathsf{QF}$ if i is even. For instance,

$$\exists y.\Big(U(y) \wedge \big(\exists x.\neg R(x,y) \wedge \forall z.\big(S(y,z) \vee \exists x.R(x,z)\big)\big)\Big)$$

is a Σ_3 formula, which can equivalently be written in prenex normal form as

$$\exists x \exists y \forall z \exists x'.U(y) \wedge \big(\neg R(x,y) \wedge (S(y,z) \vee R(x',z))\big)$$

with three quantifier blocks $\exists x \exists y$, $\forall z$, and $\exists x'$ and therefore two alternations. Variables can be freely re-ordered within a quantifier block. One needs to be careful with variable capture when putting a formula in prenex normal form, hence the fresh variable name 'x' in this example.

Clearly, $\Pi_i \subseteq \Sigma_{i+1}$ for all i, but observe that $\Sigma_i \subseteq \Sigma_{i+1}$ also holds for all i; see Figure 1.4 for a depiction. When a formula belongs to Σ_i or Π_i, we say that it has *alternation rank* i.

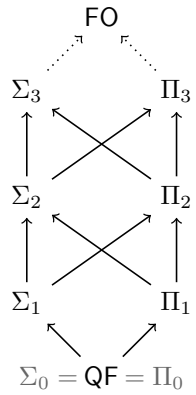


FIGURE 1.4. The alternation hierarchy of first-order logic.

Positive and Negative Fragments. A formula in negative normal form is *positive* if all its atoms are of the form $R(t_1, \ldots, t_{\mathsf{ar}(R)})$ or $t_1 = t_2$. Dually, it is *negative* if all its atoms are of the form $\neg R(t_1, \ldots, t_{\mathsf{ar}(R)})$ or $t_1 \neq t_2$. We write PosFO for the positive fragment of first-order logic. We may sometimes wish to allow disequalities $t_1 \neq t_2$ in the positive fragment, in which case we let $\mathsf{Pos}^{\neq}\mathsf{FO}$ denote the set of first-order formulæ without negative relational atoms $\neg R(t_1, \ldots, t_{\mathsf{ar}(R)})$.

This can be combined with the alternation hierarchy: for instance, $\exists x \exists y. R(x,y) \vee S(y,z)$ is a positive existential formula in $\mathsf{Pos}\Sigma_1$ and $\exists x \exists y. R(x,y) \wedge x \neq y$ is a sentence in $\mathsf{Pos}^{\neq}\Sigma_1$.

A formula is *primitive positive* if it is a positive existential formula without disjunction. A primitive positive formula φ can equivalently be written as $\exists \bar{x}. \bigwedge_{i \in I} \alpha_i(\bar{x}_i)$ where each α_i is an atomic formula with free variables $\bar{x}_i \subseteq (\bar{x} \cup \mathsf{free}(\varphi))$. We write PP for the set of primitive positive formulæ. We have therefore the following proper inclusions among syntactic fragments of first-order logic

$$\mathsf{PP} \quad \subsetneq \quad \mathsf{Pos}\Sigma_1 \quad \subsetneq \quad \mathsf{Pos}^{\neq}\Sigma_1 \quad \subsetneq \quad \Sigma_1 \,.$$

> *Exercise 1.1* (Preservation). Let σ be a signature, $\mathfrak{A}$ and $\mathfrak{B}$ be two structures over σ, and φ a first-order formula over σ. We say that a homomorphism $h \colon \mathfrak{A} \to \mathfrak{B}$ *preserves* φ if, for all $\bar{a}$ in $A^{\mathsf{free}(\varphi)}$,
>
> $$\mathfrak{A} \models \varphi(\bar{a}) \quad \text{implies} \quad \mathfrak{B} \models \varphi(h(\bar{a})) \,.$$
>
> Show that h preserves φ in the following cases.
>
> $\mathfrak{A} \to \mathfrak{B}$: $\varphi \in \mathsf{Pos}\Sigma_1$,
> $\mathfrak{A} \hookrightarrow \mathfrak{B}$: h is injective and $\varphi \in \mathsf{Pos}^{\neq}\Sigma_1$,
> $\mathfrak{A} \subseteq \mathfrak{B}$: h is an embedding and $\varphi \in \Sigma_1$,
> $\mathfrak{A} \twoheadrightarrow \mathfrak{B}$: h is surjective and $\varphi \in \mathsf{PosFO}$,
> $\mathfrak{A} \cong \mathfrak{B}$: h is an isomorphism and $\varphi \in \mathsf{FO}$.
>
> Thus for each fragment above and for all sentences φ in that fragment, $[\![\varphi]\!]_{\mathscr{C}}$ is closed under the corresponding class of morphisms. *Hint: see for instance Hodges (1997, Section 2.4).*

Bounded Variables. One can also restrict first-order formulæ to work over a finite set of variables $\{x_1, \ldots, x_k\}$, where formulæ can be reused; we write FO^k for the set of formulæ that only use k distinct variables. For instance, $\exists y.(E(x,y) \wedge \exists z.E(y,z))$ is equivalent to $\exists y.(E(x,y) \wedge \exists x.E(y,x))$, a formula in FO^3.

1.2.4. Logical Interpretations.

TODO Marker (2002, Definition 1.3.9) Hodges (1997, Section 4.3) and Courcelle and Engelfriet (2012, Chapter 7) (also (**poizat**)?)

Further Reading

References. The material in this chapter is completely standard, and can be found in many textbooks, e.g., by Chang and Keisler (1990), Hodges (1997), or Marker (2002) for model theory in general, or Libkin (2004) for more of a focus on finite model theory. The presentation of the semantics of first-order logic in Section 1.2.2 is also called 'Tarskian semantics' and its origins seem to stem from Tarski (1935).

CHAPTER 2

A Taste of Finite Model Theory

Classical model theory tends to focus on first-order theories, and on the models of those theories, i.e., the structures satisfied by all the sentences in the theory. Finite model theory has a very different flavour, with strong ties with database theory (Vianu, 1997; Vianu, 1997; Kolaitis, 2007; Libkin, 2009).

THEOREM 2.1 (Trakhtenbrot). *undecidability of satisfiability over the class of finite structures*

The proof shown in class is taken from (Toruńczyk, 2022, Theorem 4.2); similar proofs can be found in (Libkin, 2004, Theorem 9.2) or (Arenas et al., 2022, Theorem 8.1).

CHAPTER 3

The Computational Complexity of Model-Checking

As illustrated in Example 1.7, one of the main motivations for studying first-order logic over finite structures stems from the task of answering queries over relational databases. There are however significant differences between first-order logic and SQL, notably

multiset semantics: SQL's semantics allows tuples to appear several times in a table or the output of a query; the set semantics of first-order logic can be simulated in SQL by systematically using the `DISTINCT` keyword in all `SELECT` directives (see Chapter 44 of Arenas et al., 2022);

nulls: SQL allows tuples with missing values, replaced by a value called `NULL`; this leads to considerable complications, for instance SQL uses a three-valued logic rather than a Boolean one (ibid., chapters 39–40).

Furthermore, SQL proposes many additional features that lies beyond the realm of what is expressible in first-order logic; a simple example is aggregation, which is routinely used for counting, summing, … the results of queries (ibid., Chapter 33). Thus first-order logic over finite structures can be seen as a cleaner, better-behaved way of looking at a prominent database problem; databases provide a *motivation* for studying first-order logic over finite structures, but answering real-world queries does not reduce to the first-order case.

This chapter surveys the basic results regarding the evaluation and model-checking problems for first-order logic over finite structures, for which the upcoming Section 3.1 provides the formal definitions and shows that the model-checking problem can be reduced to the case of simple finite graphs (Proposition 3.6). The model-checking problem is shown to be intractable in general in Section 3.2; more precisely it is PSPACE-complete.

Two aspects nevertheless nuance this negative result.

- First, if we fix the input formula—i.e., if we consider the *data complexity* of the problem (see Section 3.1.1)—, then the model-checking problem is in polynomial time (see Theorem 3.14) and even falls into a very low circuit complexity class called uniform AC^0 (see Section 3.4).
- Second, simply fixing the input formula is a bit too coarse, and a finer understanding of the complexity of the problem is gained when treating the size of the input formula as a *parameter*. In this framework developed in Section 3.5, the model-checking problem is not deemed tractable: it is $\mathsf{AW}[*]$-complete, and already $\mathsf{W}[1]$-complete for existential formulæ (Theorem 3.25).

3.1. The Model-Checking Problem

Let us fix a logical query language L (like one of the fragments of Section 1.2.3) along with a class of finite structures $\mathscr{C} \subseteq \mathsf{Fin}$. As discussed, the previous question about querying databases can be abstracted by the following *evaluation problem* for L queries over $\mathscr{C}$.

Problem ($\mathsf{EVAL}(\mathsf{L}, \mathscr{C})$).
instance: a finite relational signature σ, a structure $\mathfrak{A} \in \mathscr{C}[\sigma]$, and a formula $\varphi \in \mathsf{L}[\sigma]$
output: $\varphi(\mathfrak{A})$

Most of our efforts in this course will be dedicated to understanding the complexity of the associated decision problem, called the *model-checking problem*, for various instances of L and $\mathscr{C}$.

Problem ($\mathsf{MC}(\mathsf{L}, \mathscr{C})$).
instance: a finite relational signature σ, a structure $\mathfrak{A} \in \mathscr{C}[\sigma]$, and a sentence $\varphi \in \mathsf{L}[\sigma]$
question: $\mathfrak{A} \models \varphi$?

Note that the formulation of the model-checking problem could equivalently ask whether $\varphi(\mathfrak{A}) \neq \emptyset$.

Throughout this chapter, we will mostly focus on the case where $\mathsf{L} = \mathsf{FO}$ is the full first-order logic, or $\mathsf{L} = \Sigma_i$ is one of its alternation fragments, and $\mathscr{C} = \mathsf{Fin}$ is the class of all finite structures. Spoiler alert: none of these problems is tractable, and all the remaining chapters in these notes are devoted to restrictions on L or $\mathscr{C}$ (or both) that ensure tractability.

In the complexity statements we are going to make, in this chapter as in later ones, the underlying computational model will often be explicit, but if nothing is indicated then we assume a RAM model with unit cost and logarithmic word size.

3.1.1. Data and Query Complexity.

As the model-checking problem takes its roots in query evaluation over databases, where the queries are typically rather small in comparison with the huge amounts of data hosted by database management systems, the ways we will measure its computational complexity are typically refined to encompass

- its *combined complexity*, which is its computational complexity in the usual sense,
- its *query complexity* (aka *expression complexity*), where we fix the input structure $\mathfrak{A}$, and
- its *data complexity*, where we fix the formula φ.

Of those three complexity measures, data complexity is often seen as better reflecting the needs of database management systems.

3.1.2. Encoding the Input.

In order to reason about the complexity of the model-checking problem over various computational models, we need to provide a few details about how to represent its input. Regarding the signature σ, we can order the symbols arbitrarily and provide for each symbol its arity encoded in unary. For the formula φ, we will assume some

natural encoding of the syntactic tree as a string. Regarding the structure $\mathfrak{A}$, there are several possible encodings, with two different flavours that mimic the well-known 'adjacency matrix' and 'adjacency list' representations of graphs. In both cases, we will write $\|\mathfrak{A}\|$ for the size of the encoding of a finite relational structure $\mathfrak{A}$.

3.1.2.1. *Adjacency Matrix Encoding.* The idea with this encoding is to enumerate all the tuples in $\bar{a} \in A^{\mathsf{ar}(R)}$ for each relational symbol $R \in \sigma$, and use one bit to denote whether $\bar{a} \in R^{\mathfrak{A}}$ or not. More precisely, fix an arbitrary linear ordering $<$ over A. Then, for each $R \in \sigma$, its encoding $\langle R^{\mathfrak{A}} \rangle$ is a bitstring of length $|A|^{\mathsf{ar}(R)}$ with a bit for each element of $A^{\mathsf{ar}(R)}$ ordered lexicographically. Then the encoding $\langle \mathfrak{A} \rangle$ of $\mathfrak{A}$ is first the size $|A|$ in unary, followed by a separator and the concatenation of the encodings $\langle R^{\mathfrak{A}} \rangle$ of the $R^{\mathfrak{A}}$'s in the order of the encoding of σ. The size of this encoding of $\mathfrak{A}$ is then

$$\|A\| \stackrel{\text{def}}{=} |A| + 1 + \sum_{R \in \sigma} |A|^{\mathsf{ar}(R)} \ . \tag{3.1}$$

Example 3.1 (Adjacency matrix encoding)**.** Consider the linear ordering $a < b < c$ of the elements of the structure $\mathfrak{A}$ from Figure 1.1. Assume that the encoding of σ presents R and then S (e.g., σ could be encoded as 001001 to represent the arities of R and S). Then $\langle \mathfrak{A} \rangle$ would be the bitstring

$$\underbrace{000}_{|A|}\ \underbrace{1}_{\text{separator}}\ \underbrace{1}_{(a,a)\in R^{\mathfrak{A}}}\ \underbrace{1}_{(a,b)\in R^{\mathfrak{A}}}\ \underbrace{0}_{(a,c)\notin R^{\mathfrak{A}}}\ \underbrace{0}_{(b,a)\notin R^{\mathfrak{A}}}\ 00000\ \underbrace{0}_{(a,a)\notin S^{\mathfrak{A}}}\ 000\ \underbrace{1}_{(b,b)\in S^{\mathfrak{A}}}\ \underbrace{1}_{(b,c)\in S^{\mathfrak{A}}}\ 000$$

3.1.2.2. *Adjacency List Encoding.* The adjacency matrix encoding is wasteful for 'sparse' structures, where $|R^{\mathfrak{A}}|$ is small compared to $|A|^{\mathsf{ar}(R)}$. With an adjacency list encoding, we only represent the tuples $\bar{a} \in R^{\mathfrak{A}}$; incidentally, this is also what is done in database management systems. In order to do so, we fix again an arbitrary linear ordering $<$ over A. We need to represent tuples $\bar{a}$ explicitly, for instance as the concatenation of their elements' numbers (in binary) in this ordering; the encoding of $\langle R^{\mathfrak{A}} \rangle$ is then the concatenation of $|R^{\mathfrak{A}}|$ encodings of tuples, followed by a separator. As before, the encoding $\langle \mathfrak{A} \rangle$ of $\mathfrak{A}$ is first the size $|A|$ in unary, followed by a separator and the concatenation of the encodings $\langle R^{\mathfrak{A}} \rangle$ of the $R^{\mathfrak{A}}$'s in the order of the encoding of σ. The size of this encoding of $\mathfrak{A}$ is then

$$\begin{aligned} \|A\| &\stackrel{\text{def}}{=} |A| + 1 + \sum_{R \in \sigma} \left(\log |A| + |R^{\mathfrak{A}}| \cdot \mathsf{ar}(R) \cdot \log |A| \right) \\ &\in \mathsf{poly}\left(|A| + \sum_{R \in \sigma} (|R^{\mathfrak{A}}| \cdot \mathsf{ar}(R)) \right) . \end{aligned} \tag{3.2}$$

Example 3.2 (Adjacency list encoding)**.** As in Example 3.1, consider the linear ordering $a < b < c$ of A and $R < S$ of σ. Because $|A| + 1 \leq 2^{\ell}$ for $\ell = 2$, we will be able to encode each element of A as a bitstring of length 2 ($\langle a \rangle = 00$, $\langle b \rangle = 01$, and $\langle c \rangle = 10$) and use the bitstring '11' as a separator. Then $\langle \mathfrak{A} \rangle$ would be the bitstring

$$\underbrace{000}_{|A|}\ \underbrace{1}_{\text{separator}}\ \underbrace{0000}_{(a,a)\in R^{\mathfrak{A}}}\ \underbrace{0001}_{(a,b)\in R^{\mathfrak{A}}}\ \underbrace{11}_{\text{separator}}\ \underbrace{0101}_{(b,b)\in S^{\mathfrak{A}}}\ \underbrace{0110}_{(b,c)\in S^{\mathfrak{A}}}$$

From an adjacency matrix encoding, one can compute an adjacency list encoding in time $O(\|\mathfrak{A}\| \cdot \log |A|)$. However, the converse may require time $O(|\sigma| \cdot |A|^{\mathsf{ar}(\sigma)})$, which is

not polynomial when σ is part of the input. By default in the statements of these notes, we will assume an adjacency list encoding, and when needed I will state explicitly that we are assuming an adjacency matrix encoding. Also, some constructions are significantly simpler with an adjacency matrix encoding, and I will first provide proofs for this case.

3.1.3. From Arbitrary Finite Structures to Graphs. In typical instantiations of the model-checking problem, either $\mathsf{ar}(\sigma)$ the *maximal arity* in σ or the signature σ itself is assumed to be fixed; this is often implicit in discussions of the model-checking problem (and could lead to subtle issues). We are going to justify this assumption, and in fact show a stronger result: the model-checking problem over arbitrary finite structures reduces to the problem over simple graphs. We perform this reduction in two stages, first to coloured graphs in Section 3.1.3.1 and second to graphs in Section 3.1.3.2. These reductions are mostly standard, but there are two technical difficulties: one pertains to adjacency list encodings, and the second to preserving levels in the first-order alternation hierarchy.

3.1.3.1. *Reduction to Coloured Bipartite Graphs.* A *coloured graph signature* is a relational signature with one binary relation symbol $E^{(2)}$ along with a finite number of unary relation symbols. A finite structure $\mathfrak{A}$ in $\mathsf{Fin}[\sigma]$ for a coloured graph signature σ is a *coloured graph* if $E^{\mathfrak{A}}$ is irreflexive and symmetric, and is *bipartite* if it does not contain cycles of odd length. We denote by $\mathsf{ColBipartite}$ the set of coloured bipartite graphs over all coloured graph signatures. We establish the following reduction to coloured graphs.

Proposition 3.3 (Reduction to Coloured Bipartite Graphs)**.** *Let $i > 0$. There is a polynomial time many-one reduction $\mathsf{MC}(\Sigma_i, \mathsf{Fin}) \leq^{\mathsf{p}}_m \mathsf{MC}(\Sigma_i, \mathsf{ColBipartite})$ and a polynomial many-one reduction $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin}) \leq^{\mathsf{p}}_m \mathsf{MC}(\mathsf{FO}, \mathsf{ColBipartite})$.*

Proof. Let $\langle \sigma, \mathfrak{A}, \varphi \rangle$ be an instance of $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin})$, thus an instance of $\mathsf{MC}(\Sigma_i, \mathsf{Fin})$ for some $i > 0$. We assume wlog. that φ is in negative normal form. Let us further assume that i is odd, i.e., that the last block of quantifiers of φ is existential; the even case is similar (using the negative fragment of Σ_i instead). We are going to show the following chain of polynomial time many-one reductions

$$\begin{aligned}\mathsf{MC}(\Sigma_i, \mathsf{Fin}) &\leq^{\mathsf{p}}_m \mathsf{MC}(\mathsf{Pos}^{\neq}\Sigma_i, \mathsf{Fin}) \\ &\leq^{\mathsf{p}}_m \mathsf{MC}(\mathsf{Pos}^{\neq}\Sigma_i, \mathsf{ColBipartite}) \\ &\subseteq \mathsf{MC}(\Sigma_i, \mathsf{ColBipartite}) \ ,\end{aligned}$$

where $\mathsf{Pos}^{\neq}\Sigma_i$ denotes the positive fragment of Σ_i with disequalities allowed. As these reductions will be uniform for all $i > 0$ (with a slightly different chain of reductions for even i), this will also yield a reduction $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin}) \leq^{\mathsf{p}}_m \mathsf{MC}(\mathsf{FO}, \mathsf{ColBipartite})$.

Claim 3.3.1. Let $i > 0$ be odd. Then there is a polynomial time many-one reduction $\mathsf{MC}(\Sigma_i, \mathsf{Fin}) \leq^{\mathsf{p}}_m \mathsf{MC}(\mathsf{Pos}^{\neq}\Sigma_i, \mathsf{Fin})$.

Proof of the claim with an adjacency matrix encoding. This is one of those cases where the proof is considerably simple with an adjacency matrix encoding. We construct a new relational signature $\sigma' \supseteq \sigma$ with a disjoint copy $\bar{R}$ of every relation symbol $R \in \sigma$, with the same arity. We then construct an extension $\mathfrak{A}'$ of $\mathfrak{A}$ (thus with the same domain $A' \stackrel{\text{def}}{=} A$ and the same $R^{\mathfrak{A}'} \stackrel{\text{def}}{=} R^{\mathfrak{A}}$ for relation symbols $R \in \sigma$) with the added $\bar{R}^{\mathfrak{A}'} \stackrel{\text{def}}{=} A^{\mathsf{ar}(R)} \setminus R^{\mathfrak{A}}$ for

the new relation symbols. Note that, in the adjacency matrix encoding, it suffices to flip the bits representing $R^{\mathfrak{A}}$ to obtain the representation of $\bar{R}^{\mathfrak{A}'}$. Finally, we replace each negated relational atom $\neg R(\bar{x})$ in φ by $\bar{R}(\bar{x})$ to construct a new formula φ' in $\mathsf{Pos}^{\neq}\Sigma_i$.

Then, there is a simple quantifier-free interpretation I of $\mathfrak{A}'$ in $\mathfrak{A}$ where $I_R(\bar{x}) \stackrel{\text{def}}{=} R(\bar{x})$ and $I_{\bar{R}}(\bar{x}) \stackrel{\text{def}}{=} \neg R(\bar{x})$ for all relation symbols R. Thus $I(\mathfrak{A}) = \mathfrak{A}'$ and $I(\varphi') = \varphi$ are such that $\mathfrak{A}' \models \varphi'$ if and only if $\mathfrak{A} \models \varphi$, as desired. ■

Proof of the claim with an adjacency list encoding. In the case of an adjacency list encoding, there is an issue with the previous handling of negated atoms: computing $A^{\mathsf{ar}(R)} \setminus R^{\mathfrak{A}}$ would be too costly (as the arity $\mathsf{ar}(R)$ can be arbitrary) to be performed directly in a polynomial time reduction. The construction is therefore more involved.

The new signature σ' adds to σ a fresh binary symbol $<^{(2)}$, and, for each relation symbol $R^{(r)} \in \sigma$ such that $R^{\mathfrak{A}} \neq \emptyset$, three new relation symbols $R_f^{(r)}$, $R_\ell^{(r)}$, and $R_s^{(2r)}$.

The new structure $\mathfrak{A}'$ is again an extension of $\mathfrak{A}$, where we define $<^{\mathfrak{A}'}$ as an arbitrary linear ordering (one is readily available in the encoding of $\mathfrak{A}$), and we consider the corresponding lexicographic orderings over tuples in A^r for any $r \leq \mathsf{ar}(\sigma)$, which are quantifier-free definable through

$$\bar{x} <_r \bar{y} \stackrel{\text{def}}{=} \bigvee_{0<k\leq r} \left(x_k < y_k \wedge \bigwedge_{0<j<k} x_j = y_j\right).$$

For every relation symbol $R \in \sigma$ of arity r with $R^{\mathfrak{A}} \neq \emptyset$,

- $R_f^{\mathfrak{A}'}$ only contains the lexicographically smallest tuple in $R^{\mathfrak{A}}$ (the *first* such tuple in lexicographic order),
- $R_\ell^{\mathfrak{A}'}$ the lexicographically largest one (the *last* one), and
- $R_s^{\mathfrak{A}'}$ contains all the *successive* pairs of tuples in $R^{\mathfrak{A}}$, i.e., $(\bar{a}, \bar{a}') \in R_s^{\mathfrak{A}'}$ if and only if $\bar{a} \in R^{\mathfrak{A}}$, $\bar{a}' \in R^{\mathfrak{A}}$, $\bar{a}$ is lexicographically smaller than $\bar{a}'$, and no tuple from $R^{\mathfrak{A}}$ is between them. Note that this new relation holds $|R^{\mathfrak{A}}|^2$ tuples.

Consider now a tuple $\bar{a} \notin R^{\mathfrak{A}}$: the crucial observation is that this tuple must be either smaller than the one in $R_f^{\mathfrak{A}'}$, larger than the one in $R_\ell^{\mathfrak{A}'}$, or between two tuples in $R_s^{\mathfrak{A}'}$.

Then, we replace every occurrence of a negated atom $\neg R(\bar{x})$ in φ either by $x_1 = x_1$ if $R^{\mathfrak{A}} = \emptyset$ (since the negated atom is true in that case), and otherwise by the positive existential formula

$$\begin{aligned}\exists \bar{y}\bar{z}.\big(&R_f(\bar{y}) \wedge \bar{x} <_r \bar{y}\big)\\ &\vee \big(R_\ell(\bar{z}) \wedge \bar{z} <_r \bar{x}\big)\\ &\vee \big(R_s(\bar{y}, \bar{z}) \wedge \bar{y} <_r \bar{x} \wedge \bar{x} <_r \bar{z}\big)\end{aligned}$$

of size polynomial in r (and thus polynomial in the size of the formula $\neg R(\bar{x})$). Then the resulting formula φ' is in $\mathsf{Pos}^{\neq}\Sigma_i$ since i was assumed to be odd, and $\mathfrak{A} \models \varphi$ if and only if $\mathfrak{A}' \models \varphi'$. ■

Example 3.4 (Construction with an adjacency list encoding)**.** Let us illustrate the previous construction on the structure of Figure 1.1. We use the ordering $a < b < c$ as in Example 3.2. Then $R_f^{\mathfrak{A}'} = \{(a, a)\}$, $R_\ell^{\mathfrak{A}'} = \{(a, c)\}$, and $R_s^{\mathfrak{A}'} = \{(a, a, a, b), (a, b, a, c)\}$. For instance, $(b, b) \notin R^{\mathfrak{A}}$ and indeed (b, b) is greater in the lexicographic ordering than (a, c).

Claim 3.4.1. Let $i > 0$ be odd. Then there is a polynomial time many-one reduction $\mathsf{MC}(\mathsf{Pos}^{\neq}\Sigma_i, \mathsf{Fin}) \leq_m^{\mathsf{p}} \mathsf{MC}(\mathsf{Pos}^{\neq}\Sigma_i, \mathsf{ColBipartite})$.

Proof of the claim. This is a classical 'adjacency graph' construction (which is the same regardless of the choice of encoding). We construct a coloured graph signature σ' with one binary symbol $E^{(2)}$, a unary symbol $D^{(1)}$, a unary symbol $P_R^{(1)}$ for each relation symbol $R \in \sigma$ and a unary symbol $Q_j^{(1)}$ for each $j < \mathsf{ar}(\sigma)$.

We construct a coloured graph $\mathsf{Adj}(\mathfrak{A})$ over this signature. Its vertex set contains

- the elements of A coloured with D,
- a vertex $u_{R,\bar{a}}$ coloured with P_R for each relation symbol $R \in \sigma$ and tuple $\bar{a}$, and
- a vertex $v_{R,\bar{a},j}$ coloured with Q_j for each relation symbol $R^{(r)} \in \sigma$, tuple $\bar{a}$, and $0 < j \leq r$.

For each $R^{(r)} \in \sigma$, tuple $\bar{a} \in R^{\mathfrak{A}}$, and $0 < j \leq r$ we have the undirected edges $\{a_i, v_{R,\bar{a},j}\}$ and $\{v_{R,\bar{a},j}, u_{R,\bar{a}}\}$. The resulting graph is bipartite, with the elements of A and the $u_{R,\bar{a}}$ vertices on one partition and the $v_{R,\bar{a},j}$ vertices in the other. See Figure 3.1 for an illustration of the graph and edges we create for a tuple $(a, b, c) \in R^{\mathfrak{A}}$.

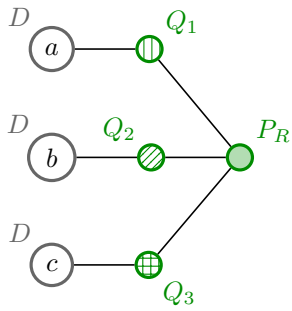


Figure 3.1. The subgraph representing a tuple $(a, b, c) \in R^{\mathfrak{A}}$ in some relational structure. The new vertices are coloured using the appropriate unary symbols.

Then, there is a simple primitive positive interpretation I of $\mathfrak{A}$ in $\mathsf{Adj}(\mathfrak{A})$: its domain formula is $\delta_I(x) \stackrel{\text{def}}{=} D(x)$, and $R_I(\bar{x}) \stackrel{\text{def}}{=} \exists y.P_R(y) \wedge \bigwedge_{0<j\leq r} \left(\exists z.Q_j(z) \wedge E(x_i, z) \wedge E(z, y)\right)$ for each relation symbol $R^{(r)} \in \sigma$; note that these formulæ are of size polynomial in $r \leq |\varphi|$.

Observe that $\varphi' \stackrel{\text{def}}{=} I(\varphi)$ is indeed in $\mathsf{Pos}^{\neq}\Sigma_i$: this is because I is positive existential, i is odd, and φ is in $\mathsf{Pos}^{\neq}\Sigma_i$. Furthermore, $\mathsf{Adj}(\mathfrak{A}) \models I(\varphi)$ if and only if $\mathfrak{A} = I(\mathsf{Adj}(\mathfrak{A})) \models \varphi$ as desired. ■

This concludes the proof of Proposition 3.3. □

As a corollary of the proofs in the case $i = 1$ (thus i odd), which is the existential case, we have the following reduction.

Corollary 3.5 (Existential case)**.** *There is a polynomial time many-one reduction* $\mathsf{MC}(\Sigma_1, \mathsf{Fin}) \leq_m^{\mathsf{p}} \mathsf{MC}(\mathsf{Pos}^{\neq}\Sigma_1, \mathsf{ColBipartite})$.

3.1.3.2. *Reduction to Graphs.* Proposition 3.3 is generally good enough for designing model-checking algorithms, as coloured graphs are not more difficult to handle than general graphs. However, the signature is still not fixed, and it would be nice to show that the model-checking problem with a fixed signature is as hard as the general problem.

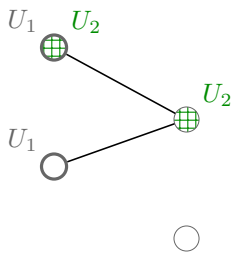
Proposition 3.6 (Reduction to Graphs)**.** *Let $i > 0$. There is a polynomial time many-one reduction* $\mathsf{MC}(\Sigma_i, \mathsf{Fin}) \leq_m^{\mathsf{P}} \mathsf{MC}(\Sigma_i, \mathsf{Graph})$ *and a polynomial time many-one reduction* $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin}) \leq_m^{\mathsf{P}} \mathsf{MC}(\mathsf{FO}, \mathsf{Graph})$.

Proof. As previously, the proof only considers the case where i is odd; the case where i is even is similar. By the proof of Proposition 3.3, it suffices to exhibit a reduction $\mathsf{MC}(\mathsf{Pos}^{\neq}\Sigma_i, \mathsf{ColBipartite}) \leq_m^{\mathsf{P}} \mathsf{MC}(\Sigma_i, \mathsf{Graph})$.

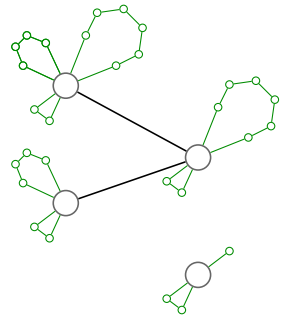
Let $\langle \sigma, G, \varphi \rangle$ be the given instance of $\mathsf{MC}(\mathsf{Pos}^{\neq}\Sigma_i, \mathsf{ColBipartite})$ and let $U_1^{(1)}, \ldots, U_n^{(1)}$ be the unary predicates from σ that actually appear in φ, in some arbitrary order. We extend the graph G into a new graph G' (G will be an induced subgraph of G') as follows:

- for each vertex, add two new vertices and three edges so that we have a copy of the 3-cycle C_3 at this vertex (disjoint from all the other vertices); additionally, if the vertex was isolated, we add one more vertex connected to it;
- for each vertex labelled by U_j, similarly add $2j + 2$ vertices and $2j + 3$ edges so that we have a copy of C_{2j+3} at this vertex (disjoint from all the other vertices).

See Figure 3.2 for an illustration of this transformation.



(A) A bipartite graph, with one vertex coloured by U_1 and U_2 (top left), one by U_1 (bottom left), one by U_2 (top right), and one uncoloured and isolated (bottom right).



(B) The corresponding uncoloured graph, with the added vertices and edges shown in green.

Figure 3.2. Illustration of the replacement of unary predicates in a bipartite graph by suitable cycles.

Because G is bipartite, it does not contain any cycle of odd length. Thus the newly introduced cycles (of odd length) are distinct from all the cycles that might have existed in G. Furthermore, any cycle C_{2j+3} for $0 \leq j \leq n$ can be uniquely identified with an existential sentence (as they form an antichain for the induced subgraph relation).

This leads to an existential interpretation I of G in G', with domain formula

$$\begin{aligned}\delta_I(x_0) \stackrel{\text{def}}{=} \exists x_1 x_2 y. E(x_0, x_1) \wedge E(x_1, x_2) \wedge E(x_2, x_0) \\ \wedge E(x_0, y) \wedge \neg E(x_1, y) \wedge \neg E(x_2, y)\end{aligned}$$

identifying the presence of a C_3 and another connected vertex at this vertex, with edge formula

$$E_I(x_1, x_2) \stackrel{\text{def}}{=} E(x_1, x_2) \ ,$$

and for each $0 < j \leq n$, the colour formula

$$(U_j)_I(x_0) \stackrel{\text{def}}{=} \exists \bar{x}. \big(E(x_j, x_0) \wedge \bigwedge_{0 \leq k < 2j+3} E(x_k, x_{k+1}) \big) \\ \wedge \big(\bigwedge_{0 \leq k < \ell - 1 < 2j+3} \neg E(x_k, x_\ell) \big)$$

identifying the presence of a C_{2j+3} at this vertex; note that these formulæ are of size polynomial in $n \leq |\varphi|$.

Observe that $\varphi' \stackrel{\text{def}}{=} I(\varphi)$ is indeed in Σ_i: this is because I is existential, i is odd, and φ is in $\mathsf{Pos}^{\neq}\Sigma_i$. Furthermore, $G' \models I(\varphi)$ if and only if $G = I(G') \models \varphi$ as desired. □

Exercise 3.1 (Reductions in the even case). Complete the proofs of Proposition 3.3 and Proposition 3.6 in the case where i is even. *Hint: consider* $\mathsf{MC}(\mathsf{Neg}^{=}\Sigma_i, \mathsf{ColBipartite})$ *as an intermediate problem, where* $\mathsf{Neg}^{=}\Sigma_i$ *denotes the set of negative* Σ_i *formulæ allowing positive equality atoms.*

3.2. General Complexity Bounds

The model-checking problem for first-order logic over finite structures is widely held not to be tractable, as it was one of the first problems ever shown to be PSPACE-complete in the original article of Stockmeyer (1976).

Theorem 3.7. $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin})$ *is* PSPACE*-complete in combined complexity. The same holds for any fixed signature* σ*. The same holds for query complexity if the fixed structure has size at least two.*

The Polynomial Hierarchy. As it turns out, one can refine the statement of Theorem 3.7 to take into account the fragment of the first-order alternation hierarchy we are taking our input from. This provides a hierarchy of problems complete for each level of the *polynomial hierarchy*, as we will see later in Theorem 3.8.

Recall to that end that the polynomial hierarchy can be defined through oracles (see, e.g. Arora and Barak, 2009, Definition 3.4 and Section 5.5), by letting

$$\Sigma_1^{\mathsf{P}} \stackrel{\text{def}}{=} \mathsf{NP} \qquad\qquad \Sigma_{i+1}^{\mathsf{P}} \stackrel{\text{def}}{=} \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$$

for all $i > 0$, where NP^C denotes the class of decision problems solvable in non-deterministic polynomial time by a Turing machine with access to an oracle for some problem in C. The union of all the classes in the polynomial hierarchy is the complexity class PH:

$$\mathsf{PH} \stackrel{\text{def}}{=} \bigcup_{i>0} \Sigma_i^{\mathsf{P}} \ .$$

There is therefore an infinite hierarchy of complexity classes

$$\mathsf{P} \subseteq \mathsf{NP} = \Sigma_1^{\mathsf{P}} \subseteq \Sigma_2^{\mathsf{P}} \subseteq \cdots \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE} \ ,$$

where none of the inclusions is known to be strict.

THEOREM 3.8. *Let $i > 0$. Then $\mathsf{MC}(\Sigma_i, \mathsf{Fin})$ is Σ_i^{P}-complete in combined complexity. The same holds for any fixed signature σ. The same holds for query complexity if the fixed structure has size at least two.*

We will prove Theorem 3.7 and Theorem 3.8 simultaneously, as the ingredients of the proofs are essentially the same.

3.2.1. Lower Bounds. The hardness part of the statements in theorems 3.7 and 3.8 can be proven by rather straightforward reductions from very classical decision problems over quantified Boolean formulæ.

Quantified Boolean Formulæ. A *quantified Boolean formula* is a sentence of the form

$$\lambda_1 P_1 \lambda_2 P_2 \dots \lambda_n P_n . \psi$$

where ψ is a propositional formula over the set of propositions $\{P_1, P_2, \dots, P_n\}$ and each λ_i is a quantifier in $\{\exists, \forall\}$. Put differently, this is a sentence in second-order logic over the empty signature, with no first-order quantification nor variables, and restricted to only quantify over nullary relations—i.e., over propositions.

The Σ_i quantified Boolean formulæ are as usual (c.f. Section 1.2.3) those formulæ that start with a block of existential quantifiers, and alternate at most $i-1$ times between blocks of existential quantifiers and blocks of universal quantifiers. For instance,

$$\exists P_1 \exists P_2 \forall P_3 \exists P_4 . (P_1 \vee P_2) \wedge (\neg P_1 \vee P_3) \wedge (\neg P_2 \vee P_3) \wedge (\neg P_3 \vee P_4)$$

is a Σ_3 quantified Boolean formula.

A quantified Boolean formula evaluates to true or false; the above sentence evaluates to true. The decision problems associated with quantified Boolean formulæ are typically used to establish complexity lower bounds for PSPACE and the levels of the polynomial hierarchy.

PROBLEM (TQBF).
instance: a quantified Boolean formula φ
question: does φ evaluate to true?

PROBLEM (Σ_iSAT).
instance: a Σ_i quantified Boolean formula φ
question: does φ evaluate to true?

Fact 3.9 (Stockmeyer, 1976, theorems 4.1 and 5.1)**.** *Let $i > 0$. Then $\Sigma_i\mathsf{SAT}$ is Σ_i^{P}-complete, and* TQBF *is* PSPACE*-complete.*

PROOF OF THE LOWER BOUNDS IN THEOREMS 3.7 AND 3.8. Consider a quantified Boolean formula

$$\varphi = \lambda_1 P_1 \lambda_2 P_2 \dots \lambda_n P_n . \psi \ .$$

Then φ is a Σ_i formula for $i \stackrel{\text{def}}{=} n$ if $\lambda_1 = \exists$ or for $i \stackrel{\text{def}}{=} n+1$ if $\lambda_1 = \forall$. Thus it suffices to exhibit a *single* polynomial time reduction that shows that, for all $i > 0$, $\Sigma_i\mathsf{SAT} \leq_m^{\mathsf{p}} \mathsf{MC}(\Sigma_i, \mathsf{Fin}) \subseteq \mathsf{MC}(\mathsf{FO}, \mathsf{Fin})$ in order to prove both the Σ_i^{P}-hardness and PSPACE-hardness

in the statements of the theorems (see the upcoming exercise 3.2 about more general forms of uniformity in reductions).

We work on the empty signature $\sigma \stackrel{\text{def}}{=} \emptyset$ and the unique structure $\mathfrak{A}$ with two elements over this signature. We construct a Σ_i sentence

$$\varphi' \stackrel{\text{def}}{=} \exists t \exists f \lambda_1 x_1 \lambda_2 x_2 \dots \lambda_n x_n . (t \neq f) \wedge \psi'$$

where ψ' is obtained by replacing every occurrence of a proposition P_j in ψ by $x_j = t$. Then $\mathfrak{A} \models \varphi'$ if and only if φ evaluates to true, as desired.

Finally, regarding the query complexity, observe that this construction carries over any relational signature and any fixed finite structure with at least two elements. □

Exercise 3.2 (Uniform reductions). The goal of this exercise is to formalise the idea of uniform reductions when dealing with hierarchies of problems like $\Sigma_i\mathsf{SAT}$. We focus on the case of PH and PSPACE here but the idea applies more broadly.
Assume there is a decision problem $L \in \mathsf{PSPACE}$, which is also PH-hard, i.e., for all $i > 0$, $\Sigma_i\mathsf{SAT} \leq_m^{\mathsf{p}} L$. More precisely, fix a finite alphabet (for instance $\Sigma = \{0, 1\}$), and recall that a polynomial-time many-one reduction $L' \leq_m^{\mathsf{p}} L$ is defined by a polynomial-time (deterministic) *Turing transducer* M that implements a function $\Sigma^* \to \Sigma^*$ such that, for all $w \in \Sigma^*$, $w \in L'$ if and only if $M(w) \in L$. Formally, the transducer has a read-only input tape containing initially w, a read-write work tape, and a write-only left-to-right output tape that contains $M(w)$ when it halts, which is after $|w|^{O(1)}$ steps. Thus if $\Sigma_i\mathsf{SAT} \leq_m^{\mathsf{p}} L$ for all $i > 0$, then there is a collection $(M_i)_{i>0}$ of such Turing transducers.

(1) What is the consequence if $L \in \mathsf{PH}$? *Hint: for related dire consequences, see* `https://www.scottaaronson.com/writings/phcollapse.pdf`.
(2) Assume that
   - there is a polynomial-time Turing transducer M that takes as input i encoded in unary and outputs the description of each M_i, and
   - there is a polynomial p and a constant d such that all the M_i work in time $p(i) \cdot |w|^d$.

   Show that L is PSPACE-hard under these conditions.

3.2.2. Upper Bounds. Let us turn now to proving the upper bounds in theorems 3.7 and 3.8. As the alternation hierarchy of first-order logic is built on top of the set of quantifier-free formulæ, we first focus our attention to their case. Because quantifier-free formulæ are not sentences, we need an additional ingredient in order to perform model-checking for them: we add to the input of the problem a valuation of the free variables.

Proposition 3.10. *Given as input σ a finite relational signature, $\mathfrak{A} \in \mathsf{Fin}[\sigma]$ a finite structure, $\varphi \in \mathsf{QF}[\sigma]$ a quantifier-free first-order formula, and $\bar{a} \in A^{\mathsf{free}(\varphi)}$ a valuation of the free variables of φ, one can decide whether $\mathfrak{A}, \bar{a} \models \varphi$ in deterministic time $O(|\varphi| \cdot (\|\mathfrak{A}\| + |\bar{a}|))$.*

Proof. We will not modify the valuation $\bar{a} \in A^{\mathsf{free}(\varphi)}$ given as input. The algorithm proceeds recursively, by computing whether $\mathfrak{A}, \bar{a} \models \psi$ for each subformula ψ of φ.

Base cases $x = y$ and $x \neq y$: it suffices to check whether $\bar{a}(x) = \bar{a}(y)$ or not in the encoding of the valuation in the input, thus in $O(|\bar{a}|)$.

Base cases $R(x_{i_1}, \dots, x_{i_{\mathsf{ar}(R)}})$ and $\neg R(x_{i_1}, \dots, x_{i_{\mathsf{ar}(R)}})$: it suffices to check whether the tuple $\bar{a}(x_{i_1}, \dots, x_{i_{\mathsf{ar}(R)}})$ belongs to $R^{\mathfrak{A}}$ or not in the encoding of the structure and

of the valuation (in an adjacency list encoding, we might have to go over the entire encoding of $R^{\mathfrak{A}}$ if not), thus in $O(\|\mathfrak{A}\| + \mathsf{ar}(R) \cdot |\bar{a}|)$.

Inductive cases $\psi \vee \psi'$ and $\psi \wedge \psi'$: the recursive calls for ψ and ψ' tell whether $\mathfrak{A}, \bar{a} \models \psi$ and/or $\mathfrak{A}, \bar{a} \models \psi'$ and can be combined in constant time. □

Exercise 3.3 (Query complexity with a single element). Let $\mathfrak{A}$ be any finite relational structure with $|A| = 1$. Show that the complexity of the model-checking problem for first-order logic over $\mathfrak{A}$ is P-complete. *Hint: you may use the* P*-completeness of* CIRCUIT-EVAL *for the lower bound (see, e.g., Arora and Barak, 2009, Section 7.3).*

3.2.2.1. *Alternating Turing Machines.* In order to prove the complexity upper bounds, it will be very convenient to work with *alternating Turing machines*. Regardless of the exact formal definition of Turing machines we may want to use, it can be extended to handle alternation by partitioning the set of states into *existential* and *universal* states. The configurations of the machine are then called existential or universal accordingly.

What changes is the acceptance condition of an input. We consider for this the (directed) graph of configurations of the Turing machine: the vertices are the configurations, and there is a directed edge $c \to c'$ between two configurations c and c' if there is a transition from c going to c'.

Reachability Games. The configuration graph defines a *game arena* between two players, called the existential player and the universal player, on which the players will move a token from configuration to configuration by following the edges (in game-theoretic terms, what we are about to describe is a two-players, zero-sum game played on a graph with a reachability objective). When the token is on an existential configuration, the existential player chooses to move the token along one of the edges to a new configuration, and conversely, when the token is on a universal configuration, the universal player chooses to move the token along one of outgoing the edges.

Placing a token initially on the initial configuration, the existential player's goal is to reach an accepting configuration, and the universal player's goal is to prevent that from happening. The Turing machine accepts its input if the existential player can play in such a way that, whatever the universal player does, eventually an accepting configuration is reached.

In other words, the machine accepts if the initial configuration is *winning* for the existential player, where the set of winning configurations for the existential player can be defined thus:

- every accepting configuration is winning,
- every existential configuration with an edge to a winning configuration is winning, and
- every universal configuration with all of its edges going to winning configurations is winning.

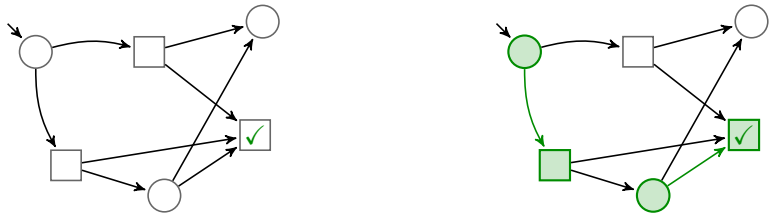
The set of winning configurations can also be defined through as the least fixed-point of an operator on sets of configurations. Let *Acc* denote the set of accepting configurations. Then

$$\mathit{Win} \stackrel{\text{def}}{=} \mu C.\mathit{Acc} \cup \mathsf{Pre}(C)$$

where, for a set C of configurations,

$$\mathsf{Pre}(C) \stackrel{\text{def}}{=} \{c \text{ existential} \mid \exists c' \in C. c \to c'\} \\ \cup \{c \text{ universal} \mid \forall c'. c \to c' \text{ implies } c' \in C\} .$$

Figure 3.3a depicts a graph of configurations, with existential configurations represented by circles, universal configurations by squares, accepting configurations with a checkmark, and the initial configuration with an incoming arrow. Figure 3.3b shows the same



(A) The game arena. (B) The winning region in green.

FIGURE 3.3. A reachability game.

configuration graph, with the winning region of the existential player in green. The figure also highlights a choice of one edge per existential configuration that ensures that the play will remain within the winning region and eventually reach an accepting configuration; this is called a *positional strategy* and, if the existential player is winning, there exists one.

Complexity Classes. Let us define AP as the class of decision problems that can be accepted by an alternating Turing machine working in polynomial time.

An alternating Turing machine is *i-alternation bounded* if any path in the configuration graph alternates at most $(i-1)$ times between existential and universal configurations. For instance, the configuration graph in Figure 3.3a is the graph of a 4-alternation bounded machine. Then let $\mathsf{A}\Sigma_i^{\mathsf{P}}$ be the class of decision problems accepted by i-alternation bounded alternating Turing machines working in polynomial time with an existential initial configuration. Observe that $\mathsf{NP} = \mathsf{A}\Sigma_1^{\mathsf{P}}$; more generally, we have the relation below.

Fact 3.11 (Chandra, Kozen, and Stockmeyer, 1981, Corollary 3.6 and Theorem 4.1)**.** $\mathsf{AP} = \mathsf{PSPACE}$ *and for all* i, $\mathsf{A}\Sigma_i^{\mathsf{P}} = \Sigma_i^{\mathsf{P}}$.

PROOF OF THE UPPER BOUNDS IN THEOREMS 3.7 AND 3.8. We can assume without loss of generality that φ is in prenex normal form: $\varphi = \lambda_1 x_1 \lambda_2 x_2 \cdots \lambda_k x_k . \psi$ with ψ a quantifier-free formula, each $\lambda_j \in \{\exists, \forall\}$. We want to exhibit an alternating Turing machine working in polynomial time that returns whether $\mathfrak{A} \models \varphi$, and performs the same number (say $i-1$) of alternations as there were alternations in the quantifier prefix of the input formula. The machine works in two stages.

First, the machine reads the quantifier prefix $\lambda_1 x_1 \lambda_2 x_2 \cdots \lambda_k x_k$ from the input and writes an assignment $\bar{a} \in A^{x_1 \cdots x_k}$ on a work tape, in time $O(k \cdot |A|)$ overall and using $i-1$ alternations. More precisely, for each $1 \leq j \leq k$, it scans the domain A in the input while staying in an existential state if λ_j is $\exists$ and in a universal state if λ_j is $\forall$; for each element $a \in A$ it has a choice between writing it on the work tape or going to the next element of A; upon the last element of A it has to pick it and move to $j+1$ (or to the next stage if $j = k$).

Second, with the current assignment on its work tape, the machine implements the deterministic polynomial time procedure of Proposition 3.10 to decide whether $\mathfrak{A}, \bar{a} \models \psi$. This is performed in polynomial time, and does not introduce any new alternation (the machine has existential and universal duplicates of the states involved, so that if $\lambda_k = \exists$, then it stays in existential mode, and in universal mode otherwise). □

Exercise 3.4 (AP = PSPACE). Show that AP = PSPACE.

3.3. Towards Practical Algorithms

The PSPACE-completeness of the model-checking problem does not necessarily rule out the existence of reasonably good algorithms on practical instances of the problem. In particular, if we endorse *data complexity* as a better measure of the practical complexity of the model-checking problem than combined complexity, then we are still in the dark.

In this section, we are going to have a glimpse at the algorithmic methods developed for database management systems, and derive a simple algorithm showing that the data complexity of the model-checking problem is in P.

3.3.1. Relational Algebra. Recall the definition in Section 1.2.2.1 of the evaluation of a formula $\varphi(\bar{x})$ over a finite structure $\mathfrak{A}$: $\varphi(\mathfrak{A}) \stackrel{\text{def}}{=} \{\bar{a} \in A^{\bar{x}} \mid \mathfrak{A} \models \varphi(\bar{a})\}$ is the set of valuations in $\mathfrak{A}$ that satisfy $\varphi(\bar{x})$. From a database perspective, this structure can be seen as a table, using the variables in $\bar{x}$ as attribute names. For instance, the evaluation of the first-order formula from Example 1.6 on the structure of Example 1.2 is the table

$\varphi(\mathfrak{A})$	
y	z
a	a
a	b
a	c
b	a

Given a structure $\mathfrak{A}$, define a *relation* over $\mathcal{X}$ as a set of valuations in $A^{\bar{x}}$ for a finite subset $\bar{x} \subseteq_{\text{fin}} \mathcal{X}$. In general, $\varphi(\mathfrak{A})$ is a relation over $\mathcal{X}$. We are going to define an algebra that allows to compute this relation inductively over the formula φ.

3.3.1.1. *Syntax.* Define a *condition* over $\mathcal{X}$ by the abstract syntax

$$\theta ::= x_1 = x_2 \mid \neg\theta \mid \theta \wedge \theta \qquad \text{(conditions)}$$

where x_1, x_2 range over $\mathcal{X}$. We then define a *relational expression* over $\mathcal{X}$ by the abstract syntax

$$e ::= \mathit{Adom}(\bar{x}) \mid R(x_1, \ldots, x_{\mathsf{ar}(R)}) \mid \pi_{\bar{x}}(e) \mid \sigma_\theta(e) \mid e - e \mid e \bowtie e \qquad \text{(relational expressions)}$$

where the x_i's range over $\mathcal{X}$ and $\bar{x}$ ranges over finite subsets of $\mathcal{X}$. The set of variables $\mathsf{free}(\theta)$ or $\mathsf{free}(e)$ of a condition or relational expression is defined inductively by

$$\begin{aligned}
\mathsf{free}(x_1 = x_2) &\stackrel{\text{def}}{=} \{x_1, x_2\} & \mathsf{free}(\neg\theta) &\stackrel{\text{def}}{=} \mathsf{free}(\theta)\\
\mathsf{free}(\theta \wedge \theta') &\stackrel{\text{def}}{=} \mathsf{free}(\theta) \cup \mathsf{free}(\theta')\\
\mathsf{free}(\mathit{Adom}(\bar{x})) &\stackrel{\text{def}}{=} \bar{x} & \mathsf{free}(R(x_1, \ldots, x_{\mathsf{ar}(R)})) &\stackrel{\text{def}}{=} \{x_1, \ldots, x_{\mathsf{ar}(R)}\}\\
\mathsf{free}(\pi_{\bar{x}}(e)) &\stackrel{\text{def}}{=} \bar{x} \cap \mathsf{free}(e) & \mathsf{free}(\sigma_\theta(e)) &\stackrel{\text{def}}{=} \mathsf{free}(e)\\
\mathsf{free}(e - e') &\stackrel{\text{def}}{=} \mathsf{free}(e) \cup \mathsf{free}(e') & \mathsf{free}(e \bowtie e') &\stackrel{\text{def}}{=} \mathsf{free}(e) \cup \mathsf{free}(e') \ .
\end{aligned}$$

Beware here that, e.g., $\mathsf{free}(R(x, x)) = \{x\}$. A relational expression is *well-formed* if

- in all occurrences of the selection operator $\sigma_\theta(e)$, $\mathsf{free}(\theta) \subseteq \mathsf{free}(e)$, and
- in all occurrences of the difference operator $(e - e')$, $\mathsf{free}(e) = \mathsf{free}(e')$.

From now on we will only work with well-formed expressions.

3.3.1.2. *Semantics.* For a condition θ and a valuation $\bar{a} \in A^{\bar{x}}$ where $\mathsf{free}(\theta) \subseteq \bar{x}$, we write $\bar{a} \models \theta$ in the following inductive cases

$$\begin{aligned}
&\bar{a} \models x_1 = x_2 && \text{if } \bar{a}(x_1) = \bar{a}(x_2) \ ,\\
&\bar{a} \models \neg\theta && \text{if } \bar{a} \not\models \theta \ ,\\
&\bar{a} \models \theta \wedge \theta' && \text{if } \bar{a} \models \theta \text{ and } \bar{a} \models \theta' \ .
\end{aligned}$$

If $\bar{a} \in A^{\bar{x}}$ is a valuation of a finite set of variables $\bar{x}$, and $\bar{y}$ is a finite set of variables, let us write $\bar{a}_{\restriction \bar{y}}$ for the valuation in $A^{\bar{x} \cap \bar{y}}$ such that $\bar{a}_{\restriction \bar{y}}(z) = \bar{a}(z)$ for all $z \in \bar{x} \cap \bar{y}$. The semantics $[\![e]\!]^{\mathfrak{A}}$ of a relational expression is a relation over $\mathcal{X}$ defined inductively by

$$\begin{aligned}
[\![\mathit{Adom}(\bar{x})]\!]^{\mathfrak{A}} &\stackrel{\text{def}}{=} A^{\bar{x}} \ , && \text{(full relation)}\\
[\![R(x_1, \ldots, x_{\mathsf{ar}(R)})]\!]^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{\bar{a} \in A^{\{x_1, \ldots, x_{\mathsf{ar}(R)}\}} \mid \mathfrak{A} \models R(\bar{a})\} \ , && \text{(atomic relation)}\\
[\![\pi_{\bar{x}}(e)]\!]^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{\bar{a}_{\restriction \bar{x}} \in A^{\bar{x} \cap \mathsf{free}(e)} \mid \bar{a} \in [\![e]\!]^{\mathfrak{A}}\} \ , && \text{(projection)}\\
[\![\sigma_\theta(e)]\!]^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{\bar{a} \in A^{\mathsf{free}(e)} \mid \bar{a} \models \theta\} \ , && \text{(selection)}\\
[\![e - e']\!]^{\mathfrak{A}} &\stackrel{\text{def}}{=} [\![e]\!]^{\mathfrak{A}} \setminus [\![e']\!]^{\mathfrak{A}} \ , && \text{(difference)}\\
[\![e \bowtie e']\!]^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{\bar{a} \in A^{\mathsf{free}(e) \cup \mathsf{free}(e')} \mid \bar{a}_{\restriction \mathsf{free}(e)} \in [\![e]\!]^{\mathfrak{A}} \text{ and } \bar{a}_{\restriction \mathsf{free}(e')} \in [\![e']\!]^{\mathfrak{A}}\} \ . && \text{(join)}
\end{aligned}$$

The operators of relational expressions are analogues of some of the most used SQL instructions: the projection operator $\pi_{\bar{x}}$ is similar to **SELECT** ..., the join operator $\bowtie$ to **FROM** ... **NATURAL JOIN** ..., the selection operator σ_θ to **WHERE** ..., and the difference operator $-$ to ... **EXCEPT** ...; see Arenas et al. (2022, Chapter 4) for more context.

Example 3.12 (Evaluation of a relational expression)**.** Consider the expression $e \stackrel{\text{def}}{=} \pi_{\{y,z\}}\big(R(x, y) \bowtie (\mathit{Adom}(y, z) - S(y, z))\big)$ and the finite structure of Example 1.2. We can compute the relation $[\![e]\!]^{\mathfrak{A}}$ by induction over e as depicted in Figure 3.4.

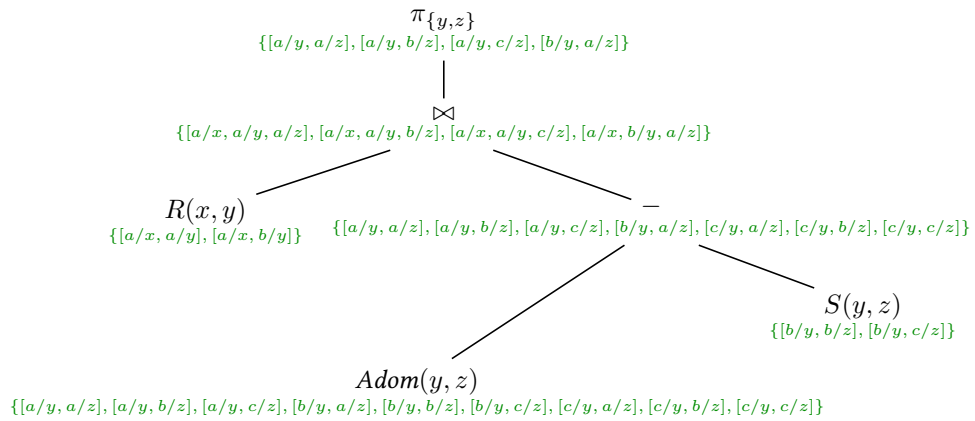


FIGURE 3.4. The evaluation of the relational expression e of Example 3.12 on the finite relational structure of Example 1.2. The semantics $[\![e']\!]^{\mathfrak{A}}$ of each sub-expression e' is provided in green.

3.3.1.3. *Codd's Theorem.* In fact, the relational expression of Example 3.12 computes the evaluation of the formula $\exists x.R(x, y) \wedge \neg S(y, z)$ from Example 1.6. More generally, the relational algebra is merely a reformulation of the definition of the evaluation of a first-order formula on a structure, by induction of the formula.

THEOREM 3.13 (Codd). *Relational algebra expressions and first-order formulæ over a relational signature are equally expressive.*

PROOF. Let us show one direction of the theorem: for all relational signatures σ and first-order formulæ $\varphi \in \mathsf{FO}[\sigma]$, there is a rational algebra expression e_φ with $\mathsf{free}(e) = \mathsf{free}(\varphi)$ and such that, for all finite structures $\mathfrak{A} \in \mathsf{Fin}[\sigma]$, $\varphi(\mathfrak{A}) = [\![e_\varphi]\!]^{\mathfrak{A}}$. The expression is defined by induction on the formula φ as follows:

$$e_{R(x_1,\dots,x_{\mathsf{ar}(R)})} \stackrel{\text{def}}{=} R(x_1, \dots, x_{\mathsf{ar}(R)}) \;, \qquad e_{x_1=x_2} \stackrel{\text{def}}{=} \sigma_{x_1=x_2}(Adom(x_1, x_2)) \;,$$
$$e_{\neg\varphi} \stackrel{\text{def}}{=} Adom(\mathsf{free}(\varphi)) - e_\varphi \;, \qquad e_{\varphi\wedge\psi} \stackrel{\text{def}}{=} e_\varphi \bowtie e_\psi \;,$$
$$e_{\exists x.\varphi} \stackrel{\text{def}}{=} \pi_{\mathsf{free}(\exists x.\varphi)}(e_\varphi) \;.$$

□

Exercise 3.5 (From formulæ to the relational algebra). Show the converse direction of the proof of Theorem 3.13.

3.3.2. Bounded Variable Width. The (variable) *width* $w(\varphi)$ of a first-order formula φ is the maximum $w(\varphi) \stackrel{\text{def}}{=} \max_{\psi \text{ subformula of } \varphi} |\mathsf{free}(\psi)|$ of the number of free variables in any subformula of φ. By definition, $w(\varphi) \geq |\mathsf{free}(\varphi)|$; for instance, for the formula $\varphi(y, z) \stackrel{\text{def}}{=} \exists x.R(x, y) \wedge \neg S(y, z)$ from Example 1.6, $w(\varphi) = 3$.

By evaluating the relational expression e_φ defined in the proof of Theorem 3.13, we are going to prove a general upper bound for both the evaluation and the model-checking problems.

THEOREM 3.14. $\mathsf{EVAL}(\mathsf{FO}, \mathsf{Fin})$ *and* $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin})$ *can be solved in deterministic time* $O(|\varphi| \cdot |A|^{w(\varphi)})$ *with an adjacency matrix encoding.*

PROOF. The computation of $\varphi(\mathfrak{A})$ proceeds by induction over the relational expression e_φ defined in the proof of Theorem 3.13. We represent each $[\![e_\varphi]\!]^{\mathfrak{A}}$ using an adjacency matrix encoding, and work with a RAM machine with unit cost and logarithmic word size.

Case $R(x_1, ..., x_{\mathsf{ar}(R)})$: let $\bar{x} \stackrel{\text{def}}{=} \{x_1, \ldots, x_{\mathsf{ar}(R)}\}$, then $w(R(x_1, \ldots, x_{\mathsf{ar}(R)})) = |\bar{x}|$, which might be smaller than $\mathsf{ar}(R)$. We build the $A^{|\bar{x}|}$ possible valuations $\bar{a} \in A^{\bar{x}}$, and for each valuation $\bar{a}$, check whether $\mathfrak{A} \models R(\bar{a})$. As the input is encoded with an adjacency matrix, we can do this check in constant time (thanks to random access), thus for a total complexity in $O(|A|^{w(R(x_1,\ldots,x_{\mathsf{ar}(R)}))})$.

Note that this is a case where an adjacency list encoding would not yield the result. For instance, for an atomic relation $R(x, x)$, we would need to read the encoding of $R^{\mathfrak{A}}$, which can be of size $O(|A|^2)$.

Case $x_1 = x_2$: let $\bar{x} \stackrel{\text{def}}{=} \{x_1, x_2\}$, then $w(x_1 = x_2) = |\bar{x}| \leq 2$. If $w(x_1 = x_2) = 1$, we return the string with $|A|$ bits set to 1, which encodes $[\![e^{\mathfrak{A}}_{x_1=x_2}]\!] = \{[a/x_1] \mid a \in A\}$ and can be computed in $O(|A|)$. Otherwise $[\![e_{x_1=x_2}]\!]^{\mathfrak{A}} = \{[a/x_1, a/x_2] \mid a \in A\}$, which can be computed in time $O(|A|^2)$.

Case $\neg\varphi$: we have computed $[\![e_\varphi]\!]^{\mathfrak{A}}$ by induction and want to compute $[\![e_{\neg\varphi}]\!]^{\mathfrak{A}} = \mathit{Adom}(\mathsf{free}(\varphi))\setminus [\![e_\varphi]\!]^{\mathfrak{A}}$. We simply flip the bits in the encoding of $[\![e_\varphi]\!]^{\mathfrak{A}}$, in time $O(|A|^{\mathsf{free}(\varphi)})$, which is also in $O(|A|^{w(\varphi)})$.

Case $\exists x.\varphi$: let $\bar{x} \stackrel{\text{def}}{=} \mathsf{free}(\varphi)$, then $|\bar{x}| \leq w(\exists x.\varphi)$. We have computed $[\![e_\varphi]\!]^{\mathfrak{A}}$ by induction and want to compute $[\![e_{\exists x.\varphi}]\!]^{\mathfrak{A}} = \pi_{\mathsf{free}(\exists x.\varphi)}([\![e_\varphi]\!]^{\mathfrak{A}})$. If $x \notin \bar{x}$ there is nothing to do. Otherwise, x is the ith variable in $\bar{x}$. We enumerate the $|A|^{i-1}$ possible prefixes $\bar{b}$ and $|A|^{|\bar{x}|-i}$ possible suffixes $\bar{c}$, and for each $(\bar{b}, \bar{c})$, check whether there exists $a \in A$ such that $\bar{b}a\bar{c} \in [\![e_\varphi]\!]^{\mathfrak{A}}$; each of those checks is in constant time thanks to random access, hence the overall complexity is in $O(|A|^{|\bar{x}|})$, which is in $O(|A|^{w(\exists x.\varphi)})$.

Case $\varphi \wedge \psi$: let $\bar{x} \stackrel{\text{def}}{=} \mathsf{free}(\varphi)$ and $\bar{y} \stackrel{\text{def}}{=} \mathsf{free}(\psi)$, then $w(\varphi \wedge \psi) = |\bar{x} \cup \bar{y}|$. We have computed $[\![e_\varphi]\!]^{\mathfrak{A}}$ and $[\![e_\psi]\!]^{\mathfrak{A}}$ and want to compute $[\![e_\varphi]\!]^{\mathfrak{A}} \bowtie [\![e_\psi]\!]^{\mathfrak{A}}$. We enumerate the $|A|^{|\bar{x}\cup\bar{y}|}$ valuations $\bar{a} \in A^{\bar{x}\cup\bar{y}}$ and check for each one whether $\bar{a}_{\restriction\bar{x}} \in [\![\varphi]\!]^{\mathfrak{A}}$ and $\bar{a}_{\restriction\bar{y}} \in [\![\psi]\!]^{\mathfrak{A}}$. Each check is in constant time, thus the whole computation can be carried in time $O(|A|^{|\bar{x}\cup\bar{y}|})$, which is in $O(|A|^{w(\varphi\wedge\psi)})$.

As every inductive step is in $O(|A|^{w(\varphi)})$, the total time is in $O(|\varphi| \cdot |A|^{w(\varphi)})$ as claimed. □

Regarding the bounded variable fragments defined in Section 1.2.3, Theorem 3.14 entails that $\mathsf{MC}(\mathsf{FO}^k, \mathsf{Fin})$ is in P for every k, since an FO^k formula obviously has width at most k. It also entails that the data complexity of $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin})$ is in P, as $w(\varphi)$ is a constant for every fixed formula φ. This can be considered as good news, as data complexity can be understood as better capturing the actual complexity in database management systems. Unfortunately, the degree of this polynomial complexity depends on $w(\varphi)$, which is rather worrying; we will discuss this further in Section 3.5. The data complexity of the model-checking problem is actually lower than P, as we are going to see now.

3.4. Circuit Complexity

In order to refine our bounds for the data complexity of the model-checking problem, we are going to have a glance at some complexity classes below P. The landscape below P contains classical complexity classes like L or NL that still rely on Turing machines as their model of computation, but there is a large variety of complexity classes based on *circuits*.

The (Boolean) circuits we are going to see in this section are an idealisation of the real electronic circuits used in hardware: they abstract away from practical issues (e.g., signal propagation, power dissipation, circuit area, etc.) in order to provide a very simple computational model—simple enough for complexity theorists to show some meaningful complexity lower bounds

3.4.1. Boolean Circuits. Essentially, a Boolean circuit is a compact representation (as an acyclic graph) of a propositional formula, allowing the sharing of identical subformulæ.

A *Boolean circuit* with n inputs is a tuple $C = (V, E, \beta, o)$ where (V, E) is finite directed acyclic graph, $\beta\colon V \to \{\wedge, \vee, \neg\} \cup \{x_1, \ldots, x_n\}$ is a labelling function, and $o \in V$, such that

- if $\beta(v) \in \{x_1, \ldots, x_n\}$ then v has in-degree 0,
- if $\beta(v) = \neg$ then v has in-degree 1,
- for every $1 \leq i \leq n$ there is at most one vertex $v \in V$ such that $\beta(v) = x_i$.

A vertex v with in-degree k is called a *gate* with *fan-in* k ; the gates v' such that $(v', v) \in E$ are called its *predecessors*. If $\beta(v) = x_i$ for some i then v is an *input gate*; o is called the *output gate*.

The *depth* of a gate v is the length of the longest path from a gate of fan-in 0 to v. The *size* of C is $|V|$ its number of gates, and its *depth* is the depth of its output gate. If all the gates have fan-in at most two, we say that C has *bounded fan-in*.

Semantics. A circuit C with n inputs defines a Boolean function $[\![v]\!]\colon \{0,1\}^n \to \{0,1\}$ by induction on the depth of the gate v. For an input $b_1 \cdots b_n \in \{0,1\}^n$ and a gate v of depth 0, the possible cases are

$$
\begin{aligned}
[\![v]\!](b_1 \cdots b_n) &\stackrel{\text{def}}{=} b_i && \text{if } \beta(v) = x_i \text{ for } 1 \leq i \leq n\ ,\\
[\![v]\!](b_1 \cdots b_n) &\stackrel{\text{def}}{=} 0 && \text{if } \beta(v) = \vee\ , \text{ or}\\
[\![v]\!](b_1 \cdots b_n) &\stackrel{\text{def}}{=} 1 && \text{if } \beta(v) = \wedge\ .
\end{aligned}
$$

Otherwise, v has depth $d > 0$ and its predecessors $v_1, \ldots, v_m$ have all depth less than d, and we can define

$$
\begin{aligned}
[\![v]\!](b_1 \cdots b_n) &\stackrel{\text{def}}{=} \mathsf{not}([\![v_1]\!](b_1 \cdots b_n)) && \text{if } \beta(v) = \neg\ ,\\
[\![v]\!](b_1 \cdots b_n) &\stackrel{\text{def}}{=} \mathsf{or}([\![v_1]\!](b_1 \cdots b_n), \ldots, [\![v_m]\!](b_1 \cdots b_n)) && \text{if } \beta(v) = \vee\ ,\\
[\![v]\!](b_1 \cdots b_n) &\stackrel{\text{def}}{=} \mathsf{and}([\![v_1]\!](b_1 \cdots b_n), \ldots, [\![v_m]\!](b_1 \cdots b_n)) && \text{if } \beta(v) = \wedge\ ,
\end{aligned}
$$

where not, or, and and denote the usual Boolean functions over $\{0,1\}$. Then the Boolean function $[\![C]\!]\colon \{0,1\}^n \to \{0,1\}$ computed by the circuit is the one computed at its output gate:

$$[\![C]\!](b_1 \cdots b_n) \stackrel{\text{def}}{=} [\![o]\!](b_1 \cdots b_n) \ .$$

By the functional completeness of propositional logic, any Boolean function has a Boolean circuit that implements it.

Figure 3.5 presents a Boolean circuit of size 11, depth 3, and maximal fan-in 4, along with its evaluation on the input 101 (in green).

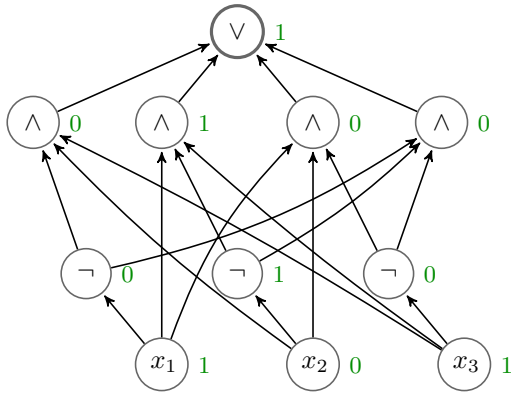


FIGURE 3.5. A Boolean circuit that implements the Boolean function $\oplus^{(3)}\colon \{0,1\}^3 \to \{0,1\}$ that returns 1 if its input contains an even number of 1s. The topmost gate is the output.

3.4.2. Circuit Complexity Classes. As each individual Boolean circuit C is only able to compute a Boolean function $[\![C]\!]\colon \{0,1\}^n \to \{0,1\}$ for a fixed n, in order to define languages of words of arbitrary length in $\{0,1\}^*$, we will consider *families* $(C_n)_{n\in\mathbb{N}}$ of circuit. Then a decision problem $L \subseteq \{0,1\}^*$ is solved by a Boolean circuit family $(C_n)_{n\in\mathbb{N}}$ if, for all $n \in \mathbb{N}$ and $w \in \{0,1\}^n$, $w \in L$ if and only if $[\![C]\!](w) = 1$.

As with other models of computation, we are going to limit the computational resources of our circuit families in order to define complexity classes. Instead of time and space, in the case of circuits, the two main resources are the size and the depth. We will say that a problem is in $\mathsf{SIZEDEPTH}(s,d)$ for two functions $s,d\colon \mathbb{N} \to \mathbb{N}$ if it can be solved by a Boolean circuit family $(C_n)_{n\in\mathbb{N}}$ where, for all n, the size of C_n is at most $O(s(n))$ and its depth at most $O(d(n))$. A well-known (non-uniform) complexity class that can be defined this way is $\mathsf{P/poly} \stackrel{\text{def}}{=} \mathsf{SIZEDEPTH}(\mathsf{poly},\mathsf{poly})$ the set of problems that can be solved by polynomial-sized circuits.

> *Exercise 3.6* (Bounded fan-in). Define similarly $\mathsf{SIZEDEPTH}_2(s,d)$ by further restricting the circuits to have bounded fan-in. Show that $\mathsf{SIZEDEPTH}(s,d) \subseteq \mathsf{SIZEDEPTH}_2(s^2, d \cdot \log s)$. *Hint: see (Vollmer, 1999, Proposition 1.17).*

The complexity class that interests us here is *non-uniform* $\mathsf{AC}^0 \stackrel{\text{def}}{=} \mathsf{SIZEDEPTH}(\mathsf{poly}, 1)$ the class of problems that can be solved by unbounded fan-in, polynomial-size, constant-depth circuits.

Proposition 3.15. *The data complexity of* $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin})$ *with an adjacency matrix encoding is in non-uniform* AC^0.

Proof. Fix a sentence φ in $\mathsf{FO}[\sigma]$ for some relational signature σ, and let $\mathfrak{A} \in \mathsf{Fin}[\sigma]$ be the input structure represented through an adjacency matrix encoding; let $n \stackrel{\text{def}}{=} \|\mathfrak{A}\|$ be the size of the input.

The circuit C_n we construct performs an disjunction between m sub-circuits $C'_{n,m}$, one for each size $m = |A| \leq n$. Each $C'_{n,m}$ performs a conjunction between

- a sub-circuit that checks that the first m bits of the input are 0s and the $(m+1)$th bit is a 1 and
- a sub-circuit that checks that $\mathfrak{A} \models \varphi$.

The latter sub-circuit identifies A with $\{0, \dots, m-1\}$. It has a gate $v(\bar{a} \models \psi)$ for each subformula $\psi(\bar{x})$ of φ and each valuation $\bar{a} \in A^{\bar{x}}$, defined inductively over ψ:

Case of $x_1 = x_2$: then $v(\bar{a} \models x_1 = x_2)$ has no predecessors and is labelled by $\wedge$ if $\bar{a}(x_1) = \bar{a}(x_2)$ and by $\vee$ otherwise.

Case of $R(\bar{x})$: then $v(\bar{a} \models R(\bar{x}))$ is an input gate, labelled by $x_{i(R,\bar{a})}$ where $i(R, \bar{a})$ gives the index of the $\bar{a}$ tuple of R in the encoding of $\mathfrak{A}$.

This is where a matrix encoding is required: with an adjacency list encoding, we cannot predict which bit in the input we are looking for. With a matrix encoding, remember than we identify A with $\{0, \dots, m-1\}$. Let $\sigma = \{R_1, \dots, R_{|\sigma|}\}$, such that $R = R_j$ in this enumeration, $r = \mathsf{ar}(R)$, $\bar{x} = \{x_1, \dots, x_r\}$ (possibly with duplicates), and write $(a_1, \dots, a_r)$ for the vector in $\{0, \dots, m-1\}^r$ defined by $\bar{a}(x_1, \dots, x_r)$. Then

$$i(R, \bar{a}) = m + 1 + \left(\sum_{\ell < j} m^{\mathsf{ar}(R_\ell)} \right) + \left(\sum_{1 \leq i \leq r} m^{r-i} \cdot a_i \right) + 1 \,. \tag{3.3}$$

Case of $\neg\psi$: then $v(\bar{a} \models \neg\psi)$ is labelled by $\neg$ and has $v(\psi, \bar{a})$ as sole predecessor.

Case of $\psi \wedge \psi'$: then $v(\bar{a} \models \psi \wedge \psi')$ is labelled by $\wedge$ and has $v(\bar{a}_{\restriction \mathsf{free}(\psi)} \models \psi)$ and $v(\bar{a}_{\restriction \mathsf{free}(\psi')} \models \psi')$ as predecessors.

Case of $\exists x.\psi$: then $v(\bar{a} \models \exists x.\psi)$ is labelled by $\vee$ and has m predecessors, which are the $v(\bar{a}[b/x] \models \psi)$ for all $b \in A$.

This sub-circuit has output $v([] \models \varphi)$ where $[]$ denotes the empty valuation. It has depth bounded by $|\varphi|$ (more precisely, by the height of the syntax tree of φ) and size bounded by $|\varphi| \cdot |A|^{w(\varphi)}$.

The full circuit C_n has depth bounded by $|\varphi| + O(1)$, which is constant since φ is fixed, and size bounded by $\sum_{1 \leq m \leq n} (|\varphi| \cdot m^{w(\varphi)} + O(m))$, which is polynomial in $n \stackrel{\text{def}}{=} \|\mathfrak{A}\|$ since φ is fixed. See Figure 3.6 for a depiction of the constructed circuit. □

Proposition 3.15 is not yet so informative: as we are going to discuss next in Section 3.4.3, non-uniform AC^0 contains some undecidable problems. Yet constant-depth circuits are quite limited, and complexity theorists also know that some easy problems are *not* in non-uniform AC^0. Define PARITY as the set of words over $\{0,1\}^*$ with an even number of 1s.

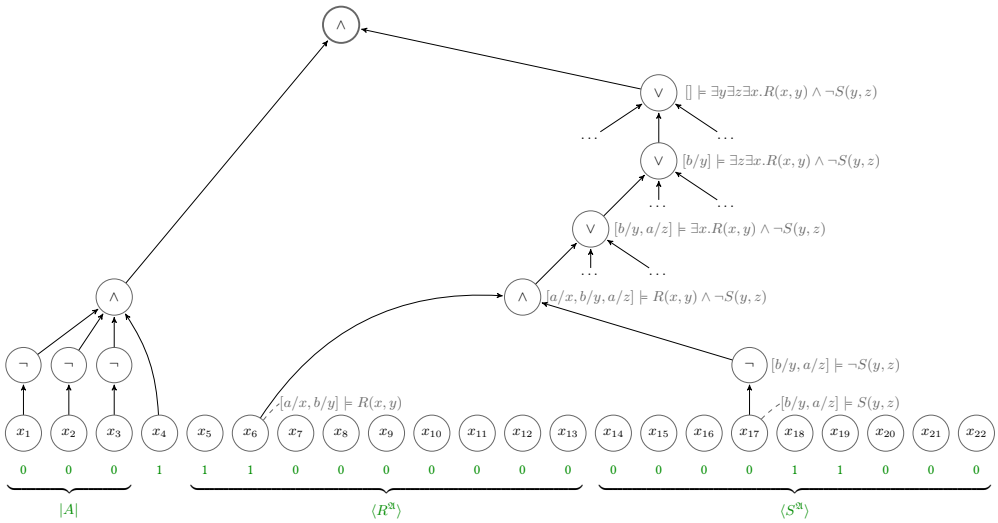


FIGURE 3.6. Part of the circuit $C'_{22,3}$ constructed in the proof of Proposition 3.15 for the formula $\exists y \exists z \exists x. R(x,y) \wedge \neg S(y,z)$. In grey, the corresponding subformula and valuation for each gate. In green below, the input bits from the matrix encoding of the structure of Example 1.2.

Fact 3.16 (Furst, Saxe, and Sipser, 1984; Ajtai, 1983)**.** PARITY *is* not *in non-uniform* AC^0.

3.4.3. Uniformity. An issue with non-uniform circuit complexity classes like P/poly or non-uniform AC^0 is that they contain some undecidable sets, because the choice of the circuit C_n for each input size n can be arbitrarily complex. Consider for this a language $L \subseteq \{0,1\}^*$ such that $L \subseteq \{1\}^*$, i.e., L is a subset of the natural numbers encoded in unary. Then $L \in \mathsf{SIZEDEPTH}(1,1)$: for each size n, let C_n be an $\wedge$ gate without predecessors if $1^n \in L$, and otherwise an $\vee$ gate without predecessors; this circuit family solves L. Thus there exist non-recursive sets in P/poly and in non-uniform AC^0.

In order to define better-behaved complexity classes, we are going to require that a description of each circuit C_n of our family $(C_n)_{n\in\mathbb{N}}$ can be effectively computed. For f a family of computable functions, we will call a circuit family $(C_n)_{n\in\mathbb{N}}$ f-*uniform* if there exists a function $f \in \mathsf{f}$ that produces an encoding $f(n) = \langle C_n \rangle$ when given n in unary. Assuming a somewhat reasonable encoding scheme for $\langle C_n \rangle$, this ensures that the functions computed by f-uniform circuit families are computable.

Logarithmic Time Uniformity. We still need to be careful: the computational complexity of the uniformity class f should not be too high and overpower the complexity of the class of circuits. In the case of AC^0, which is a highly restricted class of circuits, even logspace-uniformity would be too high.

I just provide some intuitions here, as this topic is quite technical and requires low-level bit manipulations. There are actually several possible definitions of uniformity for AC^0, which turn out to be equivalent, but here is a machine-based one.

Let us first fix the encoding. For a circuit family $\mathbf{C} = (C_n)_{n\in\mathbb{N}}$, the *direct connection language* $L_D(\mathbf{C})$ of the family is the set of all tuples $\langle g, b, p, y\rangle$ where

- g is the number of a gate v in C_n, encoded in binary,
- $b = \beta(v)$ is the label of the gate,
- p is the number of a predecessor v' of v, or 0 if v has depth zero, and
- $y = 1^n$ encodes n in unary; this is a padding ensuring that the input has size at least n.

Rather than requiring a function $f \in \mathsf{f}$ such that $f\colon n \mapsto \langle C_n\rangle$ directly, we are going to fix a complexity class C and ask for $L_D(\mathbf{C})$ to be in C.[1]

Now for the complexity class C we want to use, we consider a model of Turing machines with a read-only input tape, a finite number of work tapes of size $\log n$, and an *addressing* tape of size $\log n$. The machine behaves as usual, but has a special instruction to load the ith bit from the input tape on one of its work tapes, where i is written in binary on the addressing tape. A problem is in $\mathsf{DLOGTIME}$ if it is accepted by such a Turing machine in logarithmic time.

Using a suitable numbering of the $v(\bar{a} \models \psi)$ gates from the proof of Proposition 3.15, one can show that this very restricted form of uniformity is enough to solve the model-checking problem (see, e.g., Vollmer, 1999, Theorem 4.73). Furthermore, the converse also holds: any problem that is $\mathsf{DLOGTIME}$-uniform AC^0 reduces to a model-checking instance for a fixed formula.

Fact 3.17 (Barrington, Immermann, and Straubing, 1990, Theorem 8.1)**.** *The data complexity of* $\mathsf{MC}(\mathsf{FO}, \mathsf{Fin})$ *with an adjacency matrix encoding is* $\mathsf{DLOGTIME}$*-uniform* AC^0*-complete.*

Beyond Uniform AC^0. Uniform AC^0 is at the lower end of a hierarchy of uniform circuit complexity classes. Define $\mathsf{NC}^i \stackrel{\text{def}}{=} \mathsf{SIZEDEPTH}_2(\mathsf{poly}, \log^i n)$ for $i > 0$ and $\mathsf{AC}^i \stackrel{\text{def}}{=} \mathsf{SIZEDEPTH}(\mathsf{poly}, \log^i n)$ for $i \geq 0$. Then we have the inclusions $\mathsf{NC}^i \subseteq \mathsf{AC}^i$ for all $i > 0$ by definition and $\mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}$ for all $i \geq 0$ by exercise 3.6. The uniform versions of these classes can be compared to classical complexity classes (see Vollmer, 1999, Corollary 4.2)

$$\mathsf{AC}^0 \subsetneq \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \cdots \subseteq \mathsf{P}\,, \tag{3.4}$$

where only the first inclusion is known to be strict, by Fact 3.16.

3.4.4. Parallel Computation.

The Boolean circuits we use in order to define circuit complexity could be seen as a (rather abstract and not very realistic) model for hardware implementation. Yet, for a problem like the model-checking problem, it even less realistic that anyone would wish to implement the model-checking problem in hardware for one *fixed formula*.

Here is an alternative viewpoint: circuits can also be envisioned as a model of parallel computation. Roughly speaking, each gate can be construed as a very simple processor communicating with other processors through the wires. In this view, a circuit of size s and depth d can be seen as a description of how to compute a particular Boolean function in time d using s processors working in parallel—though it does not explain how to synchronise all these processes nor how to make them communicate.

[1] The two notions of f-uniformity and C-uniformity coincide for large enough space complexities (Vollmer, 1999, Lemma 2.25).

A somewhat more realistic model of computation for parallel computing are *parallel random access machines* (PRAM). This is a deterministic model of computation, where each process has access to its own local memory, and communicates with the other processes through a shared memory. There are various PRAM models depending on the allowed unit operations and the handling of race conditions when accessing the shared memory, and I am not too interested in the specifics; what matters is that the previous intuition about Boolean circuits yielding a model of parallel computation is captured by the following (somewhat informal) fact.

Fact 3.18 (Immerman, 1989)**.** *A problem is in* DLOGTIME*-uniform* AC^0 *if and only if it can be solved by a PRAM with a polynomial number of processors in constant time.*

This yields the following characterisation of the data complexity of model-checking.

Corollary 3.19 (ibid.)**.** *For a fixed sentence* φ*,* $\mathsf{MC}(\varphi, \mathsf{Fin})$ *with an adjacency matrix encoding can be solved by a PRAM with a polynomial number of processors in constant time.*

3.5. Parameterised Complexity

As already mentioned, due to the motivation provided by query evaluation in databases, the data complexity can be considered as a more realistic complexity measure than the combined complexity. Under this distinction, we saw that the data complexity of the finite model checking problem was in uniform AC^0, a low circuit complexity class. However, the restriction to a *fixed* formula used in data complexity is rather coarse. Both in Theorem 3.14 and in Fact 3.17, the degree of the polynomial that bounds the complexity (the time complexity in the first case and the circuit size in the second case) depends on the width of the formula and might quickly grow unwieldy.[2]

3.5.1. Parameterised Problems. Since the proposal of the distinction between combined and data complexities, a better way of understanding the complexity of decision problems with multiple inputs has been developed: parameterised complexity. The basic idea is to identify a *parameter* in our decision problem, as in the following *parameterised model-checking problem*.

> Problem (p-$\mathsf{MC}(\mathsf{L}, \mathscr{C})$).
> **instance:** a finite relational signature σ, a structure $\mathfrak{A} \in \mathscr{C}[\sigma]$, and a sentence $\varphi \in \mathsf{L}[\sigma]$
> **parameter:** $|\varphi|$
> **question:** $\mathfrak{A} \models \varphi$?

What we have done here with the model-checking problem can be defined more generally for decision problems. Fix for this a finite alphabet Σ (which can typically be taken to

[2] In our somewhat naïve presentation, this width is directly related to the number of attributes in the intermediate tables produced when evaluating SQL queries, and this number can quickly get out of hand. The optimisation engines inside database management systems try to tame the size of the intermediate tables (and not just their width), notably through *worst-case optimal join algorithms* (see Arenas et al., 2022, Chapter 26).

be $\{0,1\}$). A decision problem is a language $L \subseteq \Sigma^*$; for instance, $\mathsf{MC}(\mathsf{L},\mathscr{C}) \stackrel{\text{def}}{=} \{\langle \sigma, \mathfrak{A}, \varphi\rangle \mid \mathfrak{A} \in \mathscr{C}[\sigma], \varphi \in \mathsf{L}[\sigma], \text{ and } \varphi(\mathfrak{A}) \neq \emptyset\}$, where $\langle\cdot\rangle$ denotes an encoding function into strings. A parameterised problem simply adds an integer parameter (which will be encoded in unary).

Definition 3.20 (Parameterised Problem)**.** Let Σ be a finite alphabet. A *parameterised problem* is a subset $L \subseteq \Sigma^* \times \mathbb{N}$.

Thus, $\mathsf{p\text{-}MC}(\mathsf{L},\mathscr{C}) \stackrel{\text{def}}{=} \{(\langle \sigma, \mathfrak{A}, \varphi\rangle, |\varphi|) \mid \mathfrak{A} \in \mathscr{C}[\sigma], \varphi \in \mathsf{L}[\sigma], \text{ and } \varphi(\mathfrak{A}) \neq \emptyset\}$ is a parameterised problem. For another example, consider the following parameterised version of the well-known NP-complete SAT, CLIQUE, and COL problems.

PROBLEM (p-SAT).
instance: a propositional formula φ
parameter: its number of propositions
question: is φ satisfiable?

PROBLEM (p-CLIQUE).
instance: a finite simple graph G and $k \in \mathbb{N}$
parameter: k
question: does G contain a clique of size k?

PROBLEM (p-COL).
instance: a finite simple graph G and $k \in \mathbb{N}$
parameter: k
question: can G be coloured with k colours?

Any decision problem $L \subseteq \Sigma^*$ can be 'parameterised' using a *parameterisation*, which is a polynomial-time computable function $\kappa\colon \Sigma^* \to \mathbb{N}$; then $\{(w, \kappa(w)) \mid w \in \Sigma^*\}$ is a parameterised problem. In the case of $\mathsf{MC}(\mathsf{L},\mathscr{C})$, we used the parameterisation $\kappa\colon \langle \sigma, \mathfrak{A}, \varphi\rangle \mapsto |\varphi|$.

3.5.1.1. *Fixed-Parameter Tractability.* The issue with simply fixing a parameter is that it allows for polynomial time algorithms, where the degree of the polynomial can vary wildly depending on the parameter. A nicer notion of tractability is the following, which ensures that the polynomial has constant degree *for all* parameter values.

Definition 3.21 (Fixed-Parameter Tractability)**.** Let Σ be a finite alphabet. A parameterised problem $L \subseteq \Sigma^* \times \mathbb{N}$ is *fixed parameter tractable* if there exists a computable function $f\colon \mathbb{N} \to \mathbb{N}$ and an algorithm deciding whether $(w,k) \in L$ in (deterministic) time

$$f(k) \cdot |w|^{O(1)} .$$

We denote by FPT the class of all fixed parameter tractable problems.

Another way to think about FPT problems is that they are those problems that can be solved in deterministic polynomial time after a pre-computation based on their parameter.

Any problem in P is also in FPT, whatever the chosen parameterisation function; conversely, an FPT problem with constant parameterisation function corresponds to a decision

problem in P. Yet NP-complete problems may also be in FPT depending on the choice of parameters: as any SAT instance φ with k distinct propositions can be solved in time $O(2^k \cdot |\varphi|)$ by a brute-force algorithm, p-SAT $\in$ FPT. In fact, any decidable set $L \subseteq \Sigma^*$ becomes FPT when parameterised with the function $\kappa_{\text{size}} \colon w \mapsto |w|$.

Hence the choice of parameter is essential: it has to depend somehow on the input w (otherwise the notion of fixed-parameter tractability 'trivialises' to polynomial-time), but not the 'whole' of w (otherwise problems that we definitely do not deem tractable would be captured by FPT). Ideally, the parameter should be somehow 'natural' for the problem at hand; for the parameterised model-checking problem, the formula size is a excellent way of refining the notion of data complexity.

3.5.1.2. *Fixed-Parameter Reductions.* In order to show that some parameterised problems are in FPT, or on the contrary to show that they are somehow hard, we use a tailored notion of reductions called *fixed-parameter reductions*.

Definition 3.22 (Fixed-Parameter Reduction)**.** Let $L \subseteq \Sigma^* \times \mathbb{N}$ and $L' \subseteq \Sigma'^* \times \mathbb{N}$ be two parameterised problems over the alphabets Σ and Σ'. A *fixed parameter many-one reduction* (fpt reduction) from L to L' is a function $M \colon \Sigma^* \times \mathbb{N} \to \Sigma'^* \times \mathbb{N}$ such that,

(1) M is a many-one reduction: for all w and k, $M(w,k) \in L'$ if and only if $(w,k) \in L$,
(2) M is computable by an fpt algorithm: there exists a computable function $g \colon \mathbb{N} \to \mathbb{N}$ such that, for all w and k, $M(w,k)$ is computed in time $g(k) \cdot |w|^{O(1)}$, and
(3) the new parameter can be bounded in terms of the old one: there exists a computable function $b \colon \mathbb{N} \to \mathbb{N}$ such that, for all w, k, w' and k', if $M(w,k) = (w',k')$, then $k' \leq b(k)$.

We write $L \leq_m^{\mathsf{fp}} L'$ if there is such a reduction from L to L'. When condition 2 is strengthened to require a polynomial-time algorithm (thus $M(w,k)$ is computed in time $\mathsf{poly}(k,|w|)$), we write $L \leq_m^{\mathsf{fpp}} L'$; note that such polynomial-time fixed-parameter many-one reductions are a restricted class of polynomial-time many-one reductions between parameterised problems.

Condition 3 of Definition 3.22 is crucial in order to ensure the fixed-parameter reductions work as intended. Let us see this definition in action.

Lemma 3.23 (Closure of FPT under fixed-parameter reductions)**.** *Let $L \subseteq \Sigma^* \times \mathbb{N}$ and $L' \subseteq \Sigma'^* \times \mathbb{N}$ be two parameterised problems over the alphabets Σ and Σ'. If $L \leq_m^{\mathsf{fp}} L'$ and $L' \in$ FPT, then $L \in$ FPT.*

Proof. For an input $(w,k) \in \Sigma^* \times \mathbb{N}$, in order to decide whether it belongs to L, we first compute $(w',k') \stackrel{\text{def}}{=} M(w,k)$ in time $g(k) \cdot |w|^c$ for a constant c. Since $L' \in$ FPT, we can answer whether $(w',k') \in L'$ in time $f'(k') \cdot |w'|^{c'}$ for a computable function f' and a constant c'. Observe that $|w'|$ is also in $g(k) \cdot |w|^c$ and that $k' \leq b(k)$, hence this second step is in time $f'(b(k)) \cdot g(k)^{c'} \cdot |w|^{c+c'}$. If we add the time spent in the first step, we remain with a bound in $f'(b(k)) \cdot g(k)^{c+c'} \cdot |w|^{c+c'}$. Thus letting $f \colon k \mapsto f'(b(k)) \cdot g(k)^{c+c'}$, the whole computation is in time $f(k) \cdot |w|^{O(1)}$ for a computable function g, hence $L \in$ FPT. □

Here are now some examples of fixed-parameter reductions: it turns out that all the reductions in Section 3.1.3 were not just polynomial-time reductions, but also fixed-parameter reductions.

Corollary 3.24. *Let $i > 0$. There are polynomial-time fixed-parameter many-one reductions* $\mathsf{p\text{-}MC}(\Sigma_i, \mathsf{Fin}) \leq_m^{\mathsf{fpp}} \mathsf{p\text{-}MC}(\Sigma_i, \mathsf{ColBipartite})$, $\mathsf{p\text{-}MC}(\mathsf{FO}, \mathsf{Fin}) \leq_m^{\mathsf{fpp}} \mathsf{p\text{-}MC}(\mathsf{FO}, \mathsf{ColBipartite})$, $\mathsf{p\text{-}MC}(\Sigma_i, \mathsf{Fin}) \leq_m^{\mathsf{fpp}} \mathsf{p\text{-}MC}(\Sigma_i, \mathsf{Graph})$, *and* $\mathsf{p\text{-}MC}(\mathsf{FO}, \mathsf{Fin}) \leq_m^{\mathsf{fpp}} \mathsf{p\text{-}MC}(\mathsf{FO}, \mathsf{Graph})$.

Proof. By inspection of the proofs in Section 3.1.3. As they were polynomial-time many-one reductions, they satisfy condition 1 and the polynomial-time strengthening of condition 2 in Definition 3.22. In each one of the proofs, we mentioned that the constructed formulæ were also of size bounded in terms of the size of the original formulæ, thus the reductions also satisfy condition 3. □

Exercise 3.7 (Properties of fixed-parameter reductions). Show the following.

(1) The relation $\leq_m^{\mathsf{fp}}$ is reflexive and transitive.
(2) If $L \in \mathsf{FPT}$ and L' is non trivial (i.e., there exist $(w_0, k_0) \notin L'$ and $(w_1, k_1) \in L'$), then $L \leq_m^{\mathsf{fp}} L'$.

Hence $\leq_m^{\mathsf{fp}}$ is a quasi-ordering on parameterised problems, with FPT the set of minimal parameterised problems, which are all equivalent up to fixed-parameter reductions.

3.5.1.3. *Parameterised Intractability.* A first way of designing (presumably) intractable parameterised complexity classes is to draw our inspiration from the P vs. NP question, and design a notion of *parameterised nondeterministic polynomial time* by analogy with the definition of FPT: a parameterised problem is in paraNP if there exists a computable function $f\colon \mathbb{N} \to \mathbb{N}$ and a *nondeterministic* algorithm that solves the problem in nondeterministic time $f(k) \cdot n^{O(1)}$. Thus the parameterised problems p-SAT, p-CLIQUE, and p-COL are all three in paraNP. But only one of them is known to be paraNP-complete under fpt reductions, namely p-COL (essentially because 3COL is already NP-hard for a fixed parameter value, see Flum and Grohe, 2006, Section 2.2)—the same would have happened if we had parameterised SAT with the maximal size of clauses instead of the number of propositions.

Another attempt is to look at complexities like $n^{f(k)}$, which define a class called (uniform) XP. By Theorem 3.14, we know that $\mathsf{p\text{-}MC}(\mathsf{FO}, \mathsf{Fin}) \in \mathsf{XP}$, but this is a huge complexity class, and it seems likely that our parameterised model-checking problem should be somewhat easier.

There is a dazzling variety of parameterised complexity classes between FPT and paraNP or XP in the literature. If we focus on the most relevant ones for model-checking, there are two notable hierarchies, the $\mathsf{W}[i]$ and $\mathsf{A}[i]$ hierarchies, as depicted in Figure 3.7. The inclusions in this figure are not known to be strict, except for the fact that $\mathsf{FPT} \subsetneq \mathsf{XP}$ (see ibid., Corollary 2.26). We do know conditional separations however: $\mathsf{FPT} \neq \mathsf{paraNP}$ if and only if $\mathsf{P} \neq \mathsf{NP}$ (ibid., Corollary 2.13), and under a stronger assumption called the *exponential time hypothesis* (which assumes that there does not exist an algorithm for 3SAT in time $2^{o(n)}$), then $\mathsf{FPT} \subsetneq \mathsf{W}[1]$ (see Cygan et al., 2015, Theorem 14.21). Some examples of problems complete for the various complexity classes are indicated in green; as can be seen in the picture, our parameterised model-checking problems for first-order logic fit in the $\mathsf{A}[i]$ hierarchy.

As a last digression before presenting the $\mathsf{A}[i]$ hierarchy, consider another well-known PSPACE-complete problem related to logic and model-checking: the model-checking problem for *linear temporal logic*. Its 'natural' parameterisation is the following.

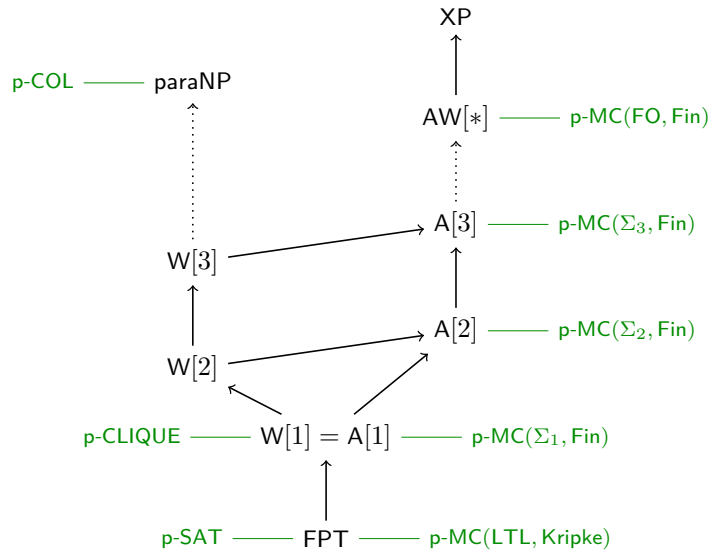


FIGURE 3.7. The inclusions between some parameterised complexity classes.

PROBLEM (p-MC(LTL, Kripke)).
instance: a finite Kripke structure $\mathfrak{M}$, a world w of $\mathfrak{M}$, and an LTL formula φ
parameter: $|\varphi|$
question: does $\mathfrak{M}, w \models \varphi$?

Without entering all the required definitions, this problem is well-known to have an algorithmic solution through the construction of a finite-state Büchi automaton of size $O(2^{|\varphi|})$, using which one obtains a fixed-parameter tractable algorithm running in time $O(2^{|\varphi|} \cdot |\mathfrak{M}|)$ overall (see, e.g. Baier and Katoen, 2008, Section 5.2), hence p-MC(LTL, Kripke) is in FPT.

3.5.1.4. *The* A$[i]$ *Hierarchy.* While the W$[i]$ hierarchy also has a characterisation in terms of model-checking problems (Flum and Grohe, 2006, Theorem 7.22), the A$[i]$ hierarchy is the most relevant for us here. It has a machine characterisation that should feel familiar by now, through the parameterised *short halting problem* for alternating one-tape Turing machines, where we might also fix the number $(i-1)$ of alternations. In the case of a fixed $i = 1$, this problem could equivalently be defined as the short halting problem for *nondeterministic* one-tape Turing machines.[3]

PROBLEM (p-SHORT-ATM).
instance: an alternating single-tape Turing machine M and $k \in \mathbb{N}$
parameter: k

[3]The restriction to *single* tape nondeterministic Turing machines is important, as the same problem for nondeterministic multitape Turing machines is W[2]-complete (Flum and Grohe, 2006, Theorem 7.28).

question: does M accept the empty string in at most k steps?

PROBLEM (p-SHORT-ATM$_i$).
instance: an alternating single-tape Turing machine M whose initial state is existential and $k \in \mathbb{N}$
parameter: k
question: does M accept the empty string in at most k steps and at most $i-1$ alternations?

For each $i > 0$, the parameterised complexity class $\mathsf{A}[i]$ is the class of parameterised problems fixed-parameter reducible to p-SHORT-ATM$_i$. The parameterised complexity class $\mathsf{AW}[*] \supseteq \bigcup_i \mathsf{A}[i]$ is similarly the class of parameterised problems, which are fixed-parameter reducible to p-SHORT-ATM. For an analogy with the polynomial hierarchy, $\mathsf{AW}[*]$ is to PSPACE what the $\mathsf{A}[i]$ are to the Σ_i classes.

Beware that, unlike for instance P, NP, or the classes in the polynomial hierarchy, here the complexity classes are not defined as the classes of languages recognised by some class of machines—the Turing machines in the short halting problem only accept the empty language or the empty word—, but through a reduction closure.

Exercise 3.8 (Short halting problem with a fixed alphabet). Consider a variant of p-SHORT-ATM where we bound the size of the tape alphabet of the Turing machines. Show that this problem is FPT.

3.5.2. The Parameterised Complexity of Model-Checking.

With the definition of the appropriate parameterised complexity classes at hand, we can now conclude this chapter with the following theorem.

THEOREM 3.25. *Let $i > 0$. Then* p-MC(Σ_i, Fin) *is* $\mathsf{A}[i]$*-complete, and* p-MC$(\mathsf{FO}, \mathsf{Fin})$ *is* $\mathsf{AW}[*]$*-complete.*

PROOF OF THE LOWER BOUND. Consider an instance $\langle M, k\rangle$ of p-SHORT-ATM. We are going to exhibit a fixed-parameter many-one reduction to p-MC$(\mathsf{FO}, \mathsf{Fin})$, that will be uniform for all numbers of alternations $(i-1)$ of M, thus also showing that p-SHORT-ATM$_i \leq_m^{\mathsf{fp}}$ p-MC(Σ_i, Fin) for all $i > 0$. Let $i-1$ be the fixed number of alternations, or let $i \stackrel{\text{def}}{=} k+1$ if this number is not fixed.

We need to look a bit under the hood of our Turing machine. Let $M = (Q_\exists \uplus Q_\forall \uplus \{q_{\text{halt}}\}, \Gamma, T, q_0)$ where $Q_\exists$ and $Q_\forall$ are its existential and universal states and q_{halt} its halting state, Γ its finite tape alphabet, $T \subseteq Q \times (\Gamma \cup \{\triangleright, \square\}) \times (\Gamma \cup \{\triangleright\}) \times \{\ell, s, r\}$ its set of transitions (where $\triangleright$ denotes the left end-marker, $\square$ the blank tape symbol, and ℓ, s, and r are the head instructions for moving left, staying in place, and moving right), and $q_0 \in Q_\exists$ is the initial state. We assume wlog. that universal configurations are able to perform at least one transition, and add 'stay' transitions that loop in q_{halt}. By treating q_{halt} as both existential and universal, this means that we are looking for a computation of exactly k steps and with exactly $i-1$ alternations that ends in the halting state.

We construct our instance $\langle \sigma, \varphi, \mathfrak{A}\rangle$ of p-MC$(\mathsf{FO}, \mathsf{Fin})$ as follows.

Signature. First, the signature σ contains the unary relation symbols $B^{(1)}$ (for the blank tape symbol), $L^{(1)}$ (for the left end-marker), $I^{(1)}$ (for the initial state), $H^{(1)}$ (for the halting state), $E^{(1)}$ (for existential states), $U^{(1)}$ (for universal states), and $P_j^{(0)}$ for all $0 \leq j \leq k+1$ (to indicate the head position on the tape). The signature also contains three relation symbols $T_\ell^{(4)}, T_s^{(4)}$, and $T_r^{(4)}$ to encode the transitions. The size of the signature is in $O(k)$: it depends on k, but not on the description of the Turing machine M.

Structure. Second, the finite structure $\mathfrak{A}$ has domain

$$A \stackrel{\text{def}}{=} Q_\exists \cup Q_\forall \cup \{q_{\text{halt}}\} \cup \Gamma \cup \{\square, \triangleright\} \cup \{0, \ldots, k+1\} \; .$$

The interpretations of the relation symbols is

$$\begin{aligned}
B^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{\square\} & L^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{\triangleright\} \\
I^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{q_0\} & H^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{q_{\text{halt}}\} \\
E^{\mathfrak{A}} &\stackrel{\text{def}}{=} Q_\exists \cup \{q_{\text{halt}}\} & U^{\mathfrak{A}} &\stackrel{\text{def}}{=} Q_\forall \cup \{q_{\text{halt}}\} \\
P_j^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{j\} & &\forall 0 \leq j \leq k+1 \; , \\
T_d^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{(q, a, q', b) \mid (q, a, q', b, d) \in T\} & &\forall d \in \{\ell, s, r\} \; .
\end{aligned}$$

The size of the encoding of the structure will depend on the choice of encoding: with an adjacency list encoding, $\|\mathfrak{A}\|$ is in $O(k + |Q| + |T|)$; with an adjacency matrix $\|\mathfrak{A}\|$ is in $O(k + (|Q| \cdot |\Gamma|)^2)$. In both cases, this can be constructed by an FPT algorithm.

Formula. Last, we construct our formula φ. We will say that a tuple $(q, j, \bar{a}) \in A^{k+4}$ is a *configuration* of our Turing machine M if q is in Q (representing the current state), j in $\{0, \ldots, k+1\}$ (representing the head position on the tape), $a_0 = \triangleright$, and the a_js for $j > 0$ are in $\Gamma \cup \{\square\}$ (representing the contents of the left end-marker and the first $k+1$ tape cells).

Let $\bar{z} \stackrel{\text{def}}{=} \{z_0, \ldots, z_{k+1}\}$ and $\bar{z}' \stackrel{\text{def}}{=} \{z'_0, \ldots, z'_{k+1}\}$. Then the formula

$$\mathit{init}(xy\bar{z}) \stackrel{\text{def}}{=} I(x) \wedge P_1(y) \wedge L(z_0) \wedge \bigwedge_{1 \leq j \leq k+1} B(z_j) \tag{3.5}$$

is such that $\mathfrak{A} \models \mathit{init}(q, j, \bar{a})$ if and only if $(q, j, \bar{a})$ is the initial configuration of the Turing machine (with the empty word on the tape). The $\mathit{init}(xy\bar{z})$ formula is of size $O(k)$. We similarly write a formula

$$\begin{aligned}
\mathit{trans}(xy\bar{z}, x'y'\bar{z}') \stackrel{\text{def}}{=} & \; \mathit{left}(xy\bar{z}, x'y'\bar{z}') \\
\vee \; & \; \mathit{stay}(xy\bar{z}, x'y'\bar{z}') \\
\vee \; & \; \mathit{right}(xy\bar{z}, x'y'\bar{z}')
\end{aligned} \tag{3.6}$$

such that $\mathfrak{A} \models \mathit{trans}(q, j, \bar{a}, q', j', \bar{a}')$ if and only if there is a transition of M from the configuration $(q, j, \bar{a})$ to the configuration $(q', j', \bar{a}')$. For instance, for a movement of the head to the left, we can write

$$\begin{aligned}\mathit{left}(xy\bar{z}, x'y'\bar{z}') \stackrel{\text{def}}{=} \bigvee_{1\leq j\leq k} \Big(& P_i(y) \wedge P_{i-1}(y') \\ & \wedge\ T_\ell(x, z_i, z', z'_i) \\ & \wedge \bigwedge_{c\neq j} z_c = z'_c \Big) .\end{aligned} \tag{3.7}$$

The $\mathit{trans}(xy\bar{z}, x'y'\bar{z}')$ formula is again of size $O(k)$.

We now construct our formula φ as

$$\varphi \stackrel{\text{def}}{=} \exists x_0 y_0 \bar{z}_0 . \mathit{init}(x_0 y_0 \bar{z}_0) \wedge \bigvee_{0\leq j\leq k} \mathit{exists}^{(1)}_{0,j}(x_0 y_0 \bar{z}_0) \tag{3.8}$$

where, before performing the ath alternation, $\mathit{exist}^{(a)}_{j,j'}(xy\bar{z})$ checks the existence of a sequence of existential configurations from the jth configuration of the computation to the j'th configuration, and the latter is a winning configuration for the reachability game: either $j = j' = k$ and $a = i + 1$ and then

$$\mathit{exist}^{(i+1)}_{k,k}(x_k y_k \bar{z}_k) \stackrel{\text{def}}{=} H(x_k) \tag{3.9}$$

or $j < j' < k$ and $a < i + 1$ and then

$$\begin{aligned}\mathit{exist}^{(a)}_{j,j'}(x_j y_j \bar{z}_j) \stackrel{\text{def}}{=} & \exists x_{j+1} y_{j+1} \bar{z}_{j+1} \cdots \exists x_{j'} y_{j'} \bar{z}_{j'} . \\ & \bigwedge_{j\leq t<j'} (E(x_t) \wedge \mathit{trans}(x_t y_t \bar{z}_t, x_{t+1} y_{t+1} \bar{z}_{t+1})) \\ & \wedge \bigwedge_{j'\leq j''\leq k} \mathit{forall}^{(a+1)}_{j',j''}(x_{j'} y_{j'} \bar{z}_{j'}).\end{aligned} \tag{3.10}$$

Note that if $j = j' = k$ and $a < i + 1$ then $\mathit{exist}^{(a)}_{k,k}(x_k y_k \bar{z}_k) \stackrel{\text{def}}{=} \mathit{forall}^{(a+1)}_{k,k}(x_k y_k \bar{z}_k)$, i.e., we perform no steps but complete the required number of alternations. The $\mathit{forall}^{(a)}_{j,j'}(xy\bar{z})$ formulæ dually check that for all sequences of configurations we end with a j'th configuration that is winning for the reachability game:

$$\mathit{forall}^{(i+1)}_{k,k}(x_k y_k \bar{z}_k) \stackrel{\text{def}}{=} H(x_k) \tag{3.11}$$

$$\begin{aligned}\mathit{forall}^{(a)}_{j,j'}(x_j y_j \bar{z}_j) \stackrel{\text{def}}{=} & \forall x_{j+1} y_{j+1} \bar{z}_{j+1} \cdots \forall x_{j'} y_{j'} \bar{z}_{j'} . \\ & \Big(\bigwedge_{j\leq t<j'} (U(x_t) \wedge \mathit{trans}(x_t y_t \bar{z}_t, x_{t+1} y_{t+1} \bar{z}_{t+1})) \Big) \\ & \to \bigvee_{j'\leq j''\leq k} \mathit{exist}^{(a+1)}_{j',j''}(x_{j'} y_{j'} \bar{z}_{j'}).\end{aligned} \tag{3.12}$$

The missing formulæ $\mathit{exist}^{(a)}_{j,j'}$ and $\mathit{forall}^{(a)}_{j,j'}$ for $a = i + 1$ and $j, j' < k$, or for $a < i + 1$ and $j = j'$ can all be taken to be $\bot$.

By construction, the formula φ is in Σ_i, and $\mathfrak{A} \models \varphi$ if and only if M halts after at most k steps and $i - 1$ alternations. Furthermore, the size of the formula only depends on k, and

not on the description of the machine M: for fixed alternation level i, the formula is of size $O(k^i)$; when i is not fixed we have necessarily $i \leq k$ thus the formula is of size $O(k^k)$. □

PROOF OF THE UPPER BOUND. Consider an instance $\langle \sigma, \varphi, \mathfrak{A} \rangle$ of p-MC(FO, Fin). We are going to exhibit a fixed-parameter many-one reduction to p-SHORT-ATM, that will be uniform for all $i > 0$ such that $\varphi \in \Sigma_i$, thus also showing that p-MC(Σ_i, Fin) $\leq_m^{\mathsf{fp}}$ p-SHORT-ATM$_i$ for all $i > 0$.

Our task is to compile the entire model-checking problem into a Turing machine M, so that it can check whether $\mathfrak{A} \models \varphi$ in a few steps, which only depend on the size of φ. We assume without loss of generality that φ is given in prenex normal form $\varphi = \lambda_1 x_1 \cdots \lambda_m x_m . \psi$ where ψ is quantifier-free and in negative normal form, $\lambda_1 = \exists$, each $\lambda_j \in \{\exists, \forall\}$, and there are $i-1$ alternations between existential and universal quantifiers. Let $\bar{x} \stackrel{\text{def}}{=} \{x_1, \ldots, x_m\}$.

Our Turing machine will work in two stages. In the first stage, it writes a valuation $\bar{a} \in A^{\bar{x}}$ on its (initially empty) single tape. For each $1 \leq j \leq m$, it has a state q_{j-1} which is existential if $\lambda_j = \exists$ and universal otherwise, with all the transitions $(q_{j-1}, \square, q_j, a, r)$ for $a \in A$. At the end of these m steps, we rewind to the left end of the tape in m additional computation steps using states $q_{m+1}, \ldots, q_{2m}$. Then the tape contents is a tuple in A^m, we are in state q_{2m} and we have performed exactly $i-1$ alternations. From now on, the computation will be deterministic, hence there will be no further quantifier alternations.

The second stage implements a procedure similar to the one presented in Proposition 3.10. However, the encodings of the formula ψ and the structure $\mathfrak{A}$ are not on the tape, but must be carried in the states of the machine. For every (quantifier-free) subformula ψ' of ψ, the machine will have a state $q_{\psi'}$ that will reach a state $q^1_{\psi'}$ if $\mathfrak{A}, \bar{a} \models \psi'$ and a state $q^0_{\psi'}$ if not. Hence we will add stay transitions from q_{2m} to q_ψ and from q^1_ψ to q_{halt} to complete the description of the machine.

Case $\psi_1 \vee \psi_2$: we have stay transitions from $q_{\psi_1 \vee \psi_2}$ to q_{ψ_1}, from $q^1_{\psi_1}$ and $q^1_{\psi_2}$ to $q^1_{\psi_1 \vee \psi_2'}$, from $q^0_{\psi_1}$ to q_{ψ_2}, and from $q^0_{\psi_2}$ to $q^0_{\psi_1 \vee \psi_2}$.

Case $\psi_1 \wedge \psi_2$: similar.

Cases $x_j = x_\ell$ and $x_j \neq x_\ell$: if $j = \ell$ we have a stay transition from $q_{x_j = x_\ell}$ to $q^1_{x_j = x_\ell}$ and from $q_{x_j \neq x_\ell}$ to $q^0_{x_j \neq x_\ell}$; otherwise assuming $j < \ell$ we scan the tape through a sequence of j states, remember the element a_j in the state, perform $\ell - j$ more steps to the right using as many states, check whether $a_j = a_\ell$, remember the outcome of the test, rewind to the left end of the tape, and switch to the appropriate state $q^0_{x_j = x_\ell}$, $q^1_{x_j = x_\ell}$, $q^0_{x_j \neq x_\ell}$, or $q^1_{x_j \neq x_\ell}$ depending on the result of the test. This requires to create $j + |A| \cdot (\ell - j) + 2\ell$ states, and the generated Turing machine will perform $2\ell + 2 \leq m + 2 \in O(|\varphi|)$ computation steps for this phase.

Cases $R(x_{i_1}, \ldots, x_{i_r})$ and $\neg R(x_{i_1}, \ldots, x_{i_r})$: where $r \stackrel{\text{def}}{=} \mathsf{ar}(R) \leq |\varphi|$. Here there are two cases depending on the chosen encoding of $\mathfrak{A}$.

Adjacency matrix: with an adjacency matrix encoding, we can proceed as in the case of equality atoms. We sort the variables $x_{i_1}, \ldots, x_{i_r}$ according to their indices. The Turing machine M loads the tuple $\bar{a}(x_{i_1}, \ldots, x_{i_r})$ in its state from the lowest index to the largest one, in one pass over the m cells of the tape, then rewinds. By keeping the $A^r \leq \|\mathfrak{A}\|$ bits encoding the membership in $R^{\mathfrak{A}}$ inside the states of the generated machine, the generated Turing machine can check whether $\bar{a}(x_{i_1}, \ldots, x_{i_r}) \in R^{\mathfrak{A}}$. This creates $O(m \cdot \|\mathfrak{A}\|)$ states and the generated Turing machine will need $O(m)$ computation steps.

Adjacency list: with an adjacency list encoding, we proceed differently.

For $0 \leq s \leq r$, we say that a tuple $\bar{b} \in A^s$ is *R-extendible* if there exists a tuple $\bar{c} \in A^{r-s}$ such that $\bar{b}\bar{c} \in R^{\mathfrak{A}}$. There are at most $r \cdot |R^{\mathfrak{A}}| \leq \|\mathfrak{A}\|$ R-extendible tuples, hence they can all appear in our states as part of our fixed-parameter reduction.

If the empty tuple is not R-extendible, we can switch directly from $q_{R(\bar{x}')}$ to $q^0_{R(\bar{x}')}$ or from $q_{\neg R(\bar{x}')}$ to $q^1_{\neg R(\bar{x}')}$. Otherwise we do a scan of the valuation on the tape for each variable x_{i_s} before rewinding to the left end-marker, and keep the currently read tuple prefix $\bar{a}(x_{i_1} \cdots x_{i_s})$ in the state. If this tuple is not R-extendible, we move directly to $q^0_{R(\bar{x}')}$ or $q^1_{\neg R(\bar{x}')}$, and otherwise restart from the left end-marker on the tape and scan for i_{s+1} steps to find $\bar{a}(x_{i_{s+1}})$ and continue. Finally, for $s = r$, the tuple is R-extendible if and only if it belongs to $R^{\mathfrak{A}}$, thus we switch to $q^0_{R(\bar{x}')}$, $q^1_{R(\bar{x}')}$, $q^0_{\neg R(\bar{x}')}$, or $q^1_{\neg R(\bar{x}')}$ accordingly. This creates $O(r \cdot m \cdot \|\mathfrak{A}\|)$ states and the constructed Turing machine will need $O(r \cdot m)$ computation steps.

Overall, we build a machine of size $f(k) \cdot \|\mathfrak{A}\|^{O(1)}$ for some computable function f, that will execute in $O(m) + O(|\psi| \cdot |\varphi| \cdot m)$ computation steps, thus in $O(|\varphi|^2)$ steps overall. □

Further Reading

References. For an overview of the contents of this chapter, see (Libkin, 2004, Chapter 6) or (Arenas et al., 2022, Chapter 7).

Query and Data Complexity. The notions of combined, query, and data complexity measures presented in Section 3.1.1 were introduced by Vardi (1982); see for instance (Libkin, 2004, Section 6.1) for a textbook presentation.

Reduction to Graphs. The reductions to the case of graphs in Section 3.1.3 are adapted from the *Normalisation Lemma* given in (Flum and Grohe, 2006, Section 8.2).

PSPACE and the Polynomial Hierarchy. The definition of the polynomial hierarchy is due to Stockmeyer (1976) and that of alternating Turing machines is due to Chandra, Kozen, and Stockmeyer (1981). The PSPACE-hardness of model-checking stated in Theorem 3.7 was first established by Stockmeyer (1976, Theorem 6.1); this is a completely classical result that you can find in many textbooks, e.g., (Libkin, 2004, Theorem 6.16), (Arenas et al., 2022, Theorem 7.1), or (Flum and Grohe, 2006, Proposition 4.28).

See for instance (Arora and Barak, 2009, Section 4.2 and Chapter 5) for more context on PSPACE and PH.

Relational Algebra. Theorem 3.13 was shown by Codd (1972). There are in fact multiple presentations of relational algebras, including 'named' and 'unnamed' ones (see, e.g., chapters 4–6 of Arenas et al., 2022). The relational algebra presented in Section 3.3.1 is specifically tuned for the easy evaluation of first-order formulæ, and slightly adapted from (Libkin, 2004, Section 6.7). In particular,

- the $Adom(\bar{x})$ atomic expression can be expressed in more traditional relational algebras using union and Cartesian products (which I did not include, since they were not required otherwise), and
- traditional named relational algebras feature a separate renaming operator (usually denoted by ρ), which is baked-in in my atomic relations.

Regarding the analysis in terms of the variable width of formulæ in Section 3.3, Theorem 3.14 is a bit of a folklore result, already appearing in the proof of Theorem B.5 by Immerman (1982). Its relevance for FO^k was later emphasised by Vardi (1995, Proposition 3.1). This is a standard result, for which you can also have a look at (Libkin, 2004, Proposition 6.6) or (Flum and Grohe, 2006, Theorem 4.24) for alternative presentations.

Circuit Complexity. The main result of Section 3.4, namely Fact 3.17, originates from (Gurevich and Lewis, 1984; Barrington, Immermann, and Straubing, 1990).

The connections between circuit complexity and parallel random access machines in Fact 3.18 were first shown in a non-uniform setting by Stockmeyer and Vishkin (1984); see for instance (Greenlaw, Hoover, and Ruzzo, 1995, Chapter 2), (Vollmer, 1999, Section 2.7), or (Immerman, 1999, Chapter 5) for more context on parallelism and circuit complexity.

Several circuit complexity classes also enjoy characterisations in terms of descriptive complexity: the decision problems in those classes are exactly the ones that can be described by some logics. Notably for the classes discussed in Section 3.4,

- non-uniform $\mathsf{AC}^0 = \mathsf{FO}[\mathrm{Arb}]$, where the latter denotes the set of languages of non-empty words over the alphabet $\{0,1\}^*$ that are accepted by a first-order formula using arbitrary numerical predicates, while
- $\mathsf{DLOGTIME}$-uniform $\mathsf{AC}^0 = \mathsf{FO}[\mathrm{BIT}]$, the latter being the set of languages over $\{0,1\}^*$ that are accepted by first-order sentences with only the BIT predicate, where $w \models \mathrm{BIT}(i,j)$ for two positions $i, j \in \{0, \dots, |w|-1\}$ if the jth bit in the binary representation of i is a 1 (Barrington, Immermann, and Straubing, 1990, Theorem 8.1).

For more details on those two results, see (Vollmer, 1999, Section 4.5.4), and more generally on the topic of descriptive complexity, see the monograph of Immerman (1999).

Parameterised Complexity. The notions of parameterised complexity, fixed-parameter tractability, fixed-parameter reductions, etc., were introduced by Downey and Fellows and developed in a series of articles starting with (Downey and Fellows, 1995a) and later compiled in their books (Downey and Fellows, 1999; Downey and Fellows, 2013).

The $\mathsf{AW}[*]$-completeness of model-checking in Theorem 3.25 was originally established by Downey, Fellows, and Taylor (1996, reproduced in Section 26.4 of Downey and Fellows, 2013), and improved by Flum and Grohe (2001), who furthermore defined the $\mathsf{A}[i]$ hierarchy, its characterisation through alternating Turing machines, and the $\mathsf{A}[i]$-completeness of model-checking for each Σ_i fragment (see Flum and Grohe, 2006, chapters 6 and 8).

See also the book by Cygan et al. (2015) for more of a focus on the algorithmic and fine-grained complexity aspects of parameterised complexity.

Beyond Model-Checking: Counting and Enumeration. Recall that we were interested in the model-checking problem as a sub-case of the evaluation problem $\mathsf{EVAL}(\mathsf{FO}, \mathsf{Fin})$. The only explicit algorithmic result in this chapter for the evaluation problem is Theorem 3.14,

which yields an $O(|\varphi| \cdot |A|^{w(\varphi)})$ time upper bound; it would be quite easy to adapt the construction of Proposition 3.15 to construct circuits that output $\varphi(\mathfrak{A})$ as well.

But you might have noticed there was a catch: the set $\varphi(\mathfrak{A})$ can be of size $|A|^{|\mathsf{free}(\varphi)|}$ in the worst case, so that it is not even clear what 'tractability' might mean for this problem. It turns out that complexity theorists have developed frameworks for this question:

Counting complexity: rather than outputting $\varphi(\mathfrak{A})$, we could ask for its cardinality. See (Arora and Barak, 2009, Chapter 17) for an introduction to the topic of counting complexity.

Enumeration complexity: where we ask to output the elements of $\varphi(\mathfrak{A})$ one-by-one, without repetitions. See (Strozecki, 2019) for a survey on enumeration complexity.

For a tutorial specifically focusing on the counting and enumeration complexity of the evaluation problem $\mathsf{EVAL}(\mathsf{FO}, \mathsf{Fin})$, see (Durand, 2020).

CHAPTER 4

The Case of Conjunctive Queries

The combined complexity bounds for the model-checking problem established in the previous chapter show that the problem is *hard*—even when restricted to the existential fragment of first-order logic, since the problem $\mathsf{MC}(\Sigma_1, \mathsf{Fin})$ is NP-complete by Theorem 3.8 and the parameterised problem $\mathsf{p\text{-}MC}(\Sigma_1, \mathsf{Fin})$ is W[1]-complete by Theorem 3.25. Furthermore, this holds already on the class of simple finite graphs by Proposition 3.6 and Corollary 3.24. Thus it would make sense to investigate whether the situation is any better when further restricting the syntax to fragments of existential first-order logic.

Ideally, we would also like to work with fragments that remain useful for answering queries in database management systems. A hint about what could be removed from the syntax of first-order logic was given in Example 1.7: it can be rather cumbersome to express negations in SQL. This is a design choice of the SQL language. A first rationale for that choice is that the overwhelming majority of SQL queries (on a well-designed database schema) are of a simple form **SELECT** ... **FROM** ... **WHERE** ..., so that there is little need to facilitate the use of negations. A second rationale is that negations incur expensive evaluation costs; you might have noticed this in the translation from first-order logic into relational algebra in the proof of Codd's Theorem 3.13. Thus, with an eye on the database motivation for the evaluation problem, it makes sense to study its complexity for a fragment of first-order logic devoid of negation. This is the fragment of *conjunctive queries*.

This chapter introduces in the upcoming Section 4.1 the basic definitions for conjunctive queries, and their connection with homomorphisms between finite structures in the *Homomorphism Theorem* (Theorem 4.3). Unfortunately, conjunctive queries do not behave any better than existential sentences in terms of the complexity of their model-checking problem: it is still NP-complete in combined complexity and W[1]-complete in parameterised complexity; see theorems 4.4 and 4.5.

The second part of the chapter therefore investigates an even more restricted class of queries called *acyclic* conjunctive queries in Section 4.2. There at last, Yannakakis' Algorithm (Theorem 4.8) provides a polynomial-time model-checking algorithm in deterministic time $O(\|\mathfrak{A}\| \cdot \log \|\mathfrak{A}\| \cdot |\varphi|)$ for an acyclic conjunctive query φ over a finite structure $\mathfrak{A}$.

4.1. Conjunctive Queries

Recall from Section 1.2.3 that *primitive positive formulæ* over a relational signature σ are defined through the abstract syntax

$$\varphi ::= R(x_1, \ldots, x_{\mathsf{ar}(R)}) \mid x_1 = x_2 \mid \varphi \wedge \varphi \mid \exists x.\varphi \qquad \text{(primitive positive formulæ)}$$

with R ranging over σ and the x_i over $\mathcal{X}$. When writing a primitive positive formula φ in prenex normal form, we obtain a logically equivalent formula

$$\varphi(\bar{x}) ::= \exists \bar{y}. \bigwedge_{i \in I} \alpha_i(\bar{x}_i)$$

for some finite index set I, where each α_i is an atomic formula with free variables $\bar{x}_i \subseteq (\bar{x} \cup \bar{y})$; note here that by definition of $\mathsf{free}(\varphi) = \bar{x}$, $\bar{x} \cap \bar{y} = \emptyset$. Some of these atoms are of the form $x_i = x_j$ for $x_i, x_j \in \bar{x} \cup \bar{y}$, and we can further simplify the syntax by applying the substitutions $[x_i/x_j]$ for each such equality atom. In this way we obtain a logically equivalent *conjunctive query*, which is a formula $\varphi(\bar{x})$ of the form

$$\varphi(\bar{x}) ::= \exists \bar{y}. \bigwedge_{i \in I} R_i(\bar{x}_i) \qquad \text{(conjunctive queries)}$$

where, for each $i \in I$, R_i is a relation symbol in the signature σ and $\bar{x}_i \subseteq (\bar{x} \cup \bar{y})$. If $\bar{x} = \emptyset$ this is called a *Boolean conjunctive query*. Let us write CQ for the set of all conjunctive queries, and $\mathsf{CQ}[\sigma]$ for its restriction to a relational signature σ.

Conjunctive queries are therefore an equivalent syntactic presentation for primitive positive formulæ.[1]

4.1.1. SPJ Algebra. The SQL fragment defined by **SELECT** ... **FROM** ... **WHERE** ... queries corresponds to the *select-project-join* (SPJ) fragment of the relational algebra of Section 3.3.1 defined by the abstract syntax

$$\theta ::= x_1 = x_2 \mid \theta \wedge \theta \qquad \text{(positive conditions)}$$

$$e ::= R(x_1, \ldots, x_{\mathsf{ar}(R)}) \mid \pi_{\bar{x}}(e) \mid \sigma_\theta(e) \mid e \bowtie e \qquad \text{(SPJ expressions)}$$

where, as before, the x_i's range over $\mathcal{X}$ and $\bar{x}$ ranges over finite subsets of $\mathcal{X}$. Then Codd's Theorem (c.f. Theorem 3.13) relativises to the SPJ algebra and conjunctive queries.

Theorem 4.1. *SPJ expressions and conjunctive queries over a relational signature are equally expressive.*

Proof. As in Theorem 3.13, we only treat the direction from conjunctive queries to SPJ expressions. To a conjunctive query $\varphi(\bar{x}) \stackrel{\text{def}}{=} \exists \bar{y}.R_1(\bar{x}_1) \wedge \cdots \wedge R_n(\bar{x}_n)$, we associate the expression

$$e_\varphi \stackrel{\text{def}}{=} \pi_{\bar{x}}\big(R_1(\bar{x}_1) \bowtie \cdots \bowtie R_n(\bar{x}_n)\big) \ . \qquad \square$$

Exercise 4.1 (From conjunctive queries to the SPJ algebra). Show the converse direction of the proof of Theorem 4.1.

[1] In the literature, conjunctive queries sometimes allow equality atoms. This does not impact their expressiveness, nor the complexity of their evaluation problem.

4.1.2. Computational Complexity. Although conjunctive queries are a restriction of existential formulæ, the complexity of their model-checking problem is not easier. The proofs are actually quite straightforward, largely thanks to a nice connection with homomorphisms.

4.1.2.1. *The Homomorphism Problem.* In classical model theory, the (atomic) *diagram* of a structure $\mathfrak{A}$ is obtained by adding to σ a constant for every element of $\mathfrak{A}$; then its diagram is the set of all the literals $\ell(\bar{c})$ with $\bar{c} \in A^{\mathsf{free}(\ell)}$ satisfied by $\mathfrak{A}$. Similarly, the *positive diagram* is the set of all the atomic formulæ $\alpha(\bar{c})$ with $\bar{c} \in A^{\mathsf{free}(\alpha)}$ satisfied by $\mathfrak{A}$; see for instance (Hodges, 1997, Section 1.4) or (Marker, 2002, Definition 2.3.2).

In the case of a finite relational structure $\mathfrak{A}$, we can work with slightly simpler definitions that can be used in essentially the same way. Let $\bar{x}_A \stackrel{\text{def}}{=} \{x_a \mid a \in A\}$ and for any $a_1 \cdots a_r \in A^r$ let $\bar{x}_{\bar{a}} \stackrel{\text{def}}{=} x_{a_1} \cdots x_{a_r}$; then define the positive diagram of $\mathfrak{A}$ as

$$\mathsf{diag}^+(\mathfrak{A}) \stackrel{\text{def}}{=} \exists \bar{x}_A . \bigwedge_{R \in \sigma} \bigwedge_{\bar{a} \in R^{\mathfrak{A}}} R(\bar{x}_{\bar{a}})$$

and observe that this is a Boolean conjunctive query.

Conversely, given a Boolean conjunctive query $\exists \bar{x}. \bigwedge_{i \in I} R_i(x_{i,1}, \ldots, x_{i,\mathsf{ar}(R_i)})$ (thus with $x_{i,1}, \ldots, x_{i,\mathsf{ar}(R_i)} \in \bar{x}$ for all i) over a finite relational signature σ, we can define its *canonical structure* $\mathsf{can}(\varphi) \stackrel{\text{def}}{=} (\bar{x}, (R^{\mathsf{can}(\varphi)})_{R \in \sigma})$ where $R^{\mathsf{can}(\varphi)} \stackrel{\text{def}}{=} \{(x_{i,1}, \ldots, x_{i,\mathsf{ar}(R_i)}) \mid R_i = R\}$ for each relation symbol $R \in \sigma$.

Then, up to isomorphism in Fin and α-renaming in CQ, we can see that the function $\mathsf{can} \colon \mathsf{CQ} \to \mathsf{Fin}$ is the inverse of $\mathsf{diag}^+ \colon \mathsf{Fin} \to \mathsf{CQ}$.

Example 4.2 (Positive diagram)**.** For the finite relational structure of Figure 1.1, its positive diagram is the Boolean conjunctive query

$$\exists x_a x_b x_c . R(x_a, x_a) \wedge R(x_a, x_b) \wedge S(x_b, x_b) \wedge S(x_b, x_c) \ .$$

Conversely, the canonical structure of this Boolean conjunctive conjunctive query is the structure depicted in Figure 1.1.

These definitions lead us to the *Homomorphism Theorem.*

THEOREM 4.3 (Homomorphism Theorem). *Let φ be a Boolean conjunctive query and $\mathfrak{A}$ a finite structure. The following are equivalent*

(1) $\mathfrak{A} \models \varphi$*,*
(2) $\mathsf{can}(\varphi) \to \mathfrak{A}$*, and*
(3) $\mathsf{diag}^+(\mathfrak{A}) \models_{\mathsf{Fin}} \varphi$*.*

PROOF. Let $\varphi \stackrel{\text{def}}{=} \exists \bar{x}. \bigwedge_{i \in I} R_i(x_{i,1}, \ldots, x_{i,\mathsf{ar}(R_i)})$.

(1) iff (2). As $\mathfrak{A} \models \varphi$, there exists a valuation $\bar{a} \in A^{\bar{x}}$ such that $\mathfrak{A}, \bar{a} \models R_i(x_{i,1}, \ldots, x_{i,\mathsf{ar}(R_i)})$ for all $i \in I$. This valuation is a homomorphism from $\mathsf{can}(\varphi)$ to $\mathfrak{A}$. Conversely, if there is a homomorphism h from $\mathsf{can}(\varphi)$ to $\mathfrak{A}$, then for all $i \in I$, $\mathfrak{A} \models R_i(h(x_{i,1}), \ldots, h(x_{i,\mathsf{ar}(R_i)}))$, thus $\mathfrak{A} \models \varphi$.

(1) iff (3). Assume $\mathfrak{A} \models \varphi$ and consider any finite structure $\mathfrak{B}$ such that $\mathfrak{B} \models \mathsf{diag}^+(\mathfrak{A})$. Then as previously this defines a homomorphism h from the structure $\mathsf{can}(\mathsf{diag}^+(\mathfrak{A})) \cong \mathfrak{A}$ to $\mathfrak{B}$. Since φ is a conjunctive query it is positive existential and therefore, by exercise 1.1, $\mathfrak{B} \models \varphi$. Conversely, assume $\mathsf{diag}^+(\mathfrak{A}) \models_{\mathsf{Fin}} \varphi$. Since $\mathfrak{A} \models \mathsf{diag}^+(\mathfrak{A})$, we have $\mathfrak{A} \models \varphi$. □

As an aside, the Homomorphism Theorem can be strengthened to arbitrary conjunctive queries $\varphi(\bar{x})$—and not only the Boolean ones—if we extend the signature with one constant symbol per free variable in $\bar{x}$. I will let you figure out the details.

This leads to the definition of two new decision problems with strong connections with the model-checking problem for conjunctive queries. For fixed classes of structures $\mathscr{C}, \mathscr{D} \subseteq \mathsf{Fin}$, the *homomorphism problem* asks for the existence of a homomorphism between two structures given as input

PROBLEM ($\mathsf{HOM}(\mathscr{C}, \mathscr{D})$).
instance: a finite relational signature σ, a structure $\mathfrak{A} \in \mathscr{C}[\sigma]$, and a structure $\mathfrak{B} \in \mathscr{D}[\sigma]$
question: $\mathfrak{A} \to \mathfrak{B}$?

For a logic L, the *finite query containment problem* asks whether one sentence φ is a logical consequence of another sentence ψ over the class of finite structures—or, equivalently, whether $\mathsf{Mod}_{\mathsf{Fin}}(\psi) \subseteq \mathsf{Mod}_{\mathsf{Fin}}(\varphi)$.

PROBLEM ($\mathsf{CONTAINMENT}(\mathsf{L})$).
instance: a finite relational signature σ and two sentences $\varphi, \psi \in \mathsf{L}[\sigma]$
question: $\psi \models_{\mathsf{Fin}} \varphi$?

The Homomorphism Theorem shows that the model-checking problem $\mathsf{MC}(\mathsf{CQ}, \mathsf{Fin})$ for conjunctive queries, the homomorphism problem $\mathsf{HOM}(\mathsf{Fin}, \mathsf{Fin})$, and the finite query containment problem $\mathsf{CONTAINMENT}(\mathsf{CQ})$ for Boolean conjunctive queries are essentially the same.

Exercise 4.2 (Reductions between the homomorphism and model-checking problems). Show that $\mathsf{MC}(\mathsf{CQ}, \mathsf{Fin}) \equiv^{\mathsf{P}}_{m} \mathsf{HOM}(\mathsf{Fin}, \mathsf{Fin})$, with either an adjacency matrix or an adjacency list encoding. *Hint: in order to prove that* $\mathsf{MC}(\mathsf{CQ}, \mathsf{Fin}) \leq^{\mathsf{P}}_{m} \mathsf{HOM}(\mathsf{Fin}, \mathsf{Fin})$ *in the case of an adjacency matrix encoding, use the proof of Claim 3.4.1.*

4.1.2.2. *Combined, Query, and Data Complexities.* Let us now turn to the combined and query complexity of the model-checking problem for Boolean conjunctive queries. As already mentioned, the problem remains as hard as for the existential fragment.

THEOREM 4.4. $\mathsf{MC}(\mathsf{CQ}, \mathsf{Fin})$ *is* NP*-complete in combined complexity and in query complexity.*

PROOF. The membership in NP follows from the case of existential sentences in Theorem 3.8. Regarding NP-hardness, we reduce the three colourability problem 3COL to the homomorphism problem $\mathsf{HOM}(\mathsf{Graph}, [K_3]_{\cong})$ for a fixed target graph, namely the triangle C_3. By exercise 4.2, this will entail the NP-hardness of $\mathsf{MC}(\mathsf{CQ}, \mathsf{Fin})$. The reduction is immediate: given an input graph $G \in \mathsf{Graph}$, G is three colourable if and only if it has a homomorphism into C_3. See Figure 4.1 for an illustration. □

Regarding data complexity, the upper bounds for general first-order sentences in $O(|\varphi| \cdot |A|^{w(\varphi)})$ and in uniform AC^0 of Theorem 3.14 and Fact 3.17 also hold for the model-checking problem for conjunctive queries.

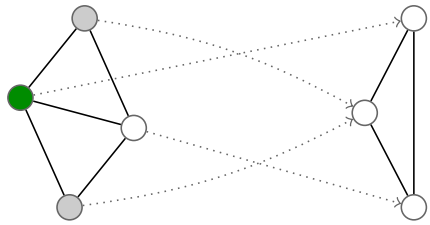


FIGURE 4.1. A 3-colourable graph (left) and a homomorphism into C_3 (right) depicted through dotted arrows.

4.1.2.3. *Parameterised Complexity.* Again, the parameterised model-checking problem for Boolean conjunctive queries is as hard as for existential sentences. The hardness proof relies on the W[1]-completeness of p-CLIQUE, which was already mentioned in Figure 3.7.

First, let us define W[1]. This parameterised complexity class is the class of parameterised problems fixed-parameter reducible to the following short halting problem for nondeterministic Turing machines.

PROBLEM (p-SHORT-NTM).
instance: a nondeterministic single-tape Turing machine M and $k \in \mathbb{N}$
parameter: k
question: does M accept the empty string in at most k steps?

Clearly, p-SHORT-NTM $=$ p-SHORT-ATM$_1$ and therefore W[1] $=$ A[1] as announced in Figure 3.7.

THEOREM 4.5. p-CLIQUE *and* p-MC(CQ, Fin) *are* W[1]*-complete.*

PROOF. Theorem 3.25 shows a fixed-parameter many-one reduction from p-SHORT-NTM $=$ p-SHORT-ATM$_1$ to p-MC(Σ_1, Fin). In turn, the proof of Proposition 3.3 actually shows a fixed-parameter many-one reduction from the latter to p-MC(Pos$^{\neq}\Sigma_1$, ColBipartite). The theorem then follows from the following two claims.

Claim 4.5.1. There is a fixed-parameter many-one reduction p-MC(Pos$^{\neq}\Sigma_1$, ColBipartite) $\leq_m^{\mathsf{fp}}$ p-CLIQUE.

PROOF OF THE CLAIM. Consider an instance $\langle \sigma, \varphi, G \rangle$ of p-MC(Pos$^{\neq}\Sigma_1$, ColBipartite); we are going to construct an instance $\langle G', k \rangle$ of p-CLIQUE.

The input sentence φ can be put into the logically equivalent prenex disjunctive normal form

$$\psi \stackrel{\text{def}}{=} \exists x_1 \dots x_k . \bigvee_{i \in I} \bigwedge_{j \in J_i} \alpha_j(\bar{x}_j) \tag{4.1}$$

where each α_j is an atomic formula that occurred inside φ and is of one of the forms $x_{i_1} = x_{i_2}$, $x_{i_1} \neq x_{i_2}$, $E(x_{i_1}, x_{i_2})$, or $U(x_{i_1})$ for some $x_{i_1}, x_{i_2} \in \{x_1, \dots, x_k\}$ and $U^{(1)} \in \sigma$. Observe that k only depends linearly on the size of the original formula φ, and so does $|J_i|$ for each i, but $|I|$ itself might be exponential in $|\varphi|$ in the worst case.

For each $i \in I$, we construct a graph G'_i that depends on $G = (V, E)$ and $\bigwedge_{j \in J_i} \alpha_j$. Its vertex set is $V \times \{1, \dots, k\}$, and it has an undirected edge $\{(v_1, i_1), (v_2, i_2)\}$ for some

$v_1, v_2 \in V$ and $1 \leq i_1 < i_2 \leq k$ if and only if, for all $j \in J_i$ such that $\bar{x}_j \subseteq \{i_1, i_2\}$, $G, [v_1/x_{i_1}, v_2/x_{i_2}] \models \alpha_j(\bar{x}_j)$. Each G'_i is of size polynomial in k and $|G|$, and thus in $|\varphi|$ and $|G|$.

Then, if there exists a tuple $v_1 \cdots v_k \in V^k$ such that $G, [v_1/x_1, \ldots, v_k/x_k] \models \bigwedge_{j \in J_i} \alpha_j$, then this entails that for each $1 \leq i_1 < i_2 \leq k$, there is an edge $\{(v_{i_1}, i_1), (v_{i_2}, i_2)\}$ in G'_i, and therefore G'_i contains a k-clique. Conversely, if G'_i contains a k-clique on some set of k vertices $\{(v_1, i_1), \ldots, (v_k, i_k)\}$, then $G, [v_1/x_1, \ldots, v_k/x_k] \models \bigwedge_{j \in J_i} \alpha_j(\bar{x}_j)$. See Figure 4.2 for a depiction of a graph G'_i on an example.

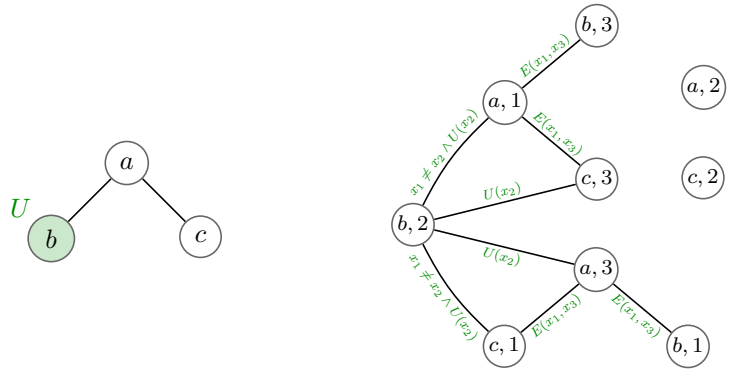


FIGURE 4.2. A coloured graph G (left), and the graph G' (right) constructed in the proof of Claim 4.5.1 for the $\mathsf{Pos}^{\neq}\Sigma_1$ sentence $\exists x_1 x_2 x_3 . x_1 \neq x_2 \wedge E(x_1, x_3) \wedge U(x_2)$. The conjunctions of atoms justifying the presence of each edge in G' are indicated in green next to the edge. The graph G' contains two 3-cliques, corresponding to the valuations $[a/x_1, b/x_2, c/x_3]$ and $[c/x_1, b/x_2, a/x_3]$.

Finally, ψ is logically equivalent to the sentence $\bigvee_{i \in I} \left(\exists x_1 \ldots x_k . \bigwedge_{j \in J_i} \alpha_j(\bar{x}_j)\right)$, and defining G' as the disjoint union of the G'_i, we have that $G \models \varphi$ if and only if G' contains a k-clique. This shows that we have defined a many-one reduction.

To conclude the proof, observe that $|G'|$ is in $2^{|\varphi|} \cdot \mathsf{poly}(|\varphi|, |G|)$, and as already mentioned k is linear in $|\varphi|$. Thus this is indeed a fixed-parameter reduction. ■

Claim 4.5.2. There is a polynomial-time fixed-parameter many-one reduction $\mathsf{p\text{-}CLIQUE} \leq_m^{\mathsf{fpp}} \mathsf{p\text{-}MC}(\mathsf{CQ}, \mathsf{Graph})$.

PROOF OF THE CLAIM. Consider for this an instance $\langle G, k \rangle$ of p-CLIQUE. We work with the graph signature $\sigma = \{E^{(2)}\}$, keep the graph G unchanged, and construct the sentence

$$\mathsf{diag}^+(K_k) = \exists x_1 \cdots x_k . \bigwedge_{1 \leq i < j \leq k} E(x_i, x_j) \,. \tag{4.2}$$

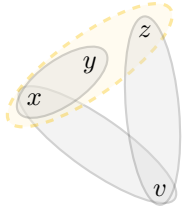
Then G contains a k-clique K_k if and only if there is a homomorphism $K_k \to G$, which is if and only if $G \models \mathsf{diag}^+(K_k)$ by the Homomorphism Theorem. Furthermore $\mathsf{diag}^+(K_k)$ is of quadratic size in k. ■

To conclude and show membership in $\mathsf{W}[1]$, $\mathsf{p\text{-}MC}(\mathsf{CQ}, \mathsf{Graph}) \subseteq \mathsf{p\text{-}MC}(\Sigma_1, \mathsf{Fin})$ and the latter is in $\mathsf{A}[1] = \mathsf{W}[1]$ by Theorem 3.25. □

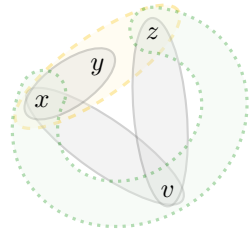
4.2. Acyclic Conjunctive Queries

Theorems 4.4 and 4.5 show that the model-checking problem remains hard for conjunctive queries. We are now going to further restrict conjunctive queries in order to (at last!) obtain tractability, with a fragment that has been thoroughly studied in database theory.

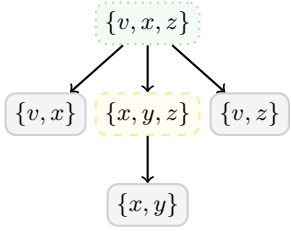
4.2.1. Hypergraphs and Join Trees. A *hypergraph* is a pair (V, E) of a set V of vertices and a set $E \subseteq 2^V$ of *hyper-edges*, each one being a subset of V. Each Boolean conjunctive query $\varphi = \exists \bar{x}.R_1(\bar{x}_1) \wedge \cdots \wedge R_n(\bar{x}_n)$ defines a hypergraph $H(\varphi)$ with $\bar{x}$ as set of vertices and $\{\bar{x}_i \mid 1 \leq i \leq n\}$ as set of hyper-edges. See figures 4.3a and 4.3b for two examples of hypergraphs associated with Boolean conjunctive queries.



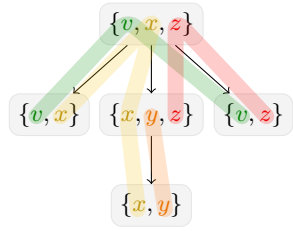
(A) The hypergraph of the conjunctive query $\exists vxyz.R(x, y, z) \wedge S(y, x) \wedge S(x, v) \wedge S(v, z)$.



(B) The hypergraph of the conjunctive query $\exists vxyz.R(x, y, z) \wedge S(y, x) \wedge S(x, v) \wedge S(v, z) \wedge T(x, v, z)$.



(C) A join tree for the hypergraph of Figure 4.3b.



(D) The sets $\{e \in E \mid v \in e\}$ in the join tree of Figure 4.3c.

FIGURE 4.3. Hypergraphs and join trees of conjunctive queries.

A *join tree* for a hypergraph $H = (V, E)$ is a (directed) forest T with E as set of nodes such that, for all $v \in V$, the set $\{e \in E \mid v \in e\}$ of nodes of T that contain v is connected in T. See Figure 4.3c for an example of a join tree.

A hypergraph H is *acyclic* (or more precisely α-*acyclic*) if it has a join tree, and a conjunctive query φ is then *acyclic* if $H(\varphi)$ is acyclic. Let ACQ denote the set of acyclic

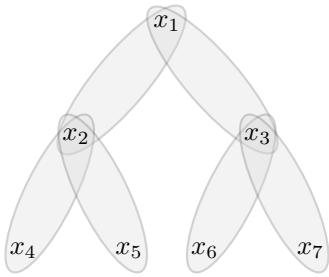
conjunctive queries. For instance, the hypergraph of Figure 4.3a is not acyclic, but perhaps counter-intuitively, the hypergraph of Figure 4.3b with one additional hyper-edge is acyclic.

The first step in order to exploit this new fragment of first-order logic is to be able to decide membership. This can be checked in linear time, and even better, a join tree can be constructed in linear time for acyclic conjunctive queries.

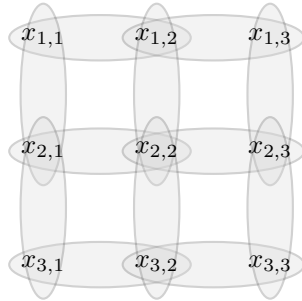
Fact 4.6 (Tarjan and Yannakakis, 1984, Section 3)**.** *Given a finite hypergraph H, we can test whether it is acyclic and compute a join forest for it in linear time.*

> *Exercise 4.3* (GYO Algorithm). The definition we just gave of acyclicity is one of several equivalent characterisations (see Abiteboul, Hull, and Vianu, 1995, Theorem 6.4.5). Among those characterisations, one gives rise to an algorithm for checking whether a hypergraph is acyclic. Show that a hypergraph $H = (V, E)$ is acyclic if and only if all its vertices can be deleted by repeatedly applying the two operations
>
> - delete a vertex that appears in at most one hyperedge (such a vertex is called an *ear*), or
> - delete a hyperedge that is contained into another; this is called a *reduction*.
>
> For instance, on the hypergraph of Figure 4.3a, we can reduce and remove the hyperedge $\{x, y\}$ and then no operation can be applied. On the hypergraph of Figure 4.3b, we can reduce to remove $\{x, y\}$, $\{v, x\}$, and $\{v, z\}$, then y is an ear and is removed, then $\{x, z\}$ can be reduced, and we are left with $\{v, x, z\}$, whose vertices are all ears and can be removed. What is the complexity of this algorithm?

4.2.2. Yannakakis' Algorithm. The join tree of an acyclic Boolean conjunctive query φ essentially provides an evaluation plan for computing $[\![e_\varphi]\!]^{\mathfrak{A}}$ for the relational expression of Theorem 4.1 on a structure $\mathfrak{A}$. The idea is to perform this evaluation inductively, starting from the leaves of the join tree and working our way up to the roots. The end result is an algorithm in $O(\|\mathfrak{A}\| \cdot \log \|\mathfrak{A}\| \cdot |\varphi|)$, showing that $\mathsf{MC}(\mathsf{ACQ}, \mathsf{Fin}) \in \mathsf{P}$ in combined complexity (and thus $\mathsf{p\text{-}MC}(\mathsf{ACQ}, \mathsf{Fin}) \in \mathsf{FPT}$ in parameterised complexity).



(A) The hypergraph of the conjunctive query $\mathsf{diag}^+(T_{2,2}) = \exists x_1 x_2 x_2 x_4 x_5 x_6 x_7 . E(x_1, x_2) \wedge E(x_1, x_3) \wedge E(x_2, x_4) \wedge E(x_2, x_5) \wedge E(x_3, x_6) \wedge E(x_3, x_7)$.



(B) The hypergraph of the conjunctive query $\mathsf{diag}^+(G_{3,3}) = \exists x_{1,1} \cdots x_{3,3} . \big(\bigwedge_{1 \le i < 3} \bigwedge_{1 \le j < 3} E(x_{i,j}, x_{i+1,j}) \wedge E(x_{i,j}, x_{i,j+1}) \big) \wedge \big(\bigwedge_{1 \le i < 3} E(x_{i,3}, x_{i+1,3}) \wedge E(x_{3,i}, x_{3,i+1}) \big)$.

FIGURE 4.4. Two extreme cases of hypergraphs: a tree and a grid.

Example 4.7 (Trees and Grids)**.** Before proving this result, let us build some intuition about why acyclicity is helpful. Consider the two hypergraphs of Figure 4.4. The first one is acyclic and the second cyclic. Recall that the main source of complexity in Theorem 3.14 was the *width* of the formula; here the conjunctive queries have width seven and nine. But the first one in Figure 4.4a can equivalently be written as a primitive positive sentence

$$\exists x_1.\big(\exists x_2.E(x_1,x_2) \wedge (\exists x_4.E(x_2,x_4)) \wedge (\exists x_5.E(x_2,x_5))\big)$$
$$\wedge \big(\exists x3.E(x_1,x_3) \wedge (\exists x_6.E(x_3,x_6)) \wedge (\exists x_7.E(x_3,x_7))\big)$$

of width two—which is the arity of our signature—, and this generalises to $\mathsf{diag}^+(T)$ for any tree T. Conversely, the formula for the query of Figure 4.4b is equivalent to a query of width four, and in general $\mathsf{diag}^+(G_{n,n})$ requires width $(n+1)$ for a $n \times n$ grid.

THEOREM 4.8 (Yannakakis). $\mathsf{MC}(\mathsf{ACQ},\mathsf{Fin})$ *is in deterministic time* $O(\|\mathfrak{A}\| \cdot \log \|\mathfrak{A}\| \cdot |\varphi|)$.

PROOF. Consider for this an acyclic Boolean conjunctive query $\varphi = \exists \bar{x}.R_1(\bar{x}_1) \wedge \cdots \wedge R_n(\bar{x}_n)$ and a finite relational structure $\mathfrak{A}$. By Fact 4.6 we can compute a join tree T for the hypergraph $H(\varphi)$ in time linear in $|\varphi|$.

Local and Descendant Expressions. We are going to associate two relational expressions to every node $\bar{y} \subseteq \bar{x}$ of the join tree. Its *local expression* is

$$\mathsf{local}(\bar{y}) \stackrel{\text{def}}{=} R_{i_1}(\bar{y}) \bowtie \cdots \bowtie R_{i_m}(\bar{y}) \tag{4.3}$$

where the $R_{i_j}(\bar{y})$ are all the relational atoms of φ with $\bar{y}$ as set of free variables. Since φ is a Boolean conjunctive query, ultimately we want to compute the join of the local expressions and project away all the variables, i.e.,

$$\varphi(\mathfrak{A}) = \pi_\emptyset\left(\bigbowtie_{\bar{y}\in T} \mathsf{local}(\bar{y})\right). \tag{4.4}$$

Now here is how the join tree is exploited. As hinted at in Example 4.7, rather than projecting away all the variables once at the end, we are going to project them progressively as we go up the tree. Indeed, a variable appearing in a node $\bar{z}$ but not in its parent $\bar{y}$ will never be useful anymore, not for any ancestor of $\bar{y}$ (nor any sibling of those ancestors, nor any descendants of those siblings outside the sub-tree rooted by $\bar{y}$). Hence we define the *descendant expression* of $\bar{y}$ as the join of the local expressions in its sub-tree, followed by a projection to only keep the variables in $\bar{y}$:

$$\mathsf{descendant}(\bar{y}) \stackrel{\text{def}}{=} \pi_{\bar{y}}\left(\bigbowtie_{\bar{z}\geq_T \bar{y}} \mathsf{local}(\bar{z})\right) \tag{4.5}$$

where $\bar{z} \geq_T \bar{y}$ means that $\bar{z}$ is a descendant of $\bar{y}$ in the join tree T ($\bar{y}$ included). The interest of projecting those variables along the way is that it allows to 'contain the width:' the relations we will compute at each node of the join tree will have few variables. More precisely, for all nodes $\bar{y}$, there will be a relational atom $R_i(\bar{y})$ of φ with the same variable set such that $[\![\mathsf{descendant}(\bar{y})]\!]^{\mathfrak{A}} \subseteq [\![R_i(\bar{y})]\!]^{\mathfrak{A}}$. In particular, the size of the encoding of $[\![\mathsf{descendant}(\bar{y})]\!]^{\mathfrak{A}}$ will be bounded by $\|\mathfrak{A}\|$.

Thus, when we reach the top, $\mathfrak{A} \models \varphi$ if and only if $[\![\mathsf{descendant}(\bar{y})]\!]^{\mathfrak{A}} \neq \emptyset$ for all the roots $\bar{y}$ of T. It remains to see how to compute $[\![\mathsf{local}(\bar{y})]\!]^{\mathfrak{A}}$ and $[\![\mathsf{descendant}(\bar{y})]\!]^{\mathfrak{A}}$.

Claim 4.8.1. $[\![\mathsf{local}(\bar{y})]\!]^{\mathfrak{A}}$ can be computed in time $O(\|\mathfrak{A}\| \cdot \log \|\mathfrak{A}\| \cdot |\mathsf{local}(\bar{y})|)$.

PROOF OF THE CLAIM. Let $\mathsf{local}(\bar{y}) \stackrel{\text{def}}{=} R_{i_1}(\bar{y}) \bowtie \cdots \bowtie R_{i_m}(\bar{y})$ and fix an arbitrary linear ordering of $\bar{y}$.

First, each $[\![R_{i_j}(\bar{y})]\!]^{\mathfrak{A}} \subseteq A^{\bar{y}}$ can be computed and sorted with respect to the lexicographic ordering defined by the ordering of $\bar{y}$ in time $O(\|R_{i_j}^{\mathfrak{A}}\| \cdot \log \|R_{i_j}^{\mathfrak{A}}\|)$. Beware here that $|\bar{y}| \leq \mathsf{ar}(R_{i_j})$: we may need to keep only the tuples in $R_{i_j}^{\mathfrak{A}}$ that satisfy some equality constraints implicit in our notations.

Second, since all these atomic relations share the same set $\bar{y}$ of variables, it suffices to intersect the results:

$$[\![\mathsf{local}(\bar{y})]\!]^{\mathfrak{A}} = \bigcap_{1 \leq j \leq m} [\![R_{i_j}(\bar{y})]\!]^{\mathfrak{A}} . \tag{4.6}$$

As the elements of each $[\![R_{i_j}(\bar{y})]\!]^{\mathfrak{A}}$ are all sorted using the same ordering, this can be done in time $O(\sum_{1 \leq j \leq m} \|[\![R_{i_j}(\bar{y})]\!]^{\mathfrak{A}}\|)$. ■

Semijoins. Let us introduce an additional notation before turning to the complexity of computing $[\![\mathsf{descendant}(\bar{y})]\!]^{\mathfrak{A}}$. For e, e' two relational expression, their *semijoin* $e \ltimes e'$ is defined by $\mathsf{free}(e \ltimes e') \stackrel{\text{def}}{=} \mathsf{free}(e)$ and

$$\begin{aligned} [\![e \ltimes e']\!]^{\mathfrak{A}} &\stackrel{\text{def}}{=} \{\bar{a} \in [\![e]\!]^{\mathfrak{A}} \mid \exists \bar{b} \in [\![e']\!]^{\mathfrak{A}} \,.\, \bar{b}_{\restriction \mathsf{free}(e) \cap \mathsf{free}(e')} = \bar{a}_{\restriction \mathsf{free}(e) \cap \mathsf{free}(e')}\} \qquad \text{(semijoin)} \\ &= [\![\pi_{\mathsf{free}(e)}(e \bowtie e')]\!]^{\mathfrak{A}} . \end{aligned}$$

While this looks quite similar to a join operator, it has interesting algorithmic properties. For instance, if $\mathsf{free}(e) \cap \mathsf{free}(e') = \emptyset$, $[\![e \bowtie e']\!]^{\mathfrak{A}} = [\![e]\!]^{\mathfrak{A}} \times [\![e']\!]^{\mathfrak{A}}$ is the full Cartesian product, but $[\![e \ltimes e']\!]^{\mathfrak{A}} = [\![e]\!]^{\mathfrak{A}}$, and in general $[\![e \ltimes e']\!]^{\mathfrak{A}} \subseteq [\![e]\!]^{\mathfrak{A}}$.

Claim 4.8.2. Given $[\![e]\!]^{\mathfrak{A}}$ and $[\![e']\!]^{\mathfrak{A}}$ with representation sizes n and n', $[\![e \ltimes e']\!]^{\mathfrak{A}}$ can be computed in time $O(n \log n + n' \log n')$.

PROOF OF THE CLAIM. Let $\bar{z} \stackrel{\text{def}}{=} \mathsf{free}(e) \cap \mathsf{free}(e')$ be the set of common variables. Choose two linear orderings of $\mathsf{free}(e)$ and $\mathsf{free}(e')$ such that the common variables in $\bar{z}$ come first, then those of $\mathsf{free}(e) \setminus \bar{z}$ and $\mathsf{free}(e') \setminus \bar{z}$.

We first sort $[\![e]\!]^{\mathfrak{A}}$ and $[\![e']\!]^{\mathfrak{A}}$ with respect to the lexicographic ordering for these two orderings of the variables in time $O(n \log n + n' \log n')$.

Then we only need to advance in the two sorted sets $[\![e]\!]^{\mathfrak{A}}$ and $[\![e']\!]^{\mathfrak{A}}$ in lockstep to find those tuples $\bar{a} \in [\![e]\!]^{\mathfrak{A}}$ such that there exists $\bar{b} \in [\![e']\!]^{\mathfrak{A}}$ with $\bar{a}_{\restriction \bar{z}} = \bar{b}_{\restriction \bar{z}}$, this in time $O(n + n')$. ■

Claim 4.8.3. $[\![\mathsf{descendant}(\bar{y})]\!]^{\mathfrak{A}}$ can be computed in time $O(\|\mathfrak{A}\| \cdot \log \|\mathfrak{A}\| \cdot |\mathsf{descendant}(\bar{y})|)$.

PROOF OF THE CLAIM. We prove the claim by induction on the depth of $\bar{y}$ in T. For the base case, $\bar{y}$ is a leaf thus $\mathsf{local}(\bar{y}) = \mathsf{descendant}(\bar{y})$ and Claim 4.8.1 yields the desired result. For the induction step, let $\bar{y}_1, \ldots, \bar{y}_m$ be the immediate children of $\bar{y}$ in the join tree T. Then

$$[\![\mathsf{descendant}(\bar{y})]\!]^{\mathfrak{A}} = \bigcap_{1 \leq j \leq m} [\![\mathsf{local}(\bar{y}) \ltimes \mathsf{descendant}(\bar{y_j})]\!]^{\mathfrak{A}} . \tag{4.7}$$

By Claim 4.8.1 again, $[\![\mathsf{local}(\bar{y})]\!]^{\mathfrak{A}}$ can be computed in time $O(\|\mathfrak{A}\| \cdot \log \|\mathfrak{A}\| \cdot |\mathsf{local}(\bar{y})|)$, and by induction hypothesis, each $[\![\mathsf{descendant}(\bar{y_j})]\!]^{\mathfrak{A}}$ can also be computed in time $O(\|\mathfrak{A}\| \cdot \log \|\mathfrak{A}\| \cdot |\mathsf{descendant}(\bar{y}_j)|)$. Moreover, the representations of $[\![\mathsf{local}(\bar{y})]\!]^{\mathfrak{A}}$ and $[\![\mathsf{descendant}(\bar{y}_j)]\!]^{\mathfrak{A}}$ are of size at most $\|\mathfrak{A}\|$, thus Claim 4.8.2 shows that each $[\![\mathsf{local}(\bar{y}) \ltimes \mathsf{descendant}(\bar{y_j})]\!]^{\mathfrak{A}}$ can be computed within the desired time bound.

Finally, their intersection can be computed in time $O(m \cdot \|\mathfrak{A}\|)$. ■

This concludes the proof of Theorem 4.8: Yannakakis' algorithm consists in computing $[\![\mathsf{descendant}(\bar{y})]\!]^{\mathfrak{A}}$ for each root node $\bar{y}$ using Claim 4.8.3, and then checks whether they are all non-empty. □

Further Reading

References. Conjunctive queries were defined by Chandra and Merlin (1977), who also identified the corresponding SPJ fragment of relational algebra from Theorem 4.1, and proved the Homomorphism Theorem and the NP-completeness of the model-checking problem in Theorem 4.4.

Parameterised Complexity. Regarding parameterised complexity, the original definition of $\mathsf{W}[1]$ was provided in terms of *weighted* satisfiability problems (hence the name). With respect to that definition, the $\mathsf{W}[1]$-completeness of p-CLIQUE was first shown by Downey and Fellows (1995b, Corollary 3.2), that of p-SHORT-NTM by Cai et al. (1997, Theorem 1), and that of $\mathsf{p\text{-}MC}(\mathsf{CQ}, \mathsf{Fin})$ by Papadimitriou and Yannakakis (1999, Theorem 1). The presentation here follows (Flum and Grohe, 2006, Chapter 6).

Acyclic Conjunctive Queries. The class of acyclic conjunctive queries has emerged in the early 1980's; see (Fagin, 1983) for a comparison of various notions of acyclicity at the time—several of which defining the same notion of α*-acyclicity* used in these notes. Yannakakis's algorithm was published in (Yannakakis, 1981); another polynomial-time algorithm for acyclic conjunctive queries is the *consistency algorithm* (Beeri et al., 1983). See (Arenas et al., 2022, chapters 18–20) and (Abiteboul, Hull, and Vianu, 1995, Section 6.4) for further discussion of acyclicity. Maybe one aspect of acyclic queries that I have not expanded upon further explains their relevance: it is often possible to enforce acyclicity in the *database schema* itself (and if you design a database schema, you should do so), so that any conjunctive query over the schema will be acyclic.

Precise Complexity. The polynomial-time algorithm of Yannakakis for $\mathsf{MC}(\mathsf{ACQ}, \mathsf{Fin})$ is not the most precise complexity statement for this problem. Gottlob, Leone, and Scarcello (2001) indeed show that this problem is LOGCFL-complete under logspace reductions, a complexity class that we can fit into the hierarchy of uniform circuit complexity classes of (3.4)

$$\mathsf{AC}^0 \subsetneq \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{LOGCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \cdots \subseteq \mathsf{P}\,. \tag{4.8}$$

Notably, the polynomial-time GYO algorithm of exercise 4.3 for computing join trees also has a LOGCFL counterpart.

Evaluation of Acyclic Conjunctive Queries. Yannakakis's algorithm can be extended to answer the $\mathsf{EVAL}(\mathsf{ACQ}, \mathsf{Fin})$ problem, in *total polynomial time* $\mathsf{poly}(\|\mathfrak{A}\|, \varphi, \|\varphi(\mathfrak{A})\|)$ where we take the size of the output $\varphi(\mathfrak{A})$ into account.

From the enumeration complexity viewpoint mentioned at the end of Chapter 3, such a complexity is *not* considered tractable, and additional restrictions are needed (Durand, 2020, Section 4.1); here we do not have the full answer and this is an active subject of research (see Carmeli and Segoufin, 2023).

Beyond Acyclicity. The class of acyclic conjunctive queries is not the most general class, for which model-checking is tractable. Marx (2013, Theorem 1.4) shows that, assuming the exponential time hypothesis, $\mathsf{MC}(\mathsf{L}, \mathsf{Fin})$ for $\mathsf{L} \subseteq \mathsf{CQ}$ is in FPT if and only if L has 'bounded submodular width.' We will see a generalisation of acyclic conjunctive queries (but less general than queries of bounded submodular width) with a tractable model-checking problem in Section 6.3.

Constraint Satisfaction. The *constraint satisfaction problem* (CSP), in its uniform version, is the following generalisation of the model-checking problem to finite structures on general signatures, i.e., not only relational ones.

> Problem (CSP).
> **instance:** a finite signature σ, a finite structure $\mathfrak{A} \in \mathsf{Fin}[\sigma]$, and a Boolean conjunctive query $\varphi \in \mathsf{CQ}[\sigma]$
> **question:** $\mathfrak{A} \models \varphi$?

As functions can be encoded as relations, this problem is equivalent to $\mathsf{MC}(\mathsf{CQ}, \mathsf{Fin})$, and by the Homomorphism Theorem, to $\mathsf{HOM}(\mathsf{Fin}, \mathsf{Fin})$ and $\mathsf{CONTAINMENT}(\mathsf{CQ})$, which are all NP-complete (see, e.g., Kolaitis and Vardi, 2000).

Nevertheless, a very rich theory has emerged from the study of the *non-uniform* version $\mathsf{CSP}(\mathfrak{A})$ of the problem, where we fix the structure $\mathfrak{A}$ (called the *template* in the context of CSPs)—this corresponds to the data complexity of the model-checking problem for conjunctive queries.

The goal of this research program was to classify the complexity of $\mathsf{CSP}(\mathfrak{A})$ depending on $\mathfrak{A}$. We have seen in Theorem 4.4 that $\mathsf{CSP}(C_3)$ was NP-complete, but other structures can be far easier: Hell and Nešetřil (1990) showed that $\mathsf{CSP}(G)$ for a graph G is in P if G is bipartite and is NP-complete otherwise. What is remarkable here is that Ladner's Theorem says that if $\mathsf{P} \neq \mathsf{NP}$ then there exist NP-*intermediate* problems that are neither in P nor NP-complete, but here this *dichotomy* result shows that there are no NP-intermediate $\mathsf{CSP}(G)$ problems. Feder and Vardi (1993) famously conjectured that this dichotomy generalised to all $\mathsf{CSP}(\mathfrak{A})$ problems, and not solely graphs: they are either in P or NP-complete, but not NP-intermediate. The CSP dichotomy conjecture was solved and answered positively independently by Bulatov and Zhuk in 2017, who characterise exactly the templates $\mathfrak{A}$ that lead to CSPs either in P or NP-complete, drawing heavily from universal algebra techniques.

This is not the end of the story for CSPs: in their non-uniform version, it makes sense to investigate $\mathsf{CSP}(\mathfrak{A})$ for infinite templates $\mathfrak{A}$ as well. While there are infinite CSPs of arbitrary complexity, this area of research concentrates again on the border between P and NP, focusing its attention to finitely bounded homogeneous structures—where it applies techniques from model theory, e.g., $\aleph_0$-categoricity and Fraïssé limits. See for instance the works of Bodirsky (2008) for a survey or Mottet, Nagy, and Pinsker (2024) for recent results on this topic.

CHAPTER 5

The Case of Trees

So far, our investigation of the model-checking problem over finite structures has often shown that the problem was intractable—even for graphs, and even for conjunctive queries. The only exception at this point is Yannakakis' Algorithm in Section 4.2.2, which restricts the logical formulæ to be acyclic conjunctive queries, i.e., to have an associated join tree.

We now turn our attention to restricting the class $\mathscr{C}$ of structures in order to find more islands of tractability. Our first example of a tractable class of structures are *trees*. This somehow captures our impression from Example 4.7 that trees are somewhat 'easy.' We are indeed going to see that the model-checking problem for trees t is fixed-parameter linear time tractable in $O(f(|\varphi|) \cdot |t|)$ (see Corollary 5.23)—though it can be debatable whether 'easy' is really the right word, given the complexity of the function f involved.

The algorithm achieving this complexity goes through the construction of *tree automata*; here we start in Section 5.1.2 with a perhaps less well-known model of tree automata, that matches exactly the kind of trees we typically encounter in model-checking problems: unordered tree automata. Thanks to the effective closure properties of these automata (see Section 5.1.3), we show in Section 5.3 how to construct tree automata accepting exactly the tree models of sentences in an extension of first-order logic, namely monadic second-order logic, whose definition is recalled in Section 5.2.

These positive results come with a caveat: the complexity of the construction is *not elementary recursive* (see Section 5.3.1 for the definition), and moreover this is inherent to the problems at hand, as discussed in Section 5.4.

Finally, there are several ways of defining the notion of 'tree' besides the labelled directed trees of the upcoming Section 5.1.1, and **??** surveys the main variants and their relationships.

5.1. Tree Automata

5.1.1. Labelled Directed Trees. We have already used trees in Section 4.2 but have not really defined them. What we mean by a 'tree' is really a particular case of directed graphs: a (directed) *tree* is a pair $T = (V, E)$ where V is a non-empty set of *nodes* and $E \subseteq V^2$ the *child* relation, where

- every node has at most one *parent*: for all $v \in V$, there is at most one $v' \in V$ such that $(v', v) \in E$, and then $v \neq v'$—in other words, the parent relation E^{-1} is an irreflexive partial function—, and
- a node with no parent is a *root*, and a tree has exactly one root.

If we remove the second constraint, what we have is a (directed) *forest*.

We typically like our trees to be *labelled* by some finite alphabet Σ, i.e., to work with pairs (T, ℓ) where $\ell \colon V(T) \to \Sigma$ is a labelling function. See Figure 5.1 for an example of a tree, where we indicate the labels directly inside the nodes.

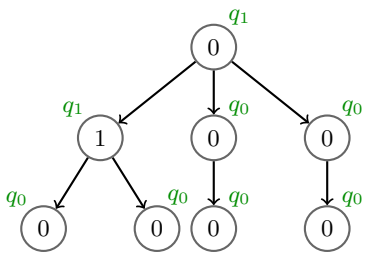


FIGURE 5.1. A directed tree over the alphabet $\{0, 1\}$, and an associated run of the automaton of Example 5.3 in green.

5.1.1.1. *Trees as Finite Structures.* In terms of structures, a labelled directed tree (T, ℓ) is seen as a structure $T = (V, E, (a)_{a\in\Sigma})$ on the signature $\{E\} \cup \Sigma \stackrel{\text{def}}{=} \{E^{(2)}\} \cup \{a^{(1)} \mid a \in \Sigma\}$ with a unary relation symbol $a^{(1)}$ for each $a \in \Sigma$, such that every node $v \in V$ has *exactly* one label from Σ, i.e., where the interpretation of each unary symbol $a^{(1)}$ in T is $a \stackrel{\text{def}}{=} \{v \in V \mid \ell(v) = a\}$. Let us define the formulæ

$$\mathit{child}(x, y) \stackrel{\text{def}}{=} E(x, y) \qquad\qquad \mathit{root}(x) \stackrel{\text{def}}{=} \forall y.\neg \mathit{child}(y, x)\ .$$

Then, the class of Σ-labelled directed trees is the set $\mathsf{Mod}_{\mathsf{Fin}}(T)$ of finite models over the signature $\{E^{(2)}\} \cup \{a^{(1)} \mid a \in \Sigma\}$ of a theory T with axioms

$$\forall x.\neg \mathit{child}(x, x) \tag{5.1}$$

$$\forall x.\forall y.(\mathit{child}(y, x) \to \forall z.(\mathit{child}(z, x) \to y = z) \tag{5.2}$$

$$\forall x. \bigvee_{a\in\Sigma} \left(a(x) \wedge \bigwedge_{b\in\Sigma\setminus\{a\}} \neg b(x) \right) \tag{5.3}$$

$$\exists x.\mathit{root}(x) \wedge (\forall z.\mathit{root}(z) \to z = x) \tag{5.4}$$

We will write $\mathsf{Tree}(\Sigma)$ for this class of structures, and Tree for their union over all finite alphabets Σ.

5.1.1.2. *Trees as Unordered, Unranked Terms.* Another, more convenient way of defining trees is as an inductive datatype. First define a (finite) *multiset* over a set S as a function $m \colon S \to \mathbb{N}$ with finite *support*, meaning that $\mathsf{Supp}(m) \stackrel{\text{def}}{=} \{s \in S \mid m(s) > 0\}$ is finite. A multiset can also be written by listing its elements in an arbitrary order, for instance

$m = \{\!\{a, b, a, c\}\!\}$ is the multiset with support $\{a, b, c\}$ such that $m(a) = 2$, $m(b) = 1$, and $m(c) = 1$. We write $\mathbb{M}(S)$ for the set of finite multisets over S.

The set $\mathsf{Tree}(\Sigma)$ of trees over Σ is defined inductively as the smallest set such that, if $\{\!\{t_1, \ldots, t_n\}\!\}$ is a (possibly empty) multiset of trees, then $a\{\!\{t_1, \ldots, t_n\}\!\}$ is a tree in $\mathsf{Tree}(\Sigma)$ rooted by new a node labelled by a and having the roots of the disjoint union of $t_1, \ldots, t_n$ as children. A tree $a\{\!\{\,\}\!\}$ with an empty multiset of children is called a *leaf* and we will more simply write a to denote it.

What we have provided here is in fact an *algebraic definition* of the set of trees (seen as finite structures) from a set of (ground) *unordered, unranked terms* over Σ, defined by the abstract syntax

$$t ::= a\{\!\{t, \ldots, t\}\!\} \qquad \text{(unordered unranked terms)}$$

where a ranges over Σ: the operation in this algebra simply builds a disjoint union of its arguments and adds a new a-labelled root node connected to the previous roots.

Example 5.1 (Directed tree)**.** Figure 5.1 displays the tree over the alphabet $\{a, b\}$ defined by the term $a\{\!\{a\{\!\{a\}\!\}, b\{\!\{a, a\}\!\}, a\{\!\{a\}\!\}\}\!\}$.

Right now this is pretty transparent, as the term and its associated tree are essentially the same object—in fact we will not make a distinction between a labelled tree (T, ℓ) seen as a labelled directed graph, the same tree $T = (V, E, (a)_{a\in\Sigma})$ seen as a structure, and the corresponding tree t seen as an unordered unranked term—, but later in these notes we will have more complex algebraic operations for defining more complex structures from terms.

5.1.2. Unordered Tree Automata.

Tree automata provide an operational (and also an algebraic) way of accepting trees, leading to decision algorithms. We first start with an abstract definition allowing infinite automata, which we will instantiate later in Section 5.1.2.1.

Definition 5.2 (Unordered Tree Automaton)**.** An *unordered tree automaton* over an alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, \delta, F)$ where Q is a finite set of states, $\delta \subseteq \Sigma \times \mathbb{M}(Q) \times Q$ a (possibly infinite) set of transitions, and $F \subseteq Q$ a set of accepting states.

Runs. A *run* of $\mathcal{A}$ on a tree $t = (V, E, (a)_{a\in\Sigma})$ is a relabelling $\rho = (V, E, (q)_{q\in Q})$ of the tree using the alphabet Q instead of Σ—thus with the same nodes and child relation—that satisfies the following local constraint:

> for every node $a\{\!\{t_1, \ldots, t_n\}\!\}$ of t, the corresponding node of ρ has a label $q \in Q$ compatible with the transition relation: if the roots of $t_1, \ldots, t_n$ are respectively labelled $q_1, \ldots, q_n$, then $(a, \{\!\{q_1, \ldots, q_n\}\!\}, q) \in \delta$.

This corresponds to a process that relabels the tree bottom-up, from the leaves to the root. We write accordingly $a\{\!\{q_1, \ldots, q_n\}\!\} \to_{\mathcal{A}} q$ instead of $(a, \{\!\{q_1, \ldots, q_n\}\!\}, q) \in \delta$.

In our proofs, it will be convenient to manipulate the set of all possible root labels of runs. Let $\mathcal{A}(t)$ be the set of states $q \in Q$ such that there exists a run of $\mathcal{A}$ on t with root label q. This is amenable to an inductive definition of the set of root labels:

$$\mathcal{A}(a\{\!\{t_1, \ldots, t_n\}\!\}) = \{q \in Q \mid \exists q_1 \in \mathcal{A}(t_1), \ldots, \exists q_n \in \mathcal{A}(t_n) \,.\, a\{\!\{q_1, \ldots, q_n\}\!\} \to_{\mathcal{A}} q\} \,. \tag{5.5}$$

Languages. For a state $q \in Q$, its *language* $L_{\mathcal{A}}(q)$ is the set of trees in $\mathsf{Tree}(\Sigma)$ for which there exists a run of $\mathcal{A}$, whose root is labelled by q. In other words,

$$L_{\mathcal{A}}(q) \stackrel{\text{def}}{=} \{t \in \mathsf{Tree}(\Sigma) \mid q \in \mathcal{A}(t)\} \,. \tag{5.6}$$

The *language* $L(\mathcal{A})$ accepted by the automaton is then

$$L(\mathcal{A}) \stackrel{\text{def}}{=} \bigcup_{q \in F} L_{\mathcal{A}}(q) = \{t \in \mathsf{Tree}(\Sigma) \mid \mathcal{A}(t) \cap F \neq \emptyset\} \,, \tag{5.7}$$

i.e., it is the set of trees for which there exists a run whose root is labelled by some accepting state in F.

Example 5.3 (Unordered tree automaton)**.** Let us consider the unordered tree automaton $\mathcal{A}$ over $\Sigma \stackrel{\text{def}}{=} \{0,1\}$ with states $\{q_0, q_1, q_\perp\}$, set of final states $F \stackrel{\text{def}}{=} \{q_1\}$, and transitions

$$\begin{array}{llll}
0(m) \to_{\mathcal{A}} q_0 & \text{if } m(q_1) = 0 & \text{and} & m(q_\perp) = 0 \,; \\
1(m) \to_{\mathcal{A}} q_1 & \text{if } m(q_1) = 0 & \text{and} & m(q_\perp) = 0 \,; \\
0(m) \to_{\mathcal{A}} q_1 & \text{if } m(q_1) = 1 & \text{and} & m(q_\perp) = 0 \,; \\
0(m) \to_{\mathcal{A}} q_\perp & \text{if } m(q_1) \geq 2 & \text{or} & m(q_\perp) \geq 1 \,; \\
1(m) \to_{\mathcal{A}} q_\perp & \text{if } m(q_1) \geq 1 & \text{or} & m(q_\perp) \geq 1 \,.
\end{array}$$

See Figure 5.1 for an example of a run for this automaton. Then $L_{\mathcal{A}}(q_0)$ is the set of trees with only 0 labels, $L(\mathcal{A}) = L_{\mathcal{A}}(q_1)$ the set of trees with exactly one node labelled by 1, and $L_{\mathcal{A}}(q_\perp)$ is the set of trees with more than one node labelled by 1.

Deterministic and Complete Automata. The automata in Definition 5.2 are possibly *non-deterministic* and allow in general multiple runs on the same tree. An unordered tree automaton $\mathcal{A} = (Q, \Sigma, \delta, F)$ is *deterministic* if, for all $a \in \Sigma$, $m \in \mathbb{M}(Q)$, and $q, q' \in Q$, $a(m) \to_{\mathcal{A}} q$ and $a(m) \to_{\mathcal{A}} q'$ imply $q = q'$. In other words, in a deterministic automaton, δ represents a partial function $\Sigma \times \mathbb{M}(Q) \hookrightarrow Q$, and thus $\mathcal{A} \colon \mathsf{Tree}(\Sigma) \hookrightarrow Q$ is a partial function.

An automaton is *complete* if, for all $a \in \Sigma$ and $m \in \mathbb{M}(Q)$, there exists $q \in Q$ such that $a(m) \to_{\mathcal{A}} q$. Thus, in a complete deterministic automaton, δ represents a total function $\Sigma \times \mathbb{M}(Q) \to Q$; this entails that, for any tree in $\mathsf{Tree}(\Sigma)$, there is exactly one run of the automaton and a single possible state at the root, given by the function $\mathcal{A} \colon \mathsf{Tree}(\Sigma) \to Q$. Observe that the automaton of Example 5.3 is complete deterministic.

5.1.2.1. *One-Step Languages.* The issue so far with Definition 5.2 is that our unordered tree automata are *not finite*: they may have an infinite set of transitions. What we need is a finite representation for potentially infinite sets of multisets in $\mathbb{M}(Q)$. Put abstractly, we want a 'one-step language,' namely a countable set $\mathsf{C}[Q]$ of constraints χ along with a satisfaction relation $\Vdash_1$ allowing to define when a multiset over Q satisfies a constraint, denoted by $m \Vdash_1 \chi$.

The notion of one-step language leads to a generic construction of finite automata representing potentially infinite unordered tree automata. Let us define a C*-tree automaton* over an alphabet Σ as a tuple $\mathcal{A} = (Q, \Sigma, \delta, F)$ where Q is a finite set of states, $\delta \subseteq \Sigma \times \mathsf{C}[Q] \times Q$ a *finite* set of transitions, and $F \subseteq Q$ a set of accepting states. Such a C-tree automaton defines a (potentially infinite) unordered tree automaton $\mathcal{A}' \stackrel{\text{def}}{=} (Q, \Sigma, \delta', F)$ with transitions

$a(m) \to_{\mathcal{A}'} q$ for all $a \in \Sigma, m \in \mathbb{M}(Q)$, and $q \in Q$ such that there is a transition $(a, \chi, q) \in \delta$ with $m \Vdash_1 \chi$.

There is a rich diversity of one-step languages used in the literature to define classes of unordered tree automata; here is a non-exhaustive list of examples:

threshold tree automata: a constraint χ is a pair $(m', c) \in \mathbb{M}(Q) \times \mathbb{N}$, and $m \Vdash_1 (m', c)$ if, for all $q \in Q$, $m'(q) = m(q)$, or $m'(q) = c$ and $m(q) \geq c$;

Presburger tree automata: a constraint χ is a formula of linear arithmetic $\mathsf{FO}[+]$ with free variables $x_1, \ldots, x_{|Q|}$, and $m \Vdash_1 \chi$ if $\mathbb{N} \models \chi(m(q_1), \ldots, m(q_{|Q|}))$;

first-order tree automata: a constraint χ is a sentence of the first-order logic $\mathsf{FO}[Q]$ over the monadic signature $\{q^{(1)} \mid q \in Q\}$; a multiset $m = \{\!\{q_1, \ldots, q_n\}\!\} \in \mathbb{M}(Q)$ can be seen as a structure $\mathfrak{M} \stackrel{\text{def}}{=} (\{1, \ldots, n\}, (q^{\mathfrak{M}})_{q \in Q})$ with $q^{\mathfrak{M}} \stackrel{\text{def}}{=} \{i \mid q_i = q\}$ for all $q \in Q$, and we let $m \Vdash_1 \chi$ if $\mathfrak{M} \models \chi$;

first-order modulo counting tree automata: the same as the previous one, but our one-step constraints are $\mathsf{FO}{+}\mathsf{MOD}[Q]$ sentences that can additionally use *modulo quantifiers* $\exists^{r[p]}$ for $r < p \in \mathbb{N}$, with semantics $\mathfrak{A}, \bar{a} \models \exists^{r[p]} x.\varphi$ if the number $|\{b \in A \mid \mathfrak{A}, \bar{a}[b/x] \models \varphi\}|$ of choices of elements $b \in A$ that satisfy φ is congruent to r modulo p.

These different one-step languages yield different models of automata, that strike different balances between expressiveness, complexity of the associated membership and emptiness decision problems, and ease of use in proofs. These notes will develop the first example of one-step languages: the class of threshold tree automata, starting next in Section 5.1.2.2; but see also exercise 5.1 for the case of Presburger tree automata where we restrict ourselves to quantifier-free constraints.

5.1.2.2. *Threshold Tree Automata.* Let us introduce some additional notation. Given $c \in \mathbb{N}$, we will write $\mathbb{M}_c(S)$ for the set of finite multisets over S with multiplicities at most c, i.e., $m \in \mathbb{M}_c(S)$ if and only if $m \in \mathbb{M}(S)$ and $m(s) \leq c$ for all $s \in S$. For $m \in \mathbb{M}(S)$, $\mathsf{cap}_c(m) \in \mathbb{M}_c(S)$ is the multiset with multiplicities 'capped' at c, i.e., $\mathsf{cap}_c(m)(s) \stackrel{\text{def}}{=} \min(c, m(s))$. We say that two multisets $m, m' \in \mathbb{M}(S)$ are *c-equivalent*, denoted by $m \equiv_c m'$, if $\mathsf{cap}_c(m) = \mathsf{cap}_c(m')$ and write $[m]_c = \{m' \mid \mathsf{cap}_c(m') = m\}$ for the equivalence class of m. Note that, if $m \equiv_c m'$ and $c > 0$, then $\mathsf{Supp}(m) = \mathsf{Supp}(m')$.

Definition 5.4 (Threshold Tree Automaton)**.** A *threshold tree automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, F, c)$ where $c > 0$ and $\delta \subseteq \Sigma \times \mathbb{M}_c(Q) \times Q$ is now a finite set of transitions. Each transition $(a, m, q) \in \delta$ defines the (potentially infinite) set of transitions steps $a(m') \to_{\mathcal{A}} q$ for all $m' \in [m]_c$.

Example 5.5 (Threshold tree automaton)**.** Let us see how to define the unordered tree automaton of Example 5.3 using a threshold tree automaton. We use the threshold $c \stackrel{\text{def}}{=} 2$ and list all the cases in $\mathbb{M}_2(\{q_0, q_1, q_\perp\})$ explicitly: first the cases without $q_\perp$

$$(0, \{\!\{\}\!\}, q_0) \quad (0, \{\!\{q_0\}\!\}, q_0) \quad (0, \{\!\{q_0, q_0\}\!\}, q_0)$$
$$(1, \{\!\{\}\!\}, q_1) \quad (1, \{\!\{q_0\}\!\}, q_1) \quad (1, \{\!\{q_0, q_0\}\!\}, q_1)$$
$$(0, \{\!\{q_1\}\!\}, q_1) \quad (0, \{\!\{q_0, q_1\}\!\}, q_1) \quad (0, \{\!\{q_0, q_0, q_1\}\!\}, q_1)$$
$$(0, \{\!\{q_1, q_1\}\!\}, q_\perp) \quad (0, \{\!\{q_0, q_1, q_1\}\!\}, q_\perp) \quad (0, \{\!\{q_0, q_0, q_1, q_1\}\!\}, q_\perp)$$

$(1,\{\!\{q_1\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_1\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_0,q_1\}\!\},q_\perp)$
$(1,\{\!\{q_1,q_1\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_1,q_1\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_0,q_1,q_1\}\!\},q_\perp)$;

then all the cases where $q_\perp$ appears once

$(0,\{\!\{q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_0,q_\perp\}\!\},q_\perp)$
$(1,\{\!\{q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_0,q_\perp\}\!\},q_\perp)$
$(0,\{\!\{q_1,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_1,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_0,q_1,q_\perp\}\!\},q_\perp)$
$(0,\{\!\{q_1,q_1,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_1,q_1,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_0,q_1,q_1,q_\perp\}\!\},q_\perp)$
$(1,\{\!\{q_1,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_1,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_0,q_1,q_\perp\}\!\},q_\perp)$
$(1,\{\!\{q_1,q_1,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_1,q_1,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_0,q_1,q_1,q_\perp\}\!\},q_\perp)$;

and finally all the cases where $q_\perp$ appears twice

$(0,\{\!\{q_\perp,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_\perp,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_0,q_\perp,q_\perp\}\!\},q_\perp)$
$(1,\{\!\{q_\perp,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_\perp,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_0,q_\perp,q_\perp\}\!\},q_\perp)$
$(0,\{\!\{q_1,q_\perp,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_1,q_\perp,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_0,q_1,q_\perp,q_\perp\}\!\},q_\perp)$
$(0,\{\!\{q_1,q_1,q_\perp,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_1,q_1,q_\perp,q_\perp\}\!\},q_\perp)$ $(0,\{\!\{q_0,q_0,q_1,q_1,q_\perp,q_\perp\}\!\},q_\perp)$
$(1,\{\!\{q_1,q_\perp,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_1,q_\perp,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_0,q_1,q_\perp,q_\perp\}\!\},q_\perp)$
$(1,\{\!\{q_1,q_1,q_\perp,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_1,q_1,q_\perp,q_\perp\}\!\},q_\perp)$ $(1,\{\!\{q_0,q_0,q_1,q_1,q_\perp,q_\perp\}\!\},q_\perp)$.

5.1.3. Closure Properties. Threshold tree automata have all the expected closure properties: the class of tree languages they define is closed under Boolean operations and relabelling. For complexity considerations, we define the *one norm* of a multiset as the sum of its multiplicities: $\|m\|_1 \stackrel{\text{def}}{=} \sum_{s\in S} m(s)$. Then we define the *size* $\|\mathcal{A}\|$ of a threshold tree automaton $\mathcal{A} = (Q,\Sigma,\delta,F,c)$ as $\sum_{(a,m,q)\in\delta}(\|m\|_1+1)$, i.e., we encode multiplicities in unary.

5.1.3.1. *Determinisation.* As a preamble to the rest of this section, let us consider how to construct complete deterministic threshold tree automata from non-deterministic ones.

By definition, a threshold tree automaton $\mathcal{A}$ is *deterministic* if, for all $a\in\Sigma$, $m,m'\in\mathbb{M}_c(Q)$, and $q,q'\in Q$, $(a,m,q)\in\delta$ and (a,m',q') with $[m]_c\cap[m']_c\neq\emptyset$ together imply $q=q'$. However, $[m]_c\cap[m']_c\neq\emptyset$ only occurs when $m=m'$ since m and m' have maximal multiplicity at most c. Thus the condition for determinism can more simply be stated by requiring that δ is a partial function of type $\Sigma\times\mathbb{M}_c(Q)\hookrightarrow Q$.

Similarly, a threshold tree automaton is *complete* if, for all $a\in\Sigma$ and $m'\in\mathbb{M}(Q)$, there exists $m\in\mathbb{M}_c(Q)$ and $q\in Q$ such that $(a,m,q)\in\delta$ and $m'\equiv_c m$, which is if and only if for all $a\in\Sigma$ and $m\in\mathbb{M}_c(Q)$, there exists $q\in Q$ such that $(a,m,q)\in\delta$. Thus the automaton is complete deterministic if and only if δ is a total function of type $\Sigma\times\mathbb{M}_c(Q)\to Q$. We can see that complete deterministic threshold tree automata requires to spell out all the $|\Sigma|\cdot(c+1)^{|Q|}$ possible transitions, for a total size $\|\mathcal{A}\|$ bounded by $c\cdot|Q|\cdot|\Sigma|\cdot(c+1)^{|Q|}$; this is not a concise formalism, as already see in Example 5.5.

The following proposition shows that threshold tree automata can be determinised, although with a worst-case double exponential blow-up.

Proposition 5.6 (Determinisation of threshold tree automata)**.** *Let $\mathcal{A}$ be a threshold tree automaton over Σ with N states and threshold c. Then we can construct an equivalent complete deterministic threshold tree automaton $\det(\mathcal{A})$ with 2^N states and threshold $c \cdot N$ such that $L(\mathcal{A}) = L(\det(A))$, thus of size in $O(|\Sigma|(c \cdot N + 1)^{2^N})$.*

Proof. We adapt the usual powerset construction for automata. Consider an threshold tree automaton $\mathcal{A} = (Q, \Sigma, \delta, F, c)$. We define $\det(\mathcal{A}) \stackrel{\text{def}}{=} (2^Q, \Sigma, \delta_{\det}, F_{\det}, c_{\det})$ where

$$F_{\det} \stackrel{\text{def}}{=} \{p \in 2^Q \mid p \cap F \neq \emptyset\} \tag{5.8}$$

and such that $\delta_{\det}$ represents the set of transitions

$$a\{\!|p_1, \ldots, p_n|\!\} \to_{\det(\mathcal{A})} \{q \in Q \mid \exists q_1 \in p_1, \ldots, \exists q_n \in p_n \,.\, a\{\!|q_1, \ldots, q_n|\!\} \to_{\mathcal{A}} q\} \tag{5.9}$$

for all letters $a \in \Sigma$ and finite multisets $\{\!|p_1, \ldots, p_n|\!\} \in \mathbb{M}(2^Q)$. Then $\det(\mathcal{A})$ will be complete deterministic by definition, and by the inductive definition of root labels in (5.5), $\mathcal{A}_{\det}(t) = \mathcal{A}(t)$ for all trees $t \in \mathsf{Tree}(\Sigma)$, and therefore by (5.7–5.8), $L(\mathcal{A}_{\det}) = L(\mathcal{A})$ as desired.

It remains to see how to represent (5.9) using multisets in $\mathbb{M}_{c_{\det}}(2^Q)$ for the new threshold $c_{\det} \stackrel{\text{def}}{=} c \cdot |Q|$. When $m = \{\!|p_1, \ldots, p_n|\!\} \in \mathbb{M}(2^Q)$ and $m' = \{\!|q_1, \ldots, q_n|\!\} \in \mathbb{M}(Q)$ are such that $q_i \in p_i$ for all $1 \leq i \leq n$, we call m a *lift* of m' and write $m' \in m_{\restriction Q}$. Let us define

$$\begin{aligned}\delta_{\det} \stackrel{\text{def}}{=} \{(a, m, \{q \in Q \mid \exists m' \in m_{\restriction Q} \,.\, (a, \mathsf{cap}_c(m'), q) \in \delta\} \\ \mid a \in \Sigma, m \in \mathbb{M}_{c \cdot |Q|}(2^Q)\} \,.\end{aligned} \tag{5.10}$$

Consider a transition step $a(m) \to p$ according to (5.9) and the corresponding transition $(a, \mathsf{cap}_{c \cdot |Q|}(m), p') \in \delta_{\det}$ according to (5.10). The following two claims show that $p = p'$, thereby proving the correctness of our definition of $\delta_{\det}$. □

Claim 5.6.1. *$p \subseteq p'$.*

Proof of the claim. If $q \in p$, then by (5.9) there exists $m_q \in m_{\restriction Q}$ with a transition $(a, \mathsf{cap}_c(m_q), q) \in \delta$. Write $m = \{\!|p_1, \ldots, p_s|\!\}$ and $m_q = \{\!|q_1, \ldots, q_s|\!\}$ with $q_i \in p_i$ for all $1 \leq i \leq s$ such that $\mathsf{cap}_c(m_q) = \{\!|q_1, \ldots, q_r|\!\}$ for some $r \leq s$.

Observe that $r \leq c \cdot |Q|$ since $\mathsf{cap}_c(m_q) \in \mathbb{M}_c(Q)$. Therefore, up to a re-indexing of the elements p_i and q_i of m and m_q for $r < i \leq s$, we can write $\mathsf{cap}_{c \cdot |Q|}(m) = \{\!|p_1, \ldots, p_n|\!\}$ for some $r \leq n \leq s$. Then $m' \stackrel{\text{def}}{=} \{\!|q_1, \ldots, q_n|\!\}$ is in $(\mathsf{cap}_{c \cdot |Q|}(m))_{\restriction Q}$ and such that $\mathsf{cap}_c(m') = \mathsf{cap}_c(m_q) = \{\!|q_1, \ldots, q_r|\!\}$, thus $(a, \mathsf{cap}_c(m'), q) \in \delta$. By (5.10) this shows $q \in p'$. ■

Claim 5.6.2. *$p' \subseteq p$.*

Proof of the claim. If $q \in p'$, then by (5.10) there exists $m' \in (\mathsf{cap}_{c \cdot |Q|}(m))_{\restriction Q}$ such that $(a, \mathsf{cap}_c(m'), q) \in \delta$. Write $m = \{\!|p_1, \ldots, p_s|\!\}$ such that $\mathsf{cap}_{c \cdot |Q|}(m) = \{\!|p_1, \ldots, p_n|\!\}$ for some $n \leq s$. With these notations, note that $m(p_k) \geq c \cdot |Q|$ for each $n + 1 \leq k \leq s$. Also write $m' = \{\!|q_1, \ldots, q_n|\!\}$ with $q_i \in p_i$ for all $1 \leq i \leq n$.

By the Pigeonhole Principle, for each $1 \leq i \leq n$ such that $\mathsf{cap}_{c \cdot |Q|}(m)(p_i) = c \cdot |Q|$, there is an index $1 \leq j \leq n$ such that $p_i = p_j$ and $m'(q_j) \geq c$. Thus, for each $n+1 \leq k \leq s$ we can find $q'_k \in p_k$ such that $\mathsf{cap}_c(m')(q'_k) = c$. Define $m_q \stackrel{\text{def}}{=} \{\!|q_1, \ldots, q_n, q'_{n+1}, \ldots, q'_s|\!\}$. On the one hand, $\mathsf{cap}_c(m_q) = \mathsf{cap}_c(m')$ and therefore $(a, \mathsf{cap}_c(m_q), q) \in \delta$. On the other hand $m_q \in m_{\restriction Q}$. By (5.9), this shows that $q \in p$. ■

Example 5.7 (Determinisation). Consider the non-deterministic threshold tree automaton with states $Q \stackrel{\text{def}}{=} \{q, q'\}$, alphabet $\Sigma \stackrel{\text{def}}{=} \{a, b\}$, final states $F \stackrel{\text{def}}{=} \{q\}$, threshold $c \stackrel{\text{def}}{=} 2$, and transitions

$$\delta \stackrel{\text{def}}{=} \{(a, \{\!\{\}\!\}, q), (a, \{\!\{\}\!\}, q'), (b, \{\!\{q, q'\}\!\}, q)\} .$$

Its language is the single tree $b\{\!\{a, a\}\!\}$. Its determinisation produces the automaton with threshold 4 and transitions $(a, \{\!\{\}\!\}, \{q, q'\})$ and $(b, \{\!\{\{q, q'\}, \{q, q'\}\}\!\}, \{q\})$ still allowing to accept the tree $b\{\!\{a, a\}\!\}$, plus many other useless transitions like $(b, \{\!\{\{q\}, \{q'\}\}\!\}, \{q\})$, $(b, \{\!\{\{q, q'\}, \{q, q'\}, \{q, q'\}\}\!\}, \emptyset)$, or $(b, \{\!\{\emptyset, \emptyset, \emptyset, \emptyset\}\!\}, \emptyset)$.

5.1.3.2. *Complementation.* Thanks to Proposition 5.6, it is straightforward to complement threshold tree automata as usual in automata theory, by first determinising them if needed.

Proposition 5.8 (Complementation of threshold tree automata)**.** *Let $\mathcal{A}$ be a complete deterministic threshold tree automaton over Σ with N states and threshold c. Then we can construct a complete deterministic threshold tree automaton $\mathcal{A}'$ with N states and threshold c such that $L(\mathcal{A}') = \mathsf{Tree}(\Sigma) \setminus L(\mathcal{A})$.*

Proof. It suffices to invert the acceptance set: if $\mathcal{A} = (Q, \Sigma, \delta, F)$, then we let $\mathcal{A}' \stackrel{\text{def}}{=} (Q, \Sigma, \delta, Q \setminus F)$. Indeed, a tree t is in $L(\mathcal{A})$ if and only if $\mathcal{A}(t) \in F$, thus t is in $\mathsf{Tree}(\Sigma) \setminus L(\mathcal{A})$ if and only if $\mathcal{A}(t) \in Q \setminus F$. □

5.1.3.3. *Intersection.* The usual construction from automata theory is straightforward to adapt to the case of threshold tree automata.

Proposition 5.9 (Intersection of threshold tree automata)**.** *Let $\mathcal{A}$ and $\mathcal{A}'$ be two threshold tree automata over Σ with N and N' states respectively and the same threshold c. Then we can construct a threshold tree automaton $\mathcal{A}_\cap$ over Σ with threshold c and $N \cdot N'$ states such that $L(\mathcal{A}_\cap) = L(\mathcal{A}) \cap L(\mathcal{A}')$. Furthermore, if $\mathcal{A}$ and $\mathcal{A}'$ were complete deterministic, then so is $\mathcal{A}_\cap$.*

Proof. This is the usual synchronisation product construction from automata theory. Let $\mathcal{A} = (Q, \Sigma, \delta, F, c)$ and $\mathcal{A}' = (Q', \Sigma, \delta', F', c)$; we construct $\mathcal{A}_\cap \stackrel{\text{def}}{=} (Q \times Q, \Sigma, \delta_\times, F \times F', c)$ where

$$\delta_\times \stackrel{\text{def}}{=} \{(a, m, (q, q')) \mid (a, m, q) \in \delta \text{ and } (a, m, q') \in \delta'\} . \qquad \square$$

The same construction also handles unions if $\mathcal{A}$ and $\mathcal{A}'$ are complete, by using $(F \times Q') \cup (Q \times F')$ as acceptance set.

5.1.3.4. *Projection.* The final closure property we will use later is closure under *relabellings*, i.e., functions $\Sigma \to \Sigma'$. Given a tree $t \in \mathsf{Tree}(\Sigma)$ and a relabelling $f : \Sigma \to \Sigma'$, we let $f(t)$ be the tree in $\mathsf{Tree}(\Sigma')$ obtained by relabelling every a-labelled node by $f(a)$, this for all $a \in \Sigma$. Formally,

$$f(a\{\!\{t_1, \ldots, t_n\}\!\}) \stackrel{\text{def}}{=} f(a)\{\!\{f(t_1), \ldots, f(t_n)\}\!\} .$$

For a tree language $L \subseteq \mathsf{Tree}(\Sigma)$, $f(L) \stackrel{\text{def}}{=} \{f(t) \mid t \in L\}$. A particular case of a relabelling is a *projection* $\Sigma \times \Sigma' \to \Sigma$ that maps $(a, b) \mapsto a$ for all $a \in \Sigma$ and $b \in \Sigma'$.

Proposition 5.10 (Relabellings of threshold tree automata)**.** *Let $\mathcal{A}$ be a threshold tree automata over Σ with N states and threshold c, and $f\colon \Sigma \to \Sigma'$ be a relabelling. Then we can construct a threshold tree automaton $\mathcal{A}_f$ over Σ' with N states and threshold c such that $L(\mathcal{A}_f) = f(L(\mathcal{A}))$.*

PROOF. Let $\mathcal{A} = (Q, \Sigma, \delta, F, c)$; we construct $\mathcal{A}_f \stackrel{\text{def}}{=} (Q, \Sigma, \delta_f, F, c)$ where

$$\delta_f \stackrel{\text{def}}{=} \{(f(a), m, q) \mid (a, m, q) \in \delta\} \ . \qquad \square$$

Note that this construction does *not* preserve determinism if f is not injective (which is typically not the case of projections).

Exercise 5.1 (Presburger tree automata). We sketched in Section 5.1.2.1 a model of Presburger tree automata. We now make the model more precise and work for this with the quantifier-free fragment of first-order logic $\mathsf{QF}[<^{(2)}, (\equiv_p)_{p\in\mathbb{Z}}, +^{(2)}, 0^{(0)}, 1^{(0)}]$, henceforth called the quantifier-free fragment of *Presburger arithmetic* and denoted by QFPA, which features binary relational symbols $<$ and $\equiv_p$ for every $p \in \mathbb{Z}$, a binary function symbol $+$, and two constant symbols 0 and 1. We fix the interpretation to work over the structure $(\mathbb{Z}, <, (\equiv_p)_{p\in\mathbb{Z}}, +, 0, 1)$ where '$a \equiv_p b$' stands for '$a \equiv b \bmod p$' and the other symbols have their natural interpretation; we write simply $\mathbb{Z} \models \varphi$ rather than $(\mathbb{Z}, <, (\equiv_p)_{p\in\mathbb{Z}}, +, 0, 1) \models \varphi$ if φ is a Presburger formula.
Rather than working with cumbersome additive terms like $1+1+\cdots+1+x+x+\cdots+x$, we can assume our atoms to be of form $a_1x_1 + \cdots + a_nx_n \leq b$ or $a_1x_1 + \cdots + a_nx_n \equiv_p b$ where $x_1, \ldots, x_n$ are variables and $a_1, \ldots, a_n, b \in \mathbb{Z}$ are constants, and we may encode these constants in binary.

Definition 5.11 (Presburger Tree Automaton)**.** A *Presburger tree automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, F)$ where $\delta \subseteq \Sigma \times \mathsf{QFPA} \times Q$ is a *finite* set of transitions of the form (a, χ, q) where χ is a quantifier-free Presburger formula with $\mathsf{free}(\chi) = Q$. Then $\mathcal{A}$ defines an unordered tree automaton with transitions $a(m) \to_{\mathcal{A}} q$ for all $m \in \mathbb{N}^Q$ such that $\mathbb{Z} \models \chi(m)$.

(1) Give a Presburger tree automaton that defines the unordered tree automaton of Example 5.3.
(2) Use the following fact to show that Presburger tree automata can be determinised and completed, with a worst-case double exponential blow-up in size.

    Fact 5.12 (Haase et al., 2024, Corollary 3.2)**.** *Given a formula $\exists\bar{y}.\varphi(\bar{x}, \bar{y})$ where φ is a quantifier-free Presburger formula, we can compute in exponential time an equivalent quantifier-free formula $\psi(\bar{x})$, i.e., for all $\bar{a} \in \mathbb{Z}^{\bar{x}}$, $\mathbb{Z} \models \psi(\bar{a})$ if and only if there exists $\bar{b} \in \mathbb{Z}^{\bar{y}}$ such that $\mathbb{Z} \models \varphi(\bar{a}, \bar{b})$. (Moreover, the formula ψ has size exponential in φ even if its constants are encoded in unary.)*

(3) Show that Presburger tree automata languages are closed under complement, intersection, and projection.

5.1.4. Decision Problems. There are two natural decision problems for tree automata: membership and emptiness. Here, we are going to see that threshold tree automata behave well with respect to both problems.

5.1.4.1. *Membership.* The (uniform) *membership problem* for a family T of tree automata over a class of tree structures $\mathscr{T}$ is the following.

> Problem (MEMBERSHIP($\mathsf{T}, \mathscr{T}$)).
> **instance:** a tree $t \in \mathscr{T}$ and an automaton $\mathcal{A} \in \mathsf{T}$
> **question:** $t \in L(\mathcal{A})$?

Proposition 5.13 (Membership for threshold tree automata)**.** *The membership problem for deterministic threshold tree automata over directed trees is in time $O(\|\mathcal{A}\| + |t|)$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, F, c)$. We process the tree t bottom-up and compute $\mathcal{A}(t') \in Q$ by induction on the subtrees t' of t using (5.5). We initially load $\mathcal{A}$ in time $O(\|\mathcal{A}\|)$ and use a perfect hash function (see, e.g., Belazzougui, Botelho, and Dietzfelbinger, 2009) for each $a \in \Sigma$ for retrieving the partial function $\delta_a \colon m \mapsto q$ for every $m \in \mathbb{M}_c(Q)$ and $q \in Q$ such that $(a, m, q) \in \delta$. Then, for a subtree $a\{\!\{t_1, \dots, t_n\}\!\}$ where we have computed $m \stackrel{\text{def}}{=} \{\!\{\mathcal{A}(t_1), \dots, \mathcal{A}(t_n)\}\!\} \in \mathbb{M}(Q)$ by induction hypothesis, we obtain $\delta_a(\mathsf{cap}_c(m))$ in constant time. Thus the overall complexity is in $O(\|\mathcal{A}\| + |t|)$. □

> *Exercise 5.2* (Membership for Presburger tree automata). What is the complexity of the membership problem for the deterministic Presburger tree automata of exercise 5.1 over directed trees?

5.1.4.2. *Emptiness.* The *non-emptiness problem* for a family T of tree automata over a class of tree structures $\mathscr{T}$ is the following.

> Problem (NON-EMPTINESS($\mathsf{T}, \mathscr{T}$)).
> **instance:** an automaton $\mathcal{A} \in \mathsf{T}$
> **question:** does there exist $t \in \mathscr{T}$ such that $t \in L(\mathcal{A})$?

Proposition 5.14 (Non-emptiness for threshold tree automata)**.** *The non-emptiness problem for threshold tree automata over directed trees is in time $O(\|\mathcal{A}\|)$ and* P*-complete.*

Proof. We only sketch the upper bound; see exercise 5.3 for the lower bound. Given a threshold tree automaton $\mathcal{A} = (Q, \Sigma, \delta, F, c)$, we construct a set Φ of Horn clauses with a positive literal in each clause, of size $\|\Phi\| \leq \|\mathcal{A}\|$. The set of propositions of Φ is Q. For each transition (a, m, q), Φ contains the clause $q \leftarrow \bigwedge_{q' \in \mathsf{Supp}(m)} q'$ (in particular, this is the clause $q \leftarrow \top$ if m is the empty multiset). Then $\Phi \models q$ if and only if $L_{\mathcal{A}}(q) \neq \emptyset$, and the set $\{q \in Q \mid \Phi \models q\}$ can be computed in deterministic time $O(\|\Phi\|)$ (see, e.g., Dowling and Gallier, 1984). □

> *Exercise 5.3* (P-hardness of non-emptiness). Show the P-hardness of the non-emptiness problem for threshold tree automata over directed trees. *Hint: as in exercise 3.3, you may use the* P*-completeness of* CIRCUIT-EVAL *for the lower bound (see, e.g., Arora and Barak, 2009, Section 7.3).*

5.2. Monadic Second-Order Logic on Trees

On the logical side, we are going to show a stronger result than the tractability of first-order model-checking over directed trees: we are going to see that monadic second-order logic is also tractable.

5.2.1. Monadic Second-Order Logic. Monadic second-order logic (MSO) is an extension of first-order logic that allows quantification over sets of elements in structures. Equivalently, this is the fragment of second-order logic where we only allow quantification over relations of arity one.

5.2.1.1. *Syntax.* Let us fix $\mathcal{X}_2$ an infinite countable set of second-order variables, all treated as unary symbols. Over a signature $\sigma = \mathcal{P} \uplus \mathcal{F}$, the set of *monadic second-order formulæ* are defined inductively through the abstract syntax

$$\varphi ::= R(t_1, \dots, t_{\mathsf{ar}(R)}) \mid t_1 = t_2 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid X(x) \mid \exists X.\varphi \qquad \text{(MSO formulæ)}$$

where R ranges over relational symbols in $\mathcal{P}$, the t_i's range over first-order terms in $T(\mathcal{F}, \mathcal{X})$, x over first-order variables in $\mathcal{X}$, and X over second-order variables in $\mathcal{X}_2$. We write $\mathsf{MSO}[\sigma]$ for the class of all monadic second-order formulæ over σ, and MSO for their union over all possible signatures. The notion of free variables extends naturally to monadic second-order formulæ and we write $\mathsf{free}_2(\varphi)$ for the set of free monadic second-order variables of φ.

5.2.1.2. *Semantics.* Consider a signature $\sigma = \mathcal{P} \uplus \mathcal{F}$ and a structure $\mathfrak{A}$ over σ. A *second-order valuation* in $\mathfrak{A}$ of a set of second-order variables $\bar{X} \subseteq \mathcal{X}_2$ is a function $\bar{A} \in (2^A)^{\bar{X}}$. We say that $\mathfrak{A}$ along with a first-order valuation $\bar{a} \in A^{\bar{x}}$ and a second-order valuation $\bar{A} \in (2^A)^{\bar{X}}$ *satisfies* a monadic second-order formula $\varphi \in \mathsf{MSO}[\sigma]$ with free first-order variables $\mathsf{free}(\varphi) \subseteq \bar{x}$ and free second-order variables $\mathsf{free}_2(\varphi) \subseteq \bar{X}$, denoted $\mathfrak{A}, \bar{A}, \bar{a} \models \varphi$, in the following inductive cases

$$\begin{aligned}
&\mathfrak{A}, \bar{A}, \bar{a} \models R(t_1, \dots, t_{\mathsf{ar}(R)}) && \text{if } (\llbracket t_1 \rrbracket^{\mathfrak{A}}_{\bar{a}}, \dots, \llbracket t_{\mathsf{ar}(f)} \rrbracket^{\mathfrak{A}}_{\bar{a}}) \in R^{\mathfrak{A}} \ , \\
&\mathfrak{A}, \bar{A}, \bar{a} \models t_1 = t_2 && \text{if } \llbracket t_1 \rrbracket^{\mathfrak{A}}_{\bar{a}} = \llbracket t_1 \rrbracket^{\mathfrak{A}}_{\bar{a}} \ , \\
&\mathfrak{A}, \bar{A}, \bar{a} \models \neg\varphi && \text{if } \mathfrak{A}, \bar{A}, \bar{a} \not\models \neg\varphi \ , \\
&\mathfrak{A}, \bar{A}, \bar{a} \models \varphi \wedge \psi && \text{if } \mathfrak{A}, \bar{A}, \bar{a} \models \varphi \text{ and } \mathfrak{A}, \bar{a} \models \psi \ , \\
&\mathfrak{A}, \bar{A}, \bar{a} \models \exists x.\varphi && \text{if } \exists b \in A \text{ such that } \mathfrak{A}, \bar{A}, \bar{a}[b/x] \models \varphi \ , \\
&\mathfrak{A}, \bar{A}, \bar{a} \models X(x) && \text{if } \bar{a}(x) \in \bar{A}(X) \ , \\
&\mathfrak{A}, \bar{A}, \bar{a} \models \exists X.\varphi && \text{if } \exists U \subseteq A \text{ such that } \mathfrak{A}[U/X], \bar{a} \models \varphi \ .
\end{aligned}$$

Example 5.15 (MSO Formulæ)**.** Consider for instance the tree t of Figure 5.2. Then

$$\mathit{leaf}(x) \stackrel{\text{def}}{=} \neg(\exists y.\mathit{child}(x, y))$$

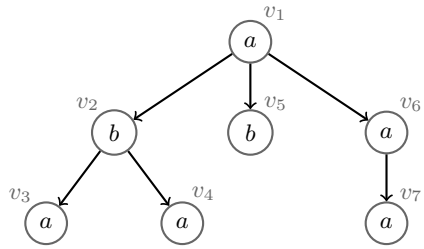


FIGURE 5.2. A tree $t \in \mathsf{Tree}(\{a, b\})$ with node set $V(t) = \{v_1, \dots, v_7\}$.

tells whether a node is a leaf or not: $t \models \mathit{leaf}(v_3)$ but $t \not\models \mathit{leaf}(v_2)$. The following formula selects a set of nodes forming a branch in a tree:

$$\begin{aligned}\mathit{branch}(X) \stackrel{\text{def}}{=} &\Big(\exists xy.\mathit{root}(x) \wedge \mathit{leaf}(y) \wedge X(x) \wedge X(y)\Big)\\ &\wedge \Big(\forall x.X(x) \to \big(\forall y.\mathit{child}(y,x) \to X(y)\big)\Big)\\ &\wedge \Big(\forall xyz.\big(X(x) \wedge \mathit{child}(y,x) \wedge \mathit{child}(y,z)\big) \to \neg X(z)\Big)\ .\end{aligned}$$

For instance, $t \models \mathit{branch}(\{v_1, v_2, v_3\})$ since this set contains the root v_1 and a leaf v_3, is closed under taking parents (the parent v_2 of v_3 is indeed included), and does not allow branching (v_4, v_5, and v_6 are indeed not included). Then

$$\varphi \stackrel{\text{def}}{=} \exists X.\mathit{branch}(X) \wedge \forall x.X(x) \to a(x)$$

is a sentence such that $t \models \varphi$, thanks to the branch $\{v_1, v_6, v_7\}$ where all the nodes are labelled with a.

Exercise 5.4 (Example of a threshold tree automaton). Provide a threshold tree automaton $\mathcal{A}$ that accepts the trees in $\mathsf{Tree}(\{a, b\})$ that have an a-labelled branch from a leaf to the root, i.e., such that $L(\mathcal{A}) = \mathsf{Mod}_{\mathsf{Tree}(\Sigma)}(\varphi)$ where φ is the formula defined in Example 5.15.

5.2.2. From Automata to Existential MSO on Trees. A good way to measure the expressiveness of monadic second order logic is to show that it can define the sets of trees accepted by tree automata.

Here we concentrate on trees $t \in \mathsf{Tree}(\Sigma)$ seen as finite structures $(V, E, (a)_{a\in\Sigma})$. Our goal is to show that the languages of trees in $\mathsf{Tree}(\Sigma)$ accepted by threshold tree automata are the sets of models of MSO sentences. In fact, a fragment of MSO suffices to that end: an *existential MSO formula* is a formula of the form $\exists \bar{X}.\varphi$ where $\bar{X} \subseteq \mathcal{X}_2$ is a finite set of second-order variables and φ is a first-order formula over the signature $\sigma \cup \{X^{(1)} \mid X \in \bar{X}\}$ extended with unary relational symbols for each second-order variable X of $\bar{X}$.

THEOREM 5.16. *Let $L = L(\mathcal{A}) \subseteq \mathsf{Tree}(\Sigma)$ be a language accepted by a threshold tree automaton $\mathcal{A}$ over Σ. Then we can construct an existential MSO sentence φ of size polynomial in $\|\mathcal{A}\|$, such that $L = \mathsf{Mod}_{\mathsf{Tree}(\Sigma)}(\varphi)$.*

PROOF. Consider a threshold tree automaton $\mathcal{A} = (Q, \Sigma, \delta, F, c)$ over Σ and a tree $t \in \mathsf{Tree}(\Sigma)$. We are going to write an existential MSO formula satisfied by t if and only if there is a run of $\mathcal{A}$ over t.

Let $\bar{X}_Q \stackrel{\text{def}}{=} \{X_q \mid q \in Q\}$. Our formula is

$$\varphi \stackrel{\text{def}}{=} \exists \bar{X}_Q . \psi \tag{5.11}$$

where ψ is a first-order formula; if $t \models \varphi$ then there exists a second-order valuation $\bar{A} \in (2^{V(t)})^{\bar{X}}$ that associates a subset $\bar{A}(X_q) \subseteq V(t)$ of the nodes to each state $q \in Q$. We need our formula ψ to ensure three conditions in order to check that this valuation $\bar{A}$ describes an accepting run, namely

(1) the sets $\bar{A}(X_q)$ for $q \in Q$ form a disjoint cover of $V(t)$,
(2) the root of t is in $\bar{A}(X_q)$ for some accepting state $q \in F$, and
(3) locally, at every node of t, the state labels are consistent with the transition relation δ.

We define accordingly

$$\psi \stackrel{\text{def}}{=} \psi_{(1)} \wedge \psi_{(2)} \wedge \psi_{(3)} \tag{5.12}$$

where each $\psi_{(i)}$ enforces condition (i) above. Regarding the disjoint cover condition (1),

$$\psi_{(1)} \stackrel{\text{def}}{=} \left(\forall x. \bigvee_{q \in Q} X_q(x) \right) \wedge \left(\bigwedge_{q \neq q' \in Q} \neg (\exists x. X_q(x) \wedge X_{q'}(x)) \right) . \tag{5.13}$$

Regarding the root label condition (2),

$$\psi_{(2)} \stackrel{\text{def}}{=} \forall x. \mathit{root}(x) \to \bigvee_{q \in F} X_q(x) \ . \tag{5.14}$$

Finally, regarding the local transition condition (3),

$$\psi_{(3)} \stackrel{\text{def}}{=} \forall x. \bigwedge_{a \in \Sigma} \bigwedge_{q \in Q} \left(a(x) \wedge X_q(x) \to \bigvee_{(a,m,q) \in \delta} \mathit{multiset}_{m,c}(\bar{X}_Q, x) \right) \tag{5.15}$$

where, for a multiset $m = \{\!\{ q_1, \ldots, q_n \}\!\}$,

$$\mathit{multiset}_{m,c}(\bar{X}_Q, x) \stackrel{\text{def}}{=} \exists x_1, \ldots, x_n . \left(\bigwedge_{1 \leq i \leq n} \mathit{child}(x, x_i) \wedge X_{q_i}(x_i) \right) \wedge \left(\bigwedge_{1 \leq i < j \leq n} x_i \neq x_j \right)$$
$$\wedge \left(\forall y. \mathit{child}(x, y) \to \left(\bigvee_{1 \leq i \leq n} y = x_i \vee \bigvee_{\substack{1 \leq i \leq n \\ m(q_i) = c}} X_{q_i}(y) \right) \right)$$

checks for the existence of n distinct children rooting the subtrees of the node selected by x, with the correct state labelling, and such that any additional child node is labelled by a state q_i such that $m(q_i) = c$. □

A comment on the significance of Theorem 5.16: existential MSO is a very small extension of first-order logic, in that it only asks for a first-order property for an expansion of the structure with a finite number of additional 'colours,' i.e., a finite number of unary relational symbols.

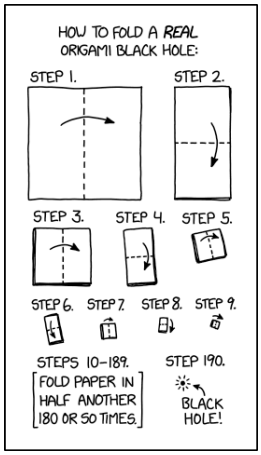


FIGURE 5.3. Origami Black Hole, from `https://xkcd.com/3033`.

5.3. Trees Are Easy!

Our goal in this section is to show that the satisfiability problem for MSO over finite directed trees is satisfiable, and that the parameterised model-checking problem for MSO over finite directed trees is fixed-parameter tractable—thus, that trees are indeed easy.

5.3.1. Caveat on Elementary Complexity. The following Section 5.4 will nuance this assessment. Indeed, the constructions we are about to see have a complexity that is *not* elementary recursive, which is bad news because—in spite of the name—there is not much that was 'elementary' about elementary recursive complexity in the first place.

Exponential Hierarchies. Before we define the notion of elementary recursive functions, let $(\mathsf{exp}_k)_{k\in\mathbb{N}}$ be the family of k-iterated exponential functions defined by

$$\mathsf{exp}^0(x) \stackrel{\text{def}}{=} x\,, \qquad\qquad \mathsf{exp}^{k+1}(x) \stackrel{\text{def}}{=} 2^{\mathsf{exp}^k(x)}\,.$$

In other words, $\mathsf{exp}^k(x) = 2^{2^{\cdots^{2^x}}}\Big\}^{k \text{ times}}$ is a 'tower' of exponentials of fixed height k with x on top. These functions quickly reach values that ridicule estimated physical quantities in our universe; as an illustration consider $\mathsf{exp}^3(3) = 2^{256}$ and Figure 5.3.

You might have encountered the k-iterated exponentiation functions before in the definition of the exponential time and space hierarchies in complexity theory:

$$k\text{-}\mathsf{EXP} \stackrel{\text{def}}{=} \mathsf{DTIME}\big(\mathsf{exp}^k(\mathsf{poly}(n))\big),$$
$$k\text{-}\mathsf{NEXP} \stackrel{\text{def}}{=} \mathsf{NTIME}\big(\mathsf{exp}^k(\mathsf{poly}(n))\big),$$
$$k\text{-}\mathsf{EXPSPACE} \stackrel{\text{def}}{=} \mathsf{NSPACE}\big(\mathsf{exp}^k(\mathsf{poly}(n))\big)$$

are such that $\mathsf{P} = 0\text{-}\mathsf{EXP}$, $\mathsf{NP} = 0\text{-}\mathsf{NEXP}$, and $\mathsf{PSPACE} = 0\text{-}\mathsf{EXPSPACE}$, then $\mathsf{EXP} = 1\text{-}\mathsf{EXP}$, $\mathsf{NEXP} = 1\text{-}\mathsf{NEXP}$, and $\mathsf{EXPSPACE} = 1\text{-}\mathsf{EXPSPACE}$, etc.

(Non-)Elementary Complexity. A function f is *elementary recursive* (aka Kalmár elementary) if there exists some k such that f can be computed in deterministic time $O(\mathsf{exp}^k(n))$, and if so we write $f \in \mathsf{elem}(n)$. The class of *elementary decision problems* (i.e., $\{0,1\}$-valued elementary recursive functions) is then

$$\mathsf{ELEMENTARY} = \bigcup_k k\text{-}\mathsf{EXP} = \bigcup_k k\text{-}\mathsf{NEXP} = \bigcup_k k\text{-}\mathsf{EXPSPACE} \ .$$

A function that is not elementary recursive is *non-elementary*; a prime example of a non-elementary function is the so-called 'tetration' or 'tower' function

$$\mathsf{tower}(x) \stackrel{\text{def}}{=} \mathsf{exp}^x(0)$$

that defines a tower of exponentials, whose height depends (linearly) on the argument. The associated complexity class is

$$\mathsf{TOWER} \stackrel{\text{def}}{=} \mathsf{DTIME}\big(\mathsf{tower}(\mathsf{elem}(n))\big)$$

the class of decision problems that can be decided in time bounded by a tower of exponentials whose height depends elementarily on the input. Here we have defined TOWER in terms of deterministic time-bounded computations, but this is immaterial: we obtain the same class if we look at non-deterministic time bounds or space bounds instead.

The class TOWER plays the role of a 'limit' class for ELEMENTARY, like PSPACE does for PH or AW$[*]$ for $\bigcup_i \mathsf{A}[i]$. Let me stress here that there are **no** 'ELEMENTARY-complete' problems under any reasonable class of reductions; this is because the exponential time and space hierarchies are *strict* by the time and space hierarchy theorems, so that a collapse like the one hypothesised in exercise 3.2.(1) for PH is impossible. By contrast, TOWER has complete problems, and we allow elementary reductions when talking about hardness for TOWER.

5.3.2. From MSO to Automata on Trees.

We are going to prove a converse to Theorem 5.16: given an MSO sentence φ, we can construct a threshold tree automaton $\mathcal{A}_\varphi$ such that $\mathsf{Mod}_{\mathsf{Tree}(\Sigma)}(\varphi) = L(\mathcal{A}_\varphi)$. We do this proof in two steps: first we show how to handle quantifier-free formulæ in Section 5.3.2.1, and then the full logic in Section 5.3.2.2.

5.3.2.1. *Valuation Trees and Quantifier-Free MSO.*

Let φ be a quantifier-free monadic second-order logic formula over $\sigma = \{E\} \cup \Sigma$. Necessarily (as we do not have constant symbols in our signature), φ has some free variables, say $\bar{x} = \mathsf{free}(\varphi)$ and $\bar{X} = \mathsf{free}_2(\varphi)$, which need to be handled somehow.

Alternative Syntax. In our constructions, having to deal with first-order variables and their valuations is a bit cumbersome, and it would be convenient to get rid of first-order variables entirely. This is doable, since a first-order valuation $\bar{a} \in A^{\bar{x}}$ can be seen as a second-order valuation $\bar{A} \in (2^A)^{\bar{x}}$ with the additional property that $\bar{A}(x)$ is a singleton $\{\bar{a}(x)\}$ for each $x \in \bar{x}$, and this can be expressed in MSO. Let

$$\mathit{singleton}(X) \stackrel{\text{def}}{=} \exists x.X(x) \wedge \forall y.X(y) \to x = y$$

be a formula with one free second-order variable X, such that $\mathfrak{A}, \bar{A} \models \mathit{singleton}(X)$ if and only if $|\bar{A}(X)| = 1$, thus allowing to check whether X behaves in fact like a first-order variable. We also need to express the atomic MSO formulæ with second-order variables, and let

$$X \subseteq Y \stackrel{\text{def}}{=} \forall x.X(x) \to Y(x)$$

be a formula with two free second-order variables X and Y, such that $\mathfrak{A}, \bar{A} \models X \subseteq Y$ if and only if $\bar{A}(X) \subseteq \bar{A}(Y)$,

$$R(X_1, \ldots, X_{\mathsf{ar}(R)}) \stackrel{\text{def}}{=} \exists x_1, \ldots, x_{\mathsf{ar}(R)}.R(x_1, \ldots, x_{\mathsf{ar}(R)}) \wedge \bigwedge_{1 \leq i \leq \mathsf{ar}(R)} X_i(x_i)$$

be a formula with $\mathsf{ar}(R)$ free second-order variables $X_1, \ldots, X_{\mathsf{ar}(R)}$, such that $\mathfrak{A}, \bar{A} \models R(\bar{X})$ if and only if there exists $\bar{a} \in R^{\mathfrak{A}}$ such that $\bar{a}_i \in \bar{A}(X_i)$ for all $1 \leq i \leq \mathsf{ar}(R)$.

Example 5.17 (Singletons, inclusions, and existential labels and edges)**.** In the tree t of Figure 5.2, we have $t \models \mathit{singleton}(\{v_1\})$, $t \models \{v_2, v_5, v_6\} \subseteq \{v_2, v_4, v_5, v_6\}$, $t \models a(\{v_2, v_4, v_5, v_6\})$, and $t \models E(\{v_1\}, \{v_2, v_4, v_5, v_6\})$.

Put together, this yields an alternative syntax for MSO formulæ over a relational signature σ

$$\varphi ::= R(X_1, \ldots, X_{\mathsf{ar}(R)}) \mid \mathit{singleton}(X) \mid X_1 \subseteq X_2 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists X.\varphi$$

where R ranges over $\mathcal{P}$ and $X, X_1, \ldots$ over $\mathcal{X}_2$; there are no first-order variables nor any first-order quantification in this syntax.

The definitions of $\mathit{singleton}(X)$, $X \subseteq Y$, and $R(X_1, \ldots, X_{\mathsf{ar}(R)})$ above show that any MSO formula in the alternative syntax is equivalent to an MSO formula in the standard syntax. Conversely, for an MSO formula φ in the standard syntax, assume wlog. that $\bar{X}_{\mathsf{free}(\varphi)} \stackrel{\text{def}}{=} \{X_x \mid x \in \mathsf{free}(\varphi)\} \subseteq \mathcal{X}_2$ is disjoint from $\mathsf{free}_2(\varphi)$. Then φ is equivalent to the MSO formula

$$\varphi^\dagger \wedge \bigwedge_{x \in \mathsf{free}(\varphi)} \mathit{singleton}(X_x) \tag{5.16}$$

over the alternative syntax, where we treat first-order variables x as second-order variables X_x, and define $\varphi^\dagger$ inductively by

$$\begin{aligned}
(R(x_1, \ldots, x_{\mathsf{ar}(R)}))^\dagger &\stackrel{\text{def}}{=} R(X_{x_1}, \ldots, X_{x_{\mathsf{ar}(R)}}) \,, & (x = y)^\dagger &\stackrel{\text{def}}{=} X_x \subseteq X_y \wedge X_y \subseteq X_x \,, \\
(\neg\varphi)^\dagger &\stackrel{\text{def}}{=} \neg(\varphi^\dagger) \,, & (\varphi \wedge \psi)^\dagger &\stackrel{\text{def}}{=} (\varphi^\dagger) \wedge (\psi^\dagger) \,, \\
(\exists x.\varphi)^\dagger &\stackrel{\text{def}}{=} \exists X_x.\mathit{singleton}(X_x) \wedge (\varphi^\dagger) \,, & (X(x))^\dagger &\stackrel{\text{def}}{=} X_x \subseteq X \,, \\
(\exists X.\varphi)^\dagger &\stackrel{\text{def}}{=} \exists X.(\varphi^\dagger) \,.
\end{aligned}$$

Observe that in this translation, we have the guarantee whenever we work with a variable X_x for some $x \in \mathsf{free}(\varphi)$ that the second-order valuation of X_x is in fact a singleton.

Example 5.18 (Alternative syntax)**.** Consider the MSO formula

$$\mathit{children}(x, X) \stackrel{\text{def}}{=} \forall y.E(x, y) \leftrightarrow X(y)$$

such that $t \models \mathit{children}(v, U)$ if and only if U is the set of children of v; in the tree t of Figure 5.2, we have $t \models \mathit{children}(v_1, \{v_2, v_5, v_6\})$. This formula is equivalent to

$$\varphi(X_x, X) \stackrel{\text{def}}{=} \mathit{singleton}(X_x) \wedge \big(\forall X_y.\mathit{singleton}(X_y) \to (E(X_x, X_y) \leftrightarrow X_y \subseteq X)\big) \ ,$$

in that $t \models \varphi(U, U')$ if and only if $U = \{v\}$ for a vertex v and U' is the set of children of v. In the tree t of Figure 5.2, we have accordingly $t \models \varphi(\{v_1\}, \{v_2, v_5, v_6\})$.

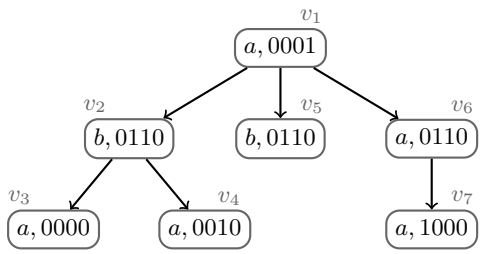
Valuation Trees. Let us assume for a moment that we are working with a formula $\varphi^\dagger$ over the alternative syntax such that $\bar{X} = \mathsf{free}_2(\varphi)^\dagger$. Rather than accepting sets of trees over Σ, the automata $\mathcal{A}$ we are going to construct accept *valuation trees* $t \otimes \bar{A}$, that combine a tree $t = (V, E, (a)_{a\in\Sigma})$ in $\mathsf{Tree}(\Sigma)$ with a second-order valuation $\bar{A} \in (2^V)^{\bar{X}}$. The idea is to put the information about which nodes of t are used in the valuations $\bar{A}$ directly in the labels of the nodes, by working with trees over the extended alphabet

$$\Sigma_{\bar{X}} \stackrel{\text{def}}{=} \Sigma \times \{0,1\}^{\bar{X}}$$

instead. The idea is quite straightforward and illustrated below in Figure 5.4.

$\bar{A}(X_1) = \{v_7\}$
$\bar{A}(X_2) = \{v_2, v_5, v_6\}$
$\bar{A}(X_3) = \{v_2, v_4, v_5, v_6\}$
$\bar{A}(X_4) = \{v_1\}$

(A) A second-order valuation $\bar{A}$ of $\bar{X} = \{X_1, X_2, X_3, X_4\}$ in the tree of Figure 5.2.



(B) The valuation tree $t \otimes \bar{A}$.

FIGURE 5.4. A valuation tree $t \otimes \bar{A}$ for the tree t of Figure 5.2.

More formally, define the projection functions $\pi_\Sigma \colon \Sigma_{\bar{X},\bar{x}} \to \Sigma$ and $\pi_X \colon \Sigma_{\bar{X}} \to \{0,1\}$ for each $X \in \bar{X}$ on labels $\ell = (a, \bar{B}) \in \Sigma_{\bar{X}}$ by

$$\pi_\Sigma(a, \bar{B}) \stackrel{\text{def}}{=} a \qquad\qquad \pi_X(a, \bar{B}) \stackrel{\text{def}}{=} \bar{B}(X) \ .$$

For instance, in Figure 5.4b, consider the label $\ell = (b, 0110)$ of node v_2: $\pi_\Sigma(\ell) = b$, $\pi_{X_1}(\ell) = 0$, $\pi_{X_2}(\ell) = 1$, $\pi_{X_3}(\ell) = 1$, and $\pi_{X_4}(\ell) = 0$. Then the valuation tree $t \otimes \bar{A}$ is the tree $t' \in \mathsf{Tree}(\Sigma_{\bar{X}})$ such that $\pi_\Sigma(t') = t$ and, for all $X \in \bar{X}$, $\bar{A}(X) = \{v \mid \exists \ell \in \Sigma_{\bar{X}} \,.\, v \in \ell \text{ and } \pi_X(\ell) = 1\}$. Any tree in $\mathsf{Tree}(\Sigma_{\bar{X}})$ is a valuation tree $t \otimes \bar{A}$ for some valuation $\bar{A}$ of $\bar{X}$.

Automata Construction. The following statement yields a complete deterministic automaton $\mathcal{A}_\varphi$ of size doubly exponential in the size of a quantifier-free MSO formula φ in the standard syntax.

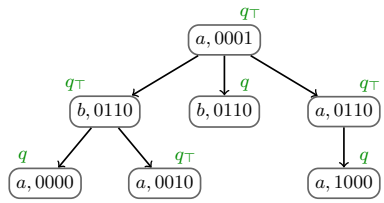
Lemma 5.19. *Let φ be a quantifier-free MSO formula over the signature $\{E\} \cup \Sigma$ with $\mathsf{free}(\varphi) = \bar{x}$ and $\mathsf{free}_2(\varphi) = \bar{X}$. Then we can construct in double exponential time a complete deterministic threshold tree automaton $\mathcal{A}_\varphi$ over the alphabet $\Sigma_{\bar{X}\cup\bar{X}_{\bar{x}}}$ with at most $2^{|\varphi|}$ states and threshold 1 such that, for all trees $t \otimes \bar{A} \in \mathsf{Tree}(\Sigma_{\bar{X}\cup\bar{X}_{\bar{x}}})$, $t \otimes \bar{A} \in L(\mathcal{A}_\varphi)$ if and only if $t, \bar{A} \models \varphi^\dagger$.*

PROOF. We are going to proceed by induction over the quantifier-free formula φ, but first we show how to handle the atomic formulæ $a(X)$, $E(X,Y)$, and $X \subseteq Y$ of the alternative syntax.

Existential Labels. Let us first define a complete deterministic threshold tree automaton $\mathcal{A}_{a(X)}$ for $X \in \bar{X} \cup \bar{X}_{\bar{x}}$ and $a \in \Sigma$. It has state set $Q \stackrel{\text{def}}{=} \{q, q_\top\}$, accepting set $\{q_\top\}$, and defines the transitions for $\ell \in \Sigma_{\bar{X} \cup \bar{X}_{\bar{x}}}$ and $m \in \mathbb{M}(Q)$

$$\begin{aligned} \ell(m) &\to q && \text{if } (\pi_\Sigma(\ell) \neq a \text{ or } \pi_X(\ell) = 0) \text{ and } m(q_\top) = 0\,; \\ \ell(m) &\to q_\top && \text{if } (\pi_\Sigma(\ell) = a \text{ and } \pi_X(\ell) = 1) \text{ or } m(q_\top) \geq 1\,. \end{aligned}$$

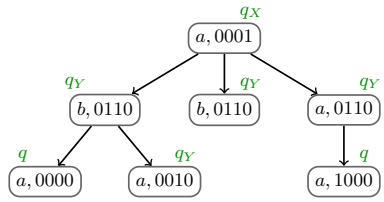
This set of transitions can be defined with threshold 1. Then $L_{\mathcal{A}_{a(X)}}(q)$ is the set of trees where no node has label $(a, \dots, 1, \dots)$ with a 1 for the variable X, and $L_{\mathcal{A}_{a(X)}}(q_\top)$ the set of trees that do.
For instance, in Figure 5.4b, the unique run of $\mathcal{A}_{a(X_3)}$ labels the nodes as displayed on the right.



Existential Edges. Let us define a complete deterministic threshold tree automaton $\mathcal{A}_{E(X,Y)}$ for $X, Y \in \bar{X} \cup \bar{X}_{\bar{x}}$. It has state set $\{q, q_Y, q_X\}$, accepting set $\{q_X\}$, and defines the transitions for $\ell \in \Sigma_{\bar{X} \cup \bar{X}_{\bar{x}}}$ and $m \in \mathbb{M}(Q)$

$$\begin{aligned} \ell(m) &\to q && \text{if } (\pi_X(\ell) = 0 \text{ or } m(q_Y) = 0) \text{ and } \pi_Y(\ell) = 0 \text{ and } m(q_X) = 0\,; \\ \ell(m) &\to q_Y && \text{if } (\pi_X(\ell) = 0 \text{ or } m(q_Y) = 0) \text{ and } \pi_Y(\ell) = 1 \text{ and } m(q_X) = 0\,; \\ \ell(m) &\to q_X && \text{if } (\pi_X(\ell) = 1 \text{ and } m(q_Y) \geq 1) \text{ or } m(q_X) \geq 1\,. \end{aligned}$$

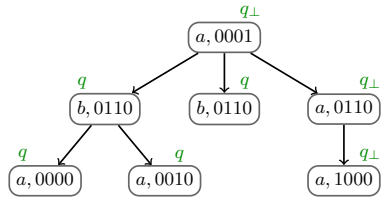
This set of transitions can be defined with threshold 1. The intuition behind this automaton is that $\mathcal{A}_{E(X,Y)}$ must first encounter a node in the valuation of Y and move to q_Y, and immediately afterwards see that its parent is in the valuation of X in order to move to q_X.
For instance, in Figure 5.4b, the unique run of $\mathcal{A}_{E(X_4,X_3)}$ labels the nodes as displayed on the right.



Containment. Let us define a complete deterministic threshold tree automaton $\mathcal{A}_{X \subseteq Y}$ for $X, Y \in \bar{X} \cup \bar{X}_{\bar{x}}$. It has state set $\{q, q_\bot\}$, accepting set $\{q\}$, and defines the transitions for $\ell \in \Sigma_{\bar{X} \cup \bar{X}_{\bar{x}}}$ and $m \in \mathbb{M}(Q)$

$$\begin{aligned} \ell(m) &\to q && \text{if } (\pi_X(\ell) = 0 \text{ or } \pi_Y(\ell) = 1) \text{ and } m(q_\bot) = 0\,; \\ \ell(m) &\to q_\bot && \text{if } (\pi_X(\ell) = 1 \text{ and } \pi_Y(\ell) = 0) \text{ or } m(q_\bot) \geq 1\,. \end{aligned}$$

This set of transitions can be defined with threshold 1. The intuition is that $\mathcal{A}_{X \subseteq Y}$ moves to $q_\bot$ and stays in that state as soon as it sees a node that contradicts the inclusion, and otherwise stays in q.
For instance, in Figure 5.4b, the unique run of $\mathcal{A}_{X_1 \subseteq X_3}$ labels the nodes as displayed on the right.



Main Induction. We proceed by induction over the quantifier-free MSO formula φ.

Case $a(x)$: then $|a(x)| = 2$ and $(a(x))^\dagger = a(X_x)$.
The automaton $\mathcal{A}_{a(X_x)}$ is a complete deterministic automaton with $2 \leq 2^2$ states and threshold 1 such that $t \otimes \bar{A} \in L(\mathcal{A}_{a(X_x)})$ if and only if $t, \bar{A} \models a(X_x)$.

Case $E(x,y)$: then $|E(x,y)| = 3$ and $(E(x,y))^\dagger = E(X_x, X_y)$. The automaton $\mathcal{A}_{E(X_x,X_y)}$ is a complete deterministic automaton with $3 \leq 2^3$ states and threshold 1 such that $t \otimes \bar{A} \in L(\mathcal{A}_{E(X_x,X_y)})$ if and only if $t, \bar{A} \models E(X_x, X_y)$.

Case $x = y$: then $|x = y| = 3$ and $(x = y)^\dagger = X_x \subseteq X_y \wedge X_y \subseteq X_x$. The product automaton constructed by Proposition 5.9 from $\mathcal{A}_{X_x \subseteq X_y}$ and $\mathcal{A}_{X_y \subseteq X_x}$ is a complete deterministic automaton with $2 \times 2 \leq 2^3$ states and threshold 1 such that $t \otimes \bar{A} \in L(\mathcal{A}_{X_x \subseteq X_y}) \cap L(\mathcal{A}_{X_y \subseteq X_x})$ if and only if $t, \bar{A} \models X_x \subseteq X_y \wedge X_y \subseteq X_x$.

Case $X(x)$: then $|X(x)| = 2$ and $(X(x))^\dagger = X_x \subseteq X$. The automaton $\mathcal{A}_{X_x \subseteq X}$ is a complete deterministic automaton with $2 \leq 2^2$ states and threshold 1 such that $t \otimes \bar{A} \in L(\mathcal{A}_{X_x \subseteq X})$ if and only if $t, \bar{A} \models X_x \subseteq X$.

Case $\neg\varphi$: then $|\neg\varphi| = 1 + |\varphi|$ and $(\neg\varphi)^\dagger = \neg(\varphi^\dagger)$. By induction hypothesis, we have constructed a complete deterministic automaton $\mathcal{A}_\varphi$, and applying Proposition 5.8 yields a complete deterministic automaton $\mathcal{A}_{\neg\varphi}$ with at most $2^{|\varphi|} \leq 2^{|\varphi|+1}$ states and threshold 1 such that $t \otimes \bar{A} \in L(\mathcal{A}_{\neg\varphi})$ if and only if $t, \bar{A} \not\models \varphi^\dagger$.

Case $\varphi \wedge \psi$: then $|\varphi \wedge \psi| = 1 + |\varphi| + |\psi|$ and $(\varphi \wedge \psi)^\dagger = (\varphi^\dagger) \wedge (\psi^\dagger)$. By induction hypothesis, we have constructed two complete deterministic automata $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$, and applying Proposition 5.9 yields a complete deterministic automaton $\mathcal{A}_{\varphi\wedge\psi}$ with at most $2^{|\varphi|} \cdot 2^{|\psi|} \leq 2^{1+|\varphi|+|\psi|}$ states and threshold 1 such that $t \otimes \bar{A} \in L(\mathcal{A}_{\varphi\wedge\psi})$ if and only if $t, \bar{A} \models \varphi^\dagger$ and $t, \bar{A} \models \psi^\dagger$. □

Exercise 5.5 (Counting MSO on Trees). Monadic second order logic can be extended with *counting*, by adding to its syntax an atomic formula

$$\mathsf{card}_{m,r}(X)$$

for each $m > r$ in $\mathbb{N}$ and $X \in \mathcal{X}_2$. The resulting logic is denoted by CMSO, or $\mathsf{C}_m\mathsf{MSO}$ if we restrict the syntax to some fixed m. Its semantics is defined by

$$\mathfrak{A}, \bar{A}, \bar{a} \models \mathsf{card}_{m,r}(X) \qquad \text{if } |\bar{A}(X)| \equiv r \bmod m$$

Define a complete deterministic Presburger tree automaton (c.f. exercise 5.1) that accepts the valuation trees of counting atomic formulæ. More precisely, let $m > r$ be in $\mathbb{N}$, $\bar{x}$ and $\bar{X}$ be two finite sets of first-order and second-order variables with $X \in \bar{X}$, and Σ be a finite alphabet. Define a complete deterministic Presburger tree automaton $\mathcal{A}_{\mathsf{card}_{m,r}(X)}$ such that, for all trees $t \otimes \bar{A} \in \mathsf{Tree}(\Sigma_{\bar{X} \cup \bar{X}_{\bar{x}}})$,

$$t \otimes \bar{A} \in L(\mathcal{A}_{\mathsf{card}_{m,r}(X)}) \quad \text{if and only if} \quad t, \bar{A} \models \mathsf{card}_{m,r}(X)\ .$$

5.3.2.2. *Full MSO.* We are now ready to tackle quantifiers and complete the proof, first in the case of formulæ at level $\Sigma_i\mathsf{MSO}$ or $\Pi_i\mathsf{MSO}$ of the MSO alternation hierarchy (which is defined analogously to the first-order alternation hierarchy in Section 1.2.3).

Lemma 5.20. *Let φ be an MSO formula over the signature $\{E\} \cup \Sigma$ with $\mathsf{free}(\varphi) = \bar{x}$ and $\mathsf{free}_2(\varphi) = \bar{X}$ and alternation rank i. Then we can construct in $(i+2)$-iterated exponential time a complete deterministic threshold tree automaton $\mathcal{A}_\varphi$ over the alphabet $\Sigma_{\bar{X} \cup \bar{X}_{\bar{x}}}$*

with $\mathsf{exp}^{i+1}(O(|\varphi|))$ *states and threshold in* $\mathsf{exp}^{i}(O(1))$ *such that, for all trees* $t \otimes \bar{A} \in \mathsf{Tree}(\Sigma_{\bar{X}\cup\bar{X}_{\bar{x}}})$, $t \otimes \bar{A} \in L(\mathcal{A}_\varphi)$ *if and only if* $t, \bar{A} \models \varphi^\dagger$.

Proof. We work by induction over i. For the base case, if $i = 0$ then φ is quantifier-free and Lemma 5.19 yields the result. Otherwise, put φ in prenex normal form, with $i > 0$ quantifier blocks, alternating $(i-1)$ times between existential and universal quantifiers. Thanks to Proposition 5.8, complementing a complete deterministic threshold tree automaton is essentially for free, thus it suffices to show how to handle the case of an initial block of existential quantifiers. Thus $\varphi = \exists\bar{Y}\exists\bar{y}.\psi$ where ψ has alternation rank $(i-1)$ and we already have constructed a complete deterministic automaton $\mathcal{A}_\psi$ with $\mathsf{exp}^i(O(|\psi|))$ states and threshold $\mathsf{exp}^{i-1}(O(1))$ by induction hypothesis.

By definition of the translation into the alternative syntax, $(\exists\bar{Y}\exists\bar{y}.\psi)^\dagger$ is equivalent to $\exists\bar{Y}\bar{Y}_{\bar{y}}.\psi^\dagger \wedge \bigwedge_{y\in\bar{y}} \mathit{singleton}(Y_y)$ and $|\exists\bar{Y}\exists\bar{y}.\psi| = |\bar{Y}|+|\bar{y}|+|\psi|$. Observe that $\mathsf{free}(\psi) = \bar{x}\cup\bar{y}$ and $\mathsf{free}_2(\psi) = \bar{X}\cup\bar{Y}$. A valuation for $\psi^\dagger$ can therefore be decomposed as a valuation $\bar{A}$ of the variables in $\bar{X}\cup\bar{X}_{\bar{x}}$ and a valuation $\bar{B}$ of the variables in $\bar{Y}\cup\bar{Y}_{\bar{y}}$. By induction hypothesis, for all trees $t \otimes \bar{A} \otimes \bar{B} \in \mathsf{Tree}(\Sigma_{\bar{X}\cup\bar{X}_{\bar{x}}\cup\bar{Y}\cup\bar{Y}_{\bar{y}}})$, $t \otimes \bar{A} \otimes \bar{B} \in L(\mathcal{A}_\psi)$ if and only if $t, \bar{A}\bar{B} \models \psi^\dagger$.

By a straightforward adaptation of the automaton in Example 5.5, there is a complete deterministic threshold tree automaton $\mathcal{A}_{\mathit{singleton}(Y_y)}$ for any $y \in \bar{y}$, with 3 states and threshold 2. Applying Proposition 5.9 to $\mathcal{A}_\psi$ and all the $\mathcal{A}_{\mathit{singleton}(Y_y)}$ for $y \in \bar{y}$ yields a complete deterministic threshold tree automaton $\mathcal{A}$ with $3^{|\bar{y}|} \cdot \mathsf{exp}^i(O(|\psi|)) \subseteq \mathsf{exp}^i(O(|\varphi|))$ states and threshold $\max(3, \mathsf{exp}^{i-1}(O(1))) = \mathsf{exp}^{i-1}(O(1))$ such that $t \otimes \bar{A} \otimes \bar{B} \in L(\mathcal{A})$ if and only if $t, \bar{A}\bar{B} \models \psi^\dagger \wedge \bigwedge_{y\in\bar{y}} \mathit{singleton}(Y_y)$.

Finally, $t, \bar{A} \models \varphi^\dagger$ if and only if there exists a valuation $\bar{B}$ of $\bar{Y}\cup\bar{Y}_{\bar{y}}$ such that $t, \bar{A}\bar{B} \models \psi^\dagger \wedge \bigwedge_{y\in\bar{y}} \mathit{singleton}(Y_y)$. Thus it suffices to apply Proposition 5.10 with the projection $f\colon \Sigma_{\bar{X}\cup\bar{X}_{\bar{x}}\cup\bar{Y}\cup\bar{Y}_{\bar{y}}} \to \Sigma_{\bar{X}\cup\bar{X}_{\bar{x}}}$ to the automaton $\mathcal{A}$ to obtain $\mathcal{A}'$ such that $t \otimes \bar{A} \in L(\mathcal{A}')$ if and only if $t, \bar{A} \models \varphi^\dagger$. Unfortunately, $\mathcal{A}'$ is not deterministic, but applying Proposition 5.6 yields the desired $\mathcal{A}_\varphi$ with $\mathsf{exp}^{i+1}(O(|\varphi|))$ states and threshold $\mathsf{exp}^i(O(1))$. □

The final result is the following statement for MSO sentences.

Theorem 5.21. *Let* φ *be an MSO sentence over the signature* $\{E\}\cup\Sigma$ *with alternation rank* i. *Then we can construct in* $(i+2)$*-iterated exponential time a complete deterministic threshold tree automaton* $\mathcal{A}_\varphi$ *over* Σ *with* $\mathsf{exp}^{i+1}(O(|\varphi|))$ *states and threshold in* $\mathsf{exp}^i(O(1))$ *such that, for all trees* $t \in \mathsf{Tree}(\Sigma)$, $t \in L(\mathcal{A}_\varphi)$ *if and only if* $t \models \varphi$.

Proof. By (5.16), because φ is a sentence, it is equivalent to $\varphi^\dagger$. The rest follows from Lemma 5.20. □

Depending on the intended applications of Theorem 5.21, a non-deterministic automaton might suffice. If φ is an MSO sentence from the $\Sigma_i\mathsf{MSO}$ fragment (i.e., of alternation rank i and starting with a block of existential quantifiers), the last determinisation in the proof of Lemma 5.20 is not required.

Proposition 5.22. *Let* φ *be an* $\Sigma_i\mathsf{MSO}$ *sentence over the signature* $\{E\}\cup\Sigma$. *Then we can construct in* $(i+1)$*-iterated exponential time a threshold tree automaton* $\mathcal{A}_\varphi$ *over* Σ *with* $\mathsf{exp}^i(O(|\varphi|))$ *states and threshold in* $\mathsf{exp}^{i-1}(O(1))$ *such that, for all trees* $t \in \mathsf{Tree}(\Sigma)$, $t \in L(\mathcal{A}_\varphi)$ *if and only if* $t \models \varphi$.

In particular, for the $\Sigma_1\mathsf{MSO}$ fragment, Proposition 5.22 yields an automaton with an exponential number of states and constant threshold. Beware here that the $\Sigma_1\mathsf{MSO}$ fragment of monadic second-order logic is *not* the same as existential MSO defined in Section 5.2.2: the latter allows arbitrary first-order universal quantification, which the former forbids.

5.3.3. Consequences. Let us look at the consequences of Theorem 5.21. The first one pertains to the parameterised model-checking problem for monadic second-order logic over directed trees, which is in FPT.

Corollary 5.23. $\mathsf{p\text{-}MC}(\mathsf{MSO}, \mathsf{Tree})$ *is in fixed-parameter linear time* $f(|\varphi|) \cdot |t|$ *for a function* f *in* $\mathsf{tower}(O(n))$.

Proof. Given Σ a finite alphabet, $\varphi \in \mathsf{MSO}[\{E\} \cup \Sigma]$ a sentence of alternation rank i, and $t \in \mathsf{Tree}(\Sigma)$, we can compute a complete deterministic threshold tree automaton $\mathcal{A}_\varphi$ in time $\mathsf{exp}^{i+2}(O(|\varphi|))$ and check whether $t \in L(\mathcal{A}_\varphi)$ in time $O(\|\mathcal{A}_\varphi\| + |t|)$ by Proposition 5.13. □

The second is that the satisfiability problem for monadic second-order logic over directed trees is decidable.

Corollary 5.24. *Let* $i > 0$. *Then* $\mathsf{SAT}(\Sigma_i\mathsf{MSO}, \mathsf{Tree})$ *is in* $(i{+}1)$-EXP, *and* $\mathsf{SAT}(\mathsf{MSO}, \mathsf{Tree})$ *is in* TOWER.

Proof. Given Σ a finite alphabet and a monadic second-order sentence φ, then $\varphi \in \Sigma_i\mathsf{MSO}[\{E\} \cup \Sigma]$ for some $i > 0$. By Proposition 5.22 we can construct in $(i+1)$-iterated exponential time a threshold tree automaton $\mathcal{A}_\varphi$ of size in $\mathsf{exp}^{i+1}(O(|\varphi|))$ such that φ is satisfiable if and only if $L(\mathcal{A}_\varphi) \neq \emptyset$. The latter check can be performed in time $O(\|\mathcal{A}_\varphi\|)$ by Proposition 5.14. □

The last consequence is that, when combining Theorem 5.21 with Theorem 5.16, we obtain the following expressiveness equivalence of monadic second-order logic on finite directed trees with that of existential MSO.

Corollary 5.25. *Let* φ *be an MSO sentence over* $\{E\} \cup \Sigma$ *for a finite alphabet* Σ. *Then we can construct in time* $\mathsf{tower}(O(|\varphi|))$ *an existential MSO sentence* ψ *logically equivalent to* φ *over* $\mathsf{Tree}(\Sigma)$.

Thus, maybe surprisingly, over finite directed trees any sentence of monadic second-order logic is logically equivalent to a first-order sentence modulo an expansion with finitely many unary relation symbols (aka a *monadic lift*).

5.4. Trees Are Hard!

TODO

CHAPTER 6

The Case of Bounded Tree-Width

The results of the previous chapter give us at last a ray of hope that model-checking some finite structures may sometimes be tractable. We are going to build on this basis, by now turning our attention to structures that are somehow 'tree-like.'

The degree of likeness to a tree is captured by an invariant called *tree-width*, with structures of low tree-width being close to being trees, and we are going learn the basics about tree-width in Section 6.1.

Crucially, we will see in Section 6.2 that graphs and structures of tree-width bounded by some fixed k have a monadic second-order interpretation in trees from a suitable term algebra, allowing to reduce their model-checking problem to the case of trees already handled in the previous chapter. The main outcome of this chapter is our first *algorithmic meta-theorem*: Courcelle's Theorem shows that over a class of structures of bounded tree-width, any property expressible in monadic second-order logic can be decided in fixed-parameter linear time (see **??**).

The second main outcome in Section 6.3 is a generalisation of Yannakakis' Algorithm for acyclic conjunctive queries (c.f. Theorem 4.8): the class of conjunctive queries of bounded tree-width admits a tractable model-checking problem over arbitrary finite structures.

6.1. Tree-Width

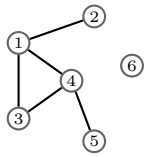
This section provides the basic definitions surrounding tree-width. We will start in the upcoming Section 6.1.1 by focusing on the case of simple, undirected graphs, but we will see in Section 6.1.2 that the notion generalises to arbitrary finite structures in a seamless fashion. Finally, we will mention the basic facts about the computation of the tree-width of a structure in Section 6.1.3.

6.1.1. Tree Decompositions. The tree-width $\mathsf{tw}(G)$ of a simple graph G provides a measure of how 'close' G is to being a tree; for instance, we will see that $\mathsf{tw}(G) = 1$ if and only if G is acyclic (thus a forest in the sense of **??**), while $\mathsf{tw}(K_k) = k - 1$ for a complete graph with k vertices.

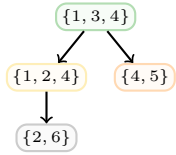
Definition 6.1 (Tree Decomposition)**.** Let G be a simple graph. A *tree decomposition* for G is a non-empty labelled directed tree $(T, \beta) \in \mathsf{Tree}(2^{V(G)})$ where $\beta \colon V(T) \to 2^{V(G)}$ maps the nodes of T to their *bags* of vertices from G, such that

(1) for each edge $\{v, v'\} \in E(G)$, there exists a node $s \in V(T)$ such that the edge belongs to its bag: $\{v, v'\} \subseteq \beta(s)$, and
(2) for each vertex $v \in V(G)$, the set of nodes $\beta^{-1}(v) \stackrel{\text{def}}{=} \{s \in V(T) \mid v \in \beta(s)\}$ is non-empty and connected in T.

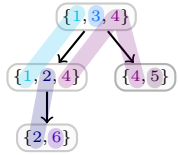
If this feels familiar, that's normal: the join trees we used for Yannakakis's Algorithm in Section 4.2.1 are indeed a restricted form of tree decompositions. We shall discuss the connection further in Section 6.3.1.2. See Figure 6.1 for some examples of tree decompositions.



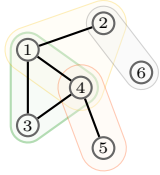
(A) A simple graph.



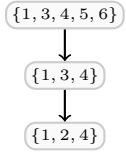
(B) A tree decomposition of the graph of Figure 6.1a.



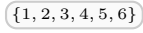
(C) The subsets $\beta^{-1}(v)$ in the tree decomposition of Figure 6.1b are connected.



(D) The bags of Figure 6.1b on the graph of Figure 6.1a: every edge belongs to some bag.



(E) Another tree decomposition of the graph of Figure 6.1a.



(F) Yet another tree decomposition of the graph of Figure 6.1a.

FIGURE 6.1. Illustrations of tree decompositions.

Definition 6.2 (Tree-width)**.** Let G be a simple graph. The *width* of a tree decomposition (T, β) is the maximal number $\max_{s \in V(T)} |\beta(s)| - 1$ of elements in the bags of T, minus one. The *tree-width* $\mathsf{tw}(G)$ of G is the minimum width over all the tree decompositions of G.

Example 6.3 (Width of tree decompositions)**.** The tree decompositions of Figure 6.1 have: width 2 in Figure 6.1b, width 3 in Figure 6.1c, and width 5 in Figure 6.1e—this is the worst case, since every simple graph has a tree decomposition of width $|V(G)| - 1$. The graph G of Figure 6.1a has therefore $\mathsf{tw}(G) \leq 2$; it actually has $\mathsf{tw}(G) = 2$ by the forthcoming Corollary 6.12 since it contains K_3 as a subgraph.

The rationale for the really annoying 'minus one' in the definition of tree-width is for the acyclic graphs to have tree-width one. Indeed, with this definition, the empty graph has tree-width -1, and a graph has tree-width 0 if and only if it is non-empty and has no

edges. An non-empty acyclic graph G with at least one edge has tree-width 1: it needs tree-width > 0 since it has an edge, and tree-width 1 suffices because we can build a tree decomposition with one node s_e per edge $e \in E(G)$, labelled by $\beta(s_e) \stackrel{\text{def}}{=} e$, and suitable tree edges connecting the nodes; see Figure 6.2. Being acyclic actually characterises having tree-width at most one; see Corollary 6.11.

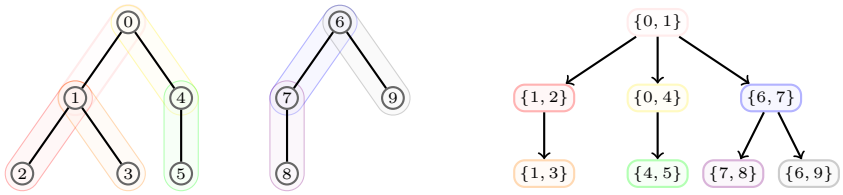


FIGURE 6.2. A tree decomposition of width 1 of an acyclic graph.

6.1.1.1. *Basic Closure Properties.* For two simple graphs H and G, we say that G is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq G$. Put differently, H is a subgraph of G if it can be obtained by removing some edges or some vertices from G. Over the class Graph, is equivalent to asking for the identity map $V(H) \to V(G)$ to be an injective homomorphism, and we will reuse the notation $H \hookrightarrow G$ of Section 1.1.2.1, since $H \hookrightarrow G$ entails that H is isomorphic to a subgraph of G.

Beware that this is *not standard notation* in graph theory, where '$H \subseteq G$' is typically used to denote that H is a subgraph of G; however we will keep our understanding that $H \subseteq G$ denotes the existence of an embedding of H into G, which in terms of graphs means that H is an *induced subgraph* of G: indeed, $H \subseteq G$ if and only if $V(H) \subseteq V(G)$ and $E(H) = E(G)_{\restriction V(H)}$, if and only if H can be obtained by removing some vertices from G. If G is a graph and $U \subseteq V(G)$, we write $G[U]$ for the subgraph of G induced by U and $G - U$ for the induced subgraph $G[V(G) \setminus U]$.

Lemma 6.4 (Monotonicity under taking subgraphs)**.** *If $H \hookrightarrow G$, then* $\mathsf{tw}(H) \leq \mathsf{tw}(G)$.

PROOF. Given a tree decomposition (T, β) of width $\mathsf{tw}(G)$ for G, (T, β') where $\beta'(s) \stackrel{\text{def}}{=} \beta(s) \cap V(H)$ for all $s \in V(T)$ is also a tree decomposition of width $\leq \mathsf{tw}(G)$ for H. □

A class $\mathscr{C}$ of graphs or coloured graphs is called *monotone* if it is closed under taking subgraphs, and *hereditary* if it is closed under taking induced subgraphs; thus a monotone class is also hereditary but the converse might not hold. Then Lemma 6.4 shows that for all k, the class of graphs of tree-width at most k is monotone.

Lemma 6.5 (Closure under disjoint unions)**.** *Let G and G' be two graphs with disjoint sets of vertices. Then* $\mathsf{tw}(G \uplus G') = \max(\mathsf{tw}(G), \mathsf{tw}(G'))$.

PROOF. To see that $\mathsf{tw}(G \uplus G') \leq \max(\mathsf{tw}(G), \mathsf{tw}(G'))$, given two tree decompositions (T, β) and (T', β') for G and G' of widths $\mathsf{tw}(G)$ and $\mathsf{tw}(G)'$ respectively, connect the root of T by an edge to the root of T' to obtain a tree decomposition for $G \uplus G'$ with width $\max(\mathsf{tw}(G), \mathsf{tw}(G'))$.

Conversely, $G \hookrightarrow G \uplus G'$ and $G' \hookrightarrow G \uplus G'$, thus $\mathsf{tw}(G) \leq \mathsf{tw}(G \uplus G')$ and $\mathsf{tw}(G') \leq \mathsf{tw}(G \uplus G')$ by Lemma 6.4, and therefore $\max(\mathsf{tw}(G), \mathsf{tw}(G')) \leq \mathsf{tw}(G \uplus G')$. □

6.1.1.2. *Small Tree Decompositions and Bounded Degeneracy.* According to Definition 6.1, a tree decomposition for a graph G can have arbitrary size (e.g., one can just insert nodes with empty bags below the leaves). A tree decomposition (T, β) for a graph G is *small* if for all $s \neq s' \in V(T)$, $\beta(s) \not\subseteq \beta(s')$. In Figure 6.1, the decompositions depicted in figures 6.1b and 6.1f are small, but the one in Figure 6.1e is not.

Lemma 6.6 (Size of small tree decompositions)**.** *If (T, β) is a small tree decomposition of a graph G, then $|V(T)| \leq |V(G)|$.*

Proof. We proceed by induction on $|V(G)|$. If $|V(G)| = 0$ then the only tree decomposition is the empty one and we are done. Otherwise, let (T, β) be a small tree decomposition of G and consider a leaf s of T and its parent s'. Since $\beta(s) \not\subseteq \beta(s')$, there exists a vertex $v \in \beta(s) \setminus \beta(s')$. By condition (2) of Definition 6.1, we then have $v \notin \beta(s'')$ for all $s'' \neq s$ in $V(T)$. Consider the new labelled tree (T', β') obtained by removing the leaf s from T and the new graph G' obtained by removing the vertex v from G: then (T', β') is a small tree decomposition of G'. By induction hypothesis, $|V(T)| = |V(T')| + 1 \leq |V(G')| + 1 = |V(G)|$. □

Lemma 6.7 (Linear time construction of small decompositions)**.** *Let G be a graph and (T, β) be a tree decomposition for G. Then we can compute in linear time in $|V(T)|$ a small tree decomposition (T', β') for G with the same width such that $V(T') \subseteq V(T)$ and $\beta'(s) = \beta(s)$ for all $s \in V(T')$.*

Proof. Starting from the leaves of (T, β), contract the edges $(s, s') \in E(T)$ for which either $\beta(s) \subseteq \beta(s')$ or $\beta(s') \subseteq \beta(s)$, keeping the largest one and its label. □

A graph G is called *k-degenerate* if, in all non-empty subgraphs $H \hookrightarrow G$, there exists a vertex of degree at most k.

Corollary 6.8. *If $\mathsf{tw}(G) \leq k$, then G is k-degenerate.*

Proof. Let G be a graph and $H \hookrightarrow G$ a non-empty subgraph of G. Then H has tree-width at most k as well by Lemma 6.4, and by Lemma 6.7 it has a small tree decomposition (T, β) of width at most k. If $|V(T)| = 1$ then $|V(H)| \leq k + 1$ and the result follows. Otherwise, let s be a leaf of T and s' its parent; since $\beta(s) \not\subseteq \beta(s')$, there exists $v \in \beta(s) \setminus \beta(V(T) \setminus \{s\})$. By condition (1) of Definition 6.1, all the edges $\{v, v'\} \in E(H)$ incident to v must be such that $v' \in \beta(s)$, hence the degree of v in H is at most k. □

Graphs of bounded degeneracy are a key notion in graph theory. Among many other properties, they are 'sparse' in the sense that they have few edges: by recursively picking a vertex of degree at most k, we see that $|E(G)| \leq k \cdot |V(G)|$ if G is k-degenerate. Not all k-degenerate graphs have tree-width at most k; for instance, grids have degree at most 4 and are thus 4-degenerate, but for all $k > 0$, $\mathsf{tw}(G_{k \times k}) = k$ (see Proposition 8.1 in Chapter 8).

6.1.1.3. *Connectivity and Separators.* One can relate connectivity inside a graph with connectivity inside any of its tree decompositions. Let G be a simple graph and (T, β) a tree decompositions of G. For a subset $S \subseteq V(T)$ of the nodes of T, let

$$\beta(S) \stackrel{\text{def}}{=} \bigcup_{s \in S} \beta(s)$$

denote the union of the bags of S. Conversely, for a subset $U \subseteq V(G)$ of the vertices of G, let

$$\beta^{-1}(U) \stackrel{\text{def}}{=} \bigcup_{v \in U} \beta^{-1}(v) = \{s \in V(T) \mid \exists v \in U \,.\, v \in \beta(s)\} \,.$$

Assume that U is a connected subset of G, for instance $\{1, 2\}$ in Figure 6.1a: there exists a path connecting any two of its vertices in G, for instance ②—①. For each edge $\{v, v'\} \in E$ along that path, $\beta^{-1}(v) \cap \beta^{-1}(v') \supseteq \{v, v'\} \neq \emptyset$ by condition (1) of a tree decomposition, and indeed $\beta^{-1}(2)$ and $\beta^{-1}(1)$ have at least one node of T in common ($\{1, 2, 4\}$ in Figure 6.1c). As both $\beta^{-1}(2)$ and $\beta^{-1}(1)$ are themselves connected subsets of T by condition (2) of a tree decomposition (see the cyan and dark blue connections in Figure 6.1c), we have that $\beta^{-1}(\{1, 2\})$ is connected in T. We just showed the following.

Lemma 6.9. *Let G be a graph and (T, β) a tree decomposition for G. If U is a connected set of vertices of G, then $\beta^{-1}(U)$ is a connected set of nodes of T.*

In a tree decomposition (T, β) of a graph G, an edge $(s, s') \in E(T)$ is sometimes also called a *cut*. This is due to the following lemma, where a set $S \subseteq V(G)$ *separates* $U_1 \subseteq V(G)$ from $U_2 \subseteq V(G)$ if $U_1 \setminus S$ and $U_2 \setminus S$ are disconnected in $G - S$. Equivalently, S is a separator for U_1 and U_2 if any path in G from a vertex in $v_1 \in U_1$ to a vertex $v_2 \in U_2$ contains a vertex from S. For instance, in Figure 6.1a, $\{1, 4\}$ is a separator for $\{2, 6\}$ and $\{3, 5\}$.

Lemma 6.10. *Let (T, β) be a tree decomposition of a graph G, $(s_1, s_2) \in E(T)$ an edge of T, and T_1 and T_2 be the two disjoint subtrees of T defined by removing this edge with $s_1 \in V(T_1)$ and $s_2 \in V(T_2)$. Then $\beta(s_1) \cap \beta(s_2)$ separates $\beta(T_1)$ from $\beta(T_2)$.*

Proof. Let $S \stackrel{\text{def}}{=} \beta(s_1) \cap \beta(s_2)$, $U_1 \stackrel{\text{def}}{=} \beta(T_1)$, and $U_2 \stackrel{\text{def}}{=} \beta(T_2)$. Let $v_1, \ldots, v_n$ be any path in G such that $v_1 \in U_1$ and $v_n \in U_2$. As $V(T) = V(T_1) \cup V(T_2)$ and because $V(G) = \beta(V(T))$ by condition (2) of a tree decomposition, we have $V(G) = U_1 \cup U_2$ and there exists some $1 \leq i \leq n$ where we 'switch' between U_1 and U_2 along the path: either $v_i \in U_1 \cap U_2$, or $v_i \in U_1$ and $v_{i+1} \in U_2$. Let us show that in both cases we visit S along the path.

(1) If $v_i \in U_1 \cap U_2$, there exist some $s'_1 \in \beta^{-1}(v_i) \cap T_1$ and some $s'_2 \in \beta^{-1}(v_i) \cap T_2$. Since $\beta^{-1}(v_i)$ is connected in T by condition (2), there is a path in T connecting $s'_1 \in T_1$ to $s'_2 \in T_2$ remaining within $\beta^{-1}(v_i)$. This path has to use the edge (s_1, s_2) since T is a tree. Therefore both $s_1 \in \beta^{-1}(v_i)$ and $s_2 \in \beta^{-1}(v_i)$, which entails that $v_i \in S$.

(2) If $v_i \in U_1$ and $v_{i+1} \in U_2$, since $\{v_i, v_{i+1}\} \in E(G)$, by condition (1) there is a node $s \in V(T)$ such that $\{v_i, v_{i+1}\} \subseteq \beta(s)$. Assume $s \in T_2$. Then $s \in \beta^{-1}(v_i) \cap T_2$, and since $v_i \in U_1$ we also have $\beta^{-1}(v_i) \cap T_1 \neq \emptyset$. As in the previous case, this entails that $s_1, s_2 \in \beta^{-1}(v_i)$ and therefore $v_i \in S$. If $s \in T_1$, then $s \in \beta^{-1}(v_{i+1}) \cap T_1$, $\beta^{-1}(v_{i+1}) \cap T_1 \neq \emptyset$, and by the same argument $v_{i+1} \in S$. □

Note that the separators defined by cuts in a tree decomposition of width k have size at most k. Having 'balanced' separators of bounded size—where the sets X and Y furthermore contain at least some proportion of the vertices—allows for divide-and-conquer algorithmic approaches. We are going to use Lemma 6.10 more prosaically to prove lower bounds on the tree-width of some classical graphs.

Corollary 6.11. *Let $k > 2$. Then the tree-width of the cycle C_k is 2.*

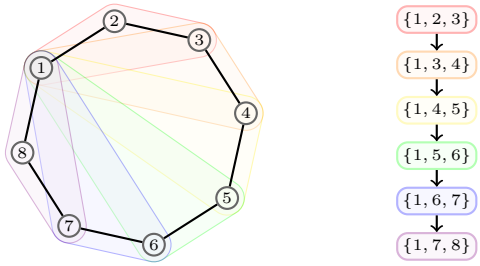


FIGURE 6.3. A tree decomposition of width 2 of the cycle C_8.

PROOF. See Figure 6.3 for an illustration of why $\mathsf{tw}(C_k) \leq 2$. For the sake of contradiction, assume that $\mathsf{tw}(C_k) \leq 1$. By Lemma 6.6 there exists a small tree decomposition (T, β) for C_k of width one. Write the vertices of C_k as $V(C_k) \stackrel{\text{def}}{=} \{1, \dots, k\}$; since $\{1, 2\} \in E(C_k)$ and $\{1, k\} \in E(C_k)$, by condition (1) of tree decompositions there exist s_1, s_2 two nodes of T such that $\{1, 2\} \subseteq \beta(s_1)$ and $\{1, k\} \subseteq \beta(s_2)$. As $|\beta(s)| \leq 2$ for all $s \in V(T)$, $s_1 \neq s_2$.

Since $1 \in \beta(s_1) \cap \beta(s_2)$, by condition (2) there is a path connecting s_1 and s_2 in T. The first edge on that path is of the form $(s_1, s) \in E(T)$ or $(s, s_1) \in E(T)$ for some $s \in V(T)$. By Lemma 6.10, this edge separates $\beta(T_1)$ from $\beta(T_2)$ in G, where T_1 and T_2 are the two disjoint subtrees of T defined by removing that edge and such that $s_1 \in V(T_1)$ and $s, s_2 \in V(T_2)$.

By definition of a separator, the vertices $2 \in \beta(T_1)$ and $k \in \beta(T_2)$ are only connected through $\beta(s_1) \cap \beta(s)$, but since (T, β) is small, $|\beta(s_1)| \leq 2$, $|\beta(s)| \leq 2$, and $1 \in \beta(s_1) \cap \beta(s)$, we actually have $\beta(s_1) \cap \beta(s) = \{1\}$. That means that the only path connecting 2 with k is through 1, contradicting that C_k is a cycle. □

By Lemma 6.4, if G contains a cycle C_k as a subgraph, then $\mathsf{tw}(G) \geq 2$. As we already saw that acyclic graphs had tree-width at most one, Corollary 6.11 entails shows that a graph has tree-width at most one if and only if it is acyclic.

Corollary 6.12. *Let G be a graph and (T, β) a tree decomposition for G. If $U \subseteq V(G)$ induces a complete graph, then there exists $s \in V(T)$ such that $U \subseteq \beta(s)$. Thus for all $k > 0$, $\mathsf{tw}(K_k) = k - 1$.*

PROOF. Since $G[U] \hookrightarrow G$ we can use the construction of Lemma 6.4 to obtain a new labelling $\beta' : s \mapsto \beta(s) \cap U$ such that (T, β') is a tree decomposition for the graph $G[U]$ with the same set of nodes and with $\beta'(s) \subseteq \beta(s)$ for all $s \in V(T)$. Apply then Lemma 6.7 to (T, β') to obtain a small decomposition (T', β'') with $V(T') \subseteq V(T)$ and $\beta''(s) = \beta'(s) \subseteq \beta(s)$ for all $s \in V(T')$. It suffices to show that $U \subseteq \beta''(s)$ for some $s \in V(T')$. To simplify notations, we now call this decomposition (T, β).

Let $U \stackrel{\text{def}}{=} \{1, \dots, k\}$ for some k (note that if $k \leq 1$ there is nothing to prove). Pick a node $s_1 \in V(T)$ such that $\beta(s_1)$ is maximal for bag inclusion: for all $s \in V(T)$, $\beta(s_1) \not\subseteq \beta(s)$.

If $U \subseteq \beta(s_1)$ we are done, so assume the contrary: there exists vertex $j \in U$ such that $j \notin \beta(s_1)$. Since (T, β) is small, $\beta(s)$ is non-empty and there another exists a vertex $i \in U$ such that $i \in \beta(s_1)$. Since $G[U]$ is complete, there exists an edge $\{i, j\}$ and by condition (1) there is a node $s \in V(T)$ with $\{i, j\} \subseteq \beta(s)$. By condition (2), s_1 and s are

connected through a path in T, and let (s_1, s_2) be the first transition taken along that path. By Lemma 6.10, $\beta(s_1) \cap \beta(s_2)$ is a separator for $\beta(T_1)$ and $\beta(T_2)$ where T_1 and T_2 are the two disjoint subtrees of T obtained by removing the edge (s_1, s_2) and such that $s_1 \in V(T_1)$ and $s_2 \in V(T_2)$, and observe that $s \in V(T_2)$ thus $j \in \beta(T_2)$.

Consider any $i' \in \beta(s_1)$: in $G[U]$, any path from $i' \in \beta(T_1)$ to $j \in \beta(T_2)$ must visit a vertex in the separator $\beta(s_1) \cap \beta(s_2)$. In particular, $\{i', j\}$ is an edge of $G[U]$, and $j \notin \beta(s_1)$, thus necessarily $i' \in \beta(s_1) \cap \beta(s_2)$. This shows that $\beta(s_1) \subseteq \beta(s_2)$, but this contradicts our choice of s_1 with $\beta(s_1)$ maximal for inclusion. □

6.1.2. Graphs vs. Structures.

6.1.2.1. *Gaifman Graphs.*

6.1.2.2. *Incidence Graphs.* Alternative to Section 1.1.2.2

Hamiltonian path is not expressible (Calò and Makowsky, 1992, Theorem 4.2) but it is in MSO_2 (Flum and Grohe, 2006, Example 11.49)

6.1.3. Computing Tree Decompositions. Unfortunately, finding a tree decomposition of small width for a graph is not an easy task. In particular, checking that we have the right width value is not easy: the following decision problem is not tractable.

> Problem (TREE-WIDTH).
> **instance:** a finite simple graph G and $k \in \mathbb{N}$
> **question:** $\mathsf{tw}(G) = k$?

Fact 6.13 (Arnborg, Corneil, and Proskurowski, 1987)**.** TREE-WIDTH *is* NP*-complete.*

Thankfully, for the applications we have in mind, we will use k as a *fixed* value, in which case computing a tree decomposition of small width becomes tractable. Even better, there are algorithms that behave well in terms of parameterised complexity.

> Problem (p-TREE-WIDTH).
> **instance:** a finite simple graph G and $k \in \mathbb{N}$
> **parameter:** k
> **question:** $\mathsf{tw}(G) = k$?

In particular, Bodlaender's Algorithm shows that p-TREE-WIDTH is in FPT, even in fixed-parameter linear time, and furthermore when the answer is positive, a tree decomposition with the desired width can be constructed.

Fact 6.14 (Bodlaender, 1996, Theorem 1.1)**.** *There is an algorithm which, given a finite simple graph G and an integer $k \in \mathbb{N}$, runs in time $2^{O(k^3)} \cdot |G|$ and either constructs a tree decomposition of width k for G or reports that $\mathsf{tw}(G) > k$.*

6.2. Courcelle's Theorem

6.3. Conjunctive Queries Redux

Truth be told, I am not overly fond of the proof of **??**, and we will see a more general statement in Chapter 7 that I find more elegant. Nevertheless, tree-width is an important notion, interesting in its own right. In particular, it has strong connections with the case of conjunctive queries from Chapter 4 and provides a generalisation of Yannakakis's Algorithm, which we are going to see now.

6.3.1. Rewritings of Conjunctive Queries. Recall from Section 3.3.2 that we have good complexity upper bounds for the evaluation and model-checking problems for formulæ of small variable width. This was even put forward in Example 4.7 as an intuitive explanation for why Yannakakis's Algorithm works.

One can attempt to attempt to minimise the width of a conjunctive formula φ by a very simple optimisation process. Let us say that a primitive-positive formula ψ is a *rewriting* of a conjunctive query φ if it can be obtained from φ by applying the rule

$$\exists x.(\theta_1 \wedge \theta_2) \rightsquigarrow (\exists x.\theta_1) \wedge \theta_2 \qquad \text{if } x \notin \mathsf{free}(\theta_2) \tag{6.1}$$

modulo associativity and commutativity of the conjunction symbol $\wedge$. Equivalently, a primitive-positive formula ψ is a rewriting if it does not contain any equality atom and all the quantifications $\exists y.\psi'$ occurring in ψ use distinct variable names, which are also distinct from the free variables in $\mathsf{free}(\psi)$.

Example 6.15 (Rewriting)**.** Consider the primitive positive formula of variable width 3

$$\psi \stackrel{\text{def}}{=} \exists x_3.E(x_2,x_3) \wedge \big(\exists x_4.E(x_1,x_4) \wedge E(x_3,x_4)\big) \wedge \big(\exists x_5.E(x_2,x_5) \wedge E(x_3,x_5)\big) \ .$$

All the variables in ψ are distinct and it does not have any equality atom; it is indeed a rewriting of the conjunctive query

$$\varphi = \exists x_3x_4x_5.E(x_2,x_3) \wedge E(x_1,x_4) \wedge E(x_3,x_4) \wedge E(x_2,x_5) \wedge E(x_3,x_5)$$

obtained from ψ by applying the rewriting rule (6.1) in reverse.
The formula ψ is in fact a normal form of φ: the rule (6.1) can no longer be applied. It is not the sole normal form (unfortunately, our rewriting system is not confluent); for instance, the following rewriting of φ is also a normal form, this time of variable width 4:

$$\exists x_4.E(x_1,x_4) \wedge \Big(\exists x_5.E(x_2,x_5) \wedge \big(\exists x_3.E(x_3,x_4) \wedge E(x_2,x_3) \wedge E(x_3,x_5)\big)\Big) \ .$$

Let us generalise the notion of canonical structures to rewritings ψ of conjunctive queries: we can turn ψ back into the original conjunctive query by applying the inverse rewriting rule $(\exists x.\theta_1) \wedge \theta_2 \rightsquigarrow \exists x.(\theta_1 \wedge \theta_2)$, and then construct the associated canonical structure. Equivalently, the structure $\mathsf{can}(\psi)$ has as domain $\mathsf{vars}(\psi)$ the set of all the variables appearing in ψ, and for all $R \in \sigma$ a tuple $(x_1,\ldots,x_{\mathsf{ar}(R)})$ belongs to $R^{\mathsf{can}(\psi)}$ if and only if an atom $R(x_1,\ldots,x_{\mathsf{ar}(R)})$ appears in ψ. For instance, Figure 6.4 shows the canonical structure associated to the rewriting of Example 6.15.

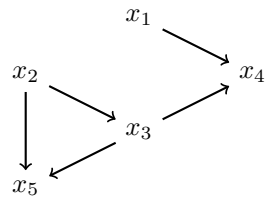


FIGURE 6.4. The canonical structure $\mathsf{can}(\psi)$ of the rewriting ψ from Example 6.15.

6.3.1.1. *A Logical Characterisation.* Recall from Section 4.1.2.1 that any finite structure $\mathfrak{A}$ has an associated Boolean conjunctive query, obtained as its *positive diagram* $\varphi = \mathsf{diag}^+(\mathfrak{A})$; conversely, any conjunctive query $\varphi(\bar{x})$ has an associated *canonical structure* $\mathfrak{A} = \mathsf{can}(\exists \bar{x}.\varphi(\bar{x}))$. We are going to see that we can relate the variable width of rewritings of φ with the tree-width of $\mathfrak{A}$. This will also shed additional light on the discussion in Example 4.7 that led to Yannakakis's Algorithm in Section 4.2.2.

Proposition 6.16. *Let φ be a conjunctive query. If φ has a rewriting of width $k+1$, then its canonical structure has tree-width at most k.*

PROOF. Consider a conjunctive query $\varphi(\bar{x}) = \exists \bar{y}. \bigwedge_{i \in I} R_i(x_{i,1}, \ldots, x_{i,\mathsf{ar}(R)})$ with $\bar{x}_i \stackrel{\text{def}}{=} \{x_{i,1}, \ldots, x_{i,\mathsf{ar}(R)}\} \subseteq \bar{x} \cup \bar{y}$ for all $i \in I$. Let $\mathfrak{A} = \mathsf{can}(\exists \bar{x}.\varphi)$ be the associated canonical structure, with domain $\bar{x} \cup \bar{y}$ and tuples $(x_{i,1}, \ldots, x_{i,\mathsf{ar}(R)}) \in R_i^{\mathfrak{A}}$ for all $i \in I$.

Let ψ be a rewriting of φ of width $w(\psi) \leq k+1$. We are going to construct a tree decomposition T of $\mathfrak{A}$ by induction on the structure of ψ. Up to associativity and commutativity of conjunctions, we can write ψ as

$$\left(\bigwedge_{i_0 \in I_0} R_{i_0}(\bar{x}_{i_0}) \right) \wedge \left(\bigwedge_{j \in J} \exists \bar{y}_j.\psi_j(\bar{x}_j) \right) \tag{6.2}$$

by grouping the 'immediate' atomic relational subformulæ $R_{i_0}(\bar{x}_{i_0})$ with $\bar{x}_{i_0} \subseteq \bar{x}$ on one side, and the 'immediate' existentially quantified subformulæ $\exists \bar{y}_j.\psi_j(\bar{x}_j)$ with $\bar{y}_j \subseteq \bar{y}$ and $\bar{x}_j \subseteq \bar{x} \cup \bar{y}_j$ on the other. Then $\bigcup_{i_0 \in I_0} \bar{x}_{i_0} \subseteq \bar{x}$ and we create a node v_0 with bag $\beta(v_0) \stackrel{\text{def}}{=} \bar{x}$ of size $\leq k+1$.

Each formula $\psi_j(\bar{x}_j)$ for $j \in J$ is a subformula of $\psi(\bar{x})$, thus has width at most $k+1$. By induction hypothesis, for each $j \in J$ we obtain a tree decomposition (T_j, β_j) of width at most k for $\mathsf{can}(\psi_j)$. We define T as the tree rooted by v_0 with the roots of the T_j's as children, with $\beta(v)$ mapping to the appropriate bag $\beta_j(v)$ whenever v is a node of T_j. Let us show that (T, β) is a tree decomposition.

- Consider a tuple $(x_{i,1}, \ldots, x_{i,\mathsf{ar}(R)}) \in R_i^{\mathfrak{A}}$ of $\mathfrak{A}$. Then either $i \in I_0$ and therefore $\{x_{i,1}, \ldots, x_{i,\mathsf{ar}(R)}\} \subseteq \beta(n_0)$, or the corresponding atom occurs within one of the subformulæ ψ_j and then there exists a bag of T_j that contains this tuple by induction hypothesis.
- Consider a vertex x of $\mathfrak{A}$. If the set of bags of T that contain x is contained within one of the T_j plus possibly $\beta(n_0)$, i.e., $\{v \in V(T) \mid x \in \beta(v)\} \subseteq \{v \in V(T_j) \mid x \in \beta_j(v)\} \cup \beta(n_0)$ for some $j \in J$, then this set is non-empty and connected by induction hypothesis. Otherwise, x appears in two distinct subtrees T_{j_1} and T_{j_2}.

But then, x appears as a variable in both ψ_{j_1} and ψ_{j_2}, thus necessarily in $\bar{x} = \beta(v_0)$ since all the quantified variables are distinct, which entails that $\{v \in V(T) \mid x \in \beta(v)\}$ is non-empty and connected. □

Example 6.17 (Tree decomposition from a rewriting)**.** Consider again the rewriting $\psi(x_1, x_2)$ of Example 6.15. The decomposition of $\psi(x_1, x_2)$ according to (6.2) yields a single subformula

$$\psi_1(x_1, x_2, x_3) \stackrel{\text{def}}{=} E(x_2, x_3) \wedge \big(\exists x_4.E(x_1, x_4) \wedge E(x_3, x_4)\big) \\ \wedge \big(\exists x_5.E(x_2, x_5) \wedge E(x_3, x_5)\big)$$

such that $\psi(x_1, x_2) = \exists x_3.\psi_1(x_1, x_2, x_3)$. In turn, the decomposition of $\psi_1(x_1, x_2, x_3)$ according to (6.2) yields the two rewritings

$$\psi_2(x_1, x_3, x_4) \stackrel{\text{def}}{=} E(x_1, x_4) \wedge E(x_3, x_4)$$
$$\psi_3(x_2, x_3, x_5) \stackrel{\text{def}}{=} E(x_2, x_5) \wedge E(x_3, x_5)$$

such that $\psi_1(x_1, x_2, x_3) = E(x_2, x_3) \wedge \big(\exists x_4.\psi_2(x_1, x_3, x_4)\big) \wedge \big(\exists x_5.\psi_3(x_2, x_3, x_5)\big)$. The corresponding tree decomposition of the canonical structure of ψ is shown in Figure 6.5; it has indeed width 2.

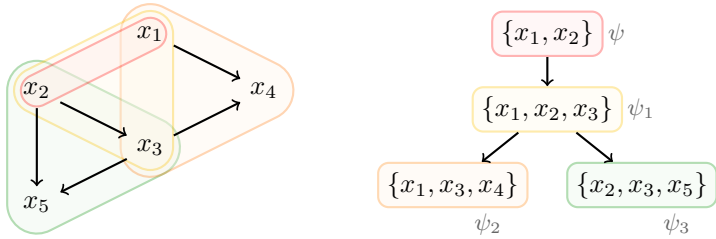


FIGURE 6.5. The tree decomposition of the structure of Figure 6.4 constructed according to the rewriting ψ from Example 6.15. Each node of the tree is annotated (in grey) with the corresponding subformula of ψ from Example 6.17.

You might have already guessed what follows: a converse to Proposition 6.16 holds.

Proposition 6.18. *Let (T, β) be a tree decomposition of width k of a structure $\mathfrak{A}$. Then we can compute in time linear in the size of (T, β) a rewriting of variable width $k + 1$ of $\mathsf{diag}^+(\mathfrak{A})$.*

PROOF. The proof follows from the algebraic construction of **??**. □

Combining propositions 6.16 and 6.18, we have the following characterisation of the tree-width of a structure.

Corollary 6.19 (Rewriting Characterisation of Tree-Width)**.** *A structure $\mathfrak{A}$ has tree-width at most k if and only if $\mathsf{diag}^+(\mathfrak{A})$ has a rewriting of variable width at most $k + 1$.*

6.3.1.2. *Conjunctive Queries of Bounded Tree-Width.* Corollary 6.19 is not the sole corollary of Proposition 6.18: recall that, by Theorem 3.14, we can solve both the evaluation and the

model-checking problem over a finite structure $\mathfrak{A}$ in time $O(|\varphi| \cdot |A|^{w(\varphi)})$ with an adjacency matrix encoding. Define the *tree-width* of a conjunctive query φ as the tree-width of its canonical structure $\mathsf{can}(\varphi)$, and let CQ_k be the class of conjunctive queries of tree-width at most k.

Corollary 6.20. *For all k, $\mathsf{EVAL}(\mathsf{CQ}_k, \mathsf{Fin})$ and $\mathsf{MC}(\mathsf{CQ}_k, \mathsf{Fin})$ can be solved in deterministic time $2^{O(k^3)} \cdot |\varphi| \cdot |A|^{k+1}$ with an adjacency matrix encoding, thus in deterministic polynomial time.*

Proof. Given a conjunctive query $\varphi \in \mathsf{CQ}_k$, compute in time $2^{O(k^3)} \cdot |\mathsf{vars}(\varphi)|$ a tree decomposition of width k of its canonical structure $\mathsf{can}(\varphi)$ using Bodlaender's algorithm (c.f. Fact 6.14). Use this decomposition to compute a rewriting ψ of variable width $k+1$ of φ in linear time by Proposition 6.18, and finally evaluate ψ against the input structure $\mathfrak{A}$ in time $O(|\psi| \cdot |A|^{w(\psi)})$ by Theorem 3.14. As ψ and φ are logically equivalent and of the same size, the result follows. □

6.3.2. Cores of Structures. (Dalmau, Kolaitis, and Vardi, 2002, Corollary 2)(Grohe, 2007, Theorem 3.1) (Flum and Grohe, 2006, Theorem 13.12)

Further Reading

References. Tree-width was defined by Robertson and Seymour (1986a) in the context of their epic multi-article proof of the Graph Minor Theorem (see Section 8.1). Tree-width has however found a large number of applications in multiple fields; for a short survey see (Eppstein, 2025). Most of the material from this chapter is inspired by (Flum and Grohe, 2006, Chapters 11 and 13) and (Kreutzer, 2009).

Computing Tree Decompositions. More efficient algorithms than Bodlaender's (see Fact 6.14) are known if we are content with tree decompositions of somewhat larger width; for instance, there is an algorithm in time $2^{O(k)} \cdot |G|$ building a tree decomposition of width $2k+1$ or rejecting the input if $\mathsf{tw}(G) > k$ (Korhonen, 2022). Recent results by Bonnet (2025)—besides providing a direct reduction from 3SAT to TREE-WIDTH and thus an alternative proof of Fact 6.13—indicate that tree-width is not approximable in its non-parameterised version.

Tree-Width and Conjunctive Queries. Proposition 6.18 was first observed by Kolaitis and Vardi (2000, Lemma 5.2); the full characterisation in terms of rewritings in Corollary 6.19 was stated by Dalmau, Kolaitis, and Vardi (2002, Theorem 7).

Bojańczyk and Pilipczuk, 2016

CHAPTER 7

The Case of Bounded Clique-Width

clique width Courcelle, Makowsky, and Rotics, 2000; Oum, 2008

CHAPTER 8

The Case of Grids

Proposition 8.1 (Tree-Width of Grids)**.** *For all* $k \geq 1$, $\mathsf{tw}(G_{k \times k}) = k$.

(Arenas et al., 2022, Proposition 21.7)
grids Robertson and Seymour, 1986b grids again Seese, 1991

8.1. Graph Minors

graph minors

References

Books and Lecture Notes

Abiteboul, Serge, Richard Hull, and Victor Vianu (1995). *Foundations of Databases*. Addison Wesley. (cit. on pp. xi, 56, 59)

Arenas, Marcelo, Pablo Barceló, Leonid Libkin, Wim Martens, and Andreas Pieris (2022). *Database Theory. Querying Data*. Preliminary version. (cit. on pp. xi, 2, 3, 13, 15, 28, 36, 45, 59, 97)

Arora, Sanjeev and Boaz Barak (2009). *Computational Complexity. A Modern Approach*. Cambridge University Press. (cit. on pp. 22, 25, 45, 47, 70)

Baier, Christel and Joost-Pieter Katoen (2008). *Principles of Model Checking*. MIT Press (cit. on p. 40)

Chang, Chen C. and Howard J. Keisler (1990). *Model Theory*. 3rd ed. Studies in Logic and the Foundations of Mathematics 73. North Holland. (cit. on p. 11)

Comon, Hubert, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi (2008). *Tree Automata. Techniques and Applications*. (cit. on p. xi)

Courcelle, Bruno and Joost Engelfriet (2012). *Graph Structure and Monadic Second Order Logic. A Language-Theoretic Approach*. Encyclopedia of mathematics and its applications 138. Cambridge University Press. (cit. on p. 11)

Cygan, Marek, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh (2015). *Parameterized Algorithms*. Springer. (cit. on pp. 39, 46)

Downey, Rod G. and Michael R. Fellows (1999). *Parameterized Complexity*. Monographs in Computer Science. Springer. (cit. on p. 46)

——— (2013). *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer. (cit. on pp. xi, 46)

Flum, Jörg and Martin Grohe (2006). *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer. (cit. on pp. xi, 39, 40, 45, 46, 59, 89, 93)

Greenlaw, Raymond, H. James Hoover, and Walter L. Ruzzo (1995). *Limits to Parallel Computation. P-Completeness Theory*. Oxford University Press. (cit. on p. 46)

Hodges, Wilfried (1997). *A Shorter Model Theory*. Cambridge University Press (cit. on pp. 11, 51)

Immerman, Neil (1999). *Descriptive Complexity*. Springer. (cit. on p. 46)

Kreutzer, Stephan (2009). *Algorithmic Meta-Theorems*. Tech. rep. TR09-147. Electronic Colloquium on Computational Complexity. (cit. on pp. xi, 93)

Libkin, Leonid (2004). *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer. (cit. on pp. xi, 11, 13, 45, 46)

Marker, David (2002). *Model Theory. An Introduction*. Graduate Texts in Mathematics 217. Springer. (cit. on pp. 11, 51)

Toruńczyk, Szymon (2022). *Lectures on Finite Model Theory*. Lecture notes. (cit. on pp. xi, 13)

Vollmer, Heribert (1999). *Introduction to Circuit Complexity. A Uniform Approach*. Texts in Theoretical Computer Science. Springer. (cit. on pp. 32, 35, 46)

Articles

Ajtai, Miklós (1983). "Σ^1_1-formulæ on finite structures". In: *Annals of Pure and Applied Logic* 24(1), pp. 1–48. (cit. on p. 34)

Arnborg, Stefan, Derek G. Corneil, and Andrzej Proskurowski (1987). "Complexity of finding embeddings in a k-tree". In: *SIAM Journal on Algebraic Discrete Methods* 8(2), pp. 277–284. (cit. on p. 89)

Barrington, David A. Mix, Neil Immermann, and Howard Straubing (1990). "On uniformity within NC^1". In: *Journal of Computer and System Sciences* 41(3), pp. 274–306. (cit. on pp. 35, 46)

Beeri, Catriel, Ronald Fagin, David Maier, and Mihalis Yannakakis (1983). "On the desirability of acyclic database schemes". In: *Journal of the ACM* 30(3), pp. 479–513. (cit. on p. 59)

Belazzougui, Djamal, Fabiano C. Botelho, and Martin Dietzfelbinger (2009). "Hash, displace, and compress". In: *Proceedings of ESA'09*. Lecture Notes in Computer Science 5757, pp. 682–693. Springer. (cit. on p. 70)

Bodirsky, Manuel (2008). "Constraint satisfaction problems with infinite templates". In: *Complexity of Constraints*. Lecture Notes in Computer Science 5250. Springer, pp. 196–228. (cit. on p. 60)

Bodlaender, Hans L. (1996). "A linear time algorithm for finding tree-decompositions of small treewidth". In: *SIAM Journal on Computing* 25(6), pp. 1305–1317. (cit. on pp. 89, 93, 107)

Bojańczyk, Mikołaj and Michał Pilipczuk (2016). "Definability equals recognizability for graphs of bounded treewidth". In: *Proceedings of LICS'16*, pp. 407–416. IEEE. (cit. on p. 93)

Bonnet, Édouard (2025). "Treewidth inapproximability and tight ETH lower bound". In: *Proceedings of STOC'25*. To appear. (cit. on p. 93)

Bulatov, Andrei A. (2017). "A dichotomy theorem for nonuniform CSPs". In: *Proceedings of FOCS'17*, pp. 319–330. IEEE. (cit. on p. 60)

Cai, Liming, Jianer Chen, Rodney G. Downey, and Michael R. Fellows (1997). "On the parameterized complexity of short computation and factorization". In: *Archive for Mathematical Logic* 36, pp. 321–337. (cit. on p. 59)

Calò, A. and Johann A. Makowsky (1992). "The Ehrenfeucht-Fraïssé games for transitive closure". In: *Proceedings of LFCS'92*. Lecture Notes in Computer Science 620, pp. 57–68. Springer. (cit. on p. 89)

Carmeli, Nofar and Luc Segoufin (2023). "Conjunctive queries with self-joins, towards a fine-grained enumeration complexity analysis". In: *Proceedings of PODS'23*, pp. 277–289. ACM. (cit. on p. 59)

Chandra, Ashok K., Dexter C. Kozen, and Larry J. Stockmeyer (1981). "Alternation". In: *Journal of the ACM* 28(1), pp. 114–133. (cit. on pp. 26, 45)

Chandra, Ashok K. and Philip M. Merlin (1977). "Optimal implementation of conjunctive queries in relational data bases". In: *Proceedings of STOC'77*, pp. 77–90. ACM. (cit. on p. 59)

Codd, Edgar F. (1972). *Relational Completeness of Database Sublanguages*. Tech. rep. RJ 987 (#17041). IBM Research Laboratory (cit. on pp. 29, 45, 49, 50, 109)

Courcelle, Bruno (1990). "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs". In: *Information and Computation* 85(1), pp. 12–75. (cit. on pp. xi, 83, 90)

Courcelle, Bruno, János A. Makowsky, and Udi Rotics (2000). "Linear time solvable optimization problems on graphs of bounded clique-width". In: *Theory of Computing Systems* 33, pp. 125–150. (cit. on p. 95)

Dalmau, Víctor, Phokion G. Kolaitis, and Moshe Y. Vardi (2002). "Constraint satisfaction, bounded treewidth, and finite-variable logics". In: *Proceedings of CP'02*. Lecture Notes in Computer Science 2470, pp. 310–326. Springer. (cit. on p. 93)

Dowling, William F. and Jean H. Gallier (1984). "Linear-time algorithms for testing the satisfiability of propositional Horn formulæ". In: *The Journal of Logic Programming* 1(3), pp. 267–284. (cit. on p. 70)

Downey, Rod G. and Michael R. Fellows (1995a). "Fixed-parameter tractability and completeness I: Basic results". In: *SIAM Journal on Computing* 24(4), pp. 873–921. (cit. on p. 46)

——— (1995b). "Fixed-parameter tractability and completeness II: On completeness for W[1]". In: *Theoretical Computer Science* 141(1–2), pp. 109–131. (cit. on p. 59)

Downey, Rodney G., Michael R. Fellows, and Udayan Taylor (1996). "The parameterized complexity of relational database queries and an improved characterization of W[1]". In: *Proceedings of DMTCS'96*, pp. 194–213. Springer (cit. on p. 46)

Durand, Arnaud (2020). "Fine-grained complexity analysis of queries: from decision to counting and enumeration". In: *Proceedings of PODS'20*, pp. 331–346. ACM. (cit. on pp. 47, 59)

Eppstein, David (2025). "What is… treewidth?" In: *Notices of the AMS* 72(2), pp. 172–175. (cit. on p. 93)

Fagin, Ronald (1983). "Degrees of acyclicity for hypergraphs and relational database schemes". In: *Journal of the ACM* 30(3), pp. 514–550. (cit. on p. 59)

Feder, Tomás and Moshe Y. Vardi (1993). "Monotone monadic SNP and constraint satisfaction". In: *Proceedings of STOC'93*, pp. 612–622. ACM. (cit. on p. 60)

Flum, Jörg and Martin Grohe (2001). "Fixed-parameter tractability, definability, and model-checking". In: *SIAM Journal on Computing* 31(1), pp. 113–145. (cit. on p. 46)

Furst, Merrick, James B. Saxe, and Michael Sipser (1984). "Parity, circuits, and the polynomial-time hierarchy". In: *Mathematical Systems Theory* 17(1), pp. 13–27. (cit. on p. 34)

Gottlob, Georg, Nicola Leone, and Francesco Scarcello (2001). "The complexity of acyclic conjunctive queries". In: *Journal of the ACM* 48(3), pp. 431–498. (cit. on p. 59)

Grohe, Martin (2007). "The complexity of homomorphism and constraint satisfaction problems seen from the other side". In: *Journal of the ACM* 54(1), art. 1. (cit. on p. 93)

Gurevich, Yuri and Harry R. Lewis (1984). "A logic for constant-depth circuits". In: *Information and Control* 61(1), pp. 65–74. (cit. on p. 46)

Haase, Christoph, Shankara Narayanan Krishna, Khushraj Madnani, Om Swostik Mishra, and Georg Zetzsche (2024). "An efficient quantifier elimination procedure for Presburger arithmetic". In: *Proceedings of ICALP'24*. Leibniz International Proceedings in Informatics (LIPIcs) 297, art. 142. LZI. (cit. on p. 69)

Hell, Pavol and Jaroslav Nešetřil (1990). "On the complexity of H-coloring". In: *Journal of Combinatorial Theory, Series B* 48(1), pp. 92–110. (cit. on p. 60)

Immerman, Neil (1982). "Upper and lower bounds for first order expressibility". In: *Journal of Computer and System Sciences* 25(1), pp. 76–98. (cit. on p. 46)

——— (1989). "Expressibility and parallel complexity". In: *SIAM Journal on Computing* 18(3), pp. 625–638. (cit. on p. 36)

Kolaitis, Phokion G. (2007). "Reflections on finite model theory". In: *Proceedings of LICS'07*, pp. 257–269. IEEE. (cit. on p. 13)

Kolaitis, Phokion G. and Moshe Y. Vardi (2000). "Conjunctive-query containment and constraint satisfaction". In: *Journal of Computer and System Sciences* 61(2), pp. 302–332. (cit. on pp. 60, 93)

Korhonen, Tuukka (2022). "A single-exponential time 2-approximation algorithm for treewidth". In: *Proceedings of FOCS'21*, pp. 184–192. (cit. on p. 93)

Libkin, Leonid (2009). "The finite model theory toolbox of a database theoretician". In: *Proceedings of PODS'09*, pp. 65–76. ACM. (cit. on p. 13)

Marx, Dániel (2013). "Tractable hypergraph properties for constraint satisfaction and conjunctive queries". In: *Journal of the ACM* 60(6), art. 42. (cit. on p. 60)

Mottet, Antoine, Tomáš Nagy, and Michael Pinsker (2024). "An order out of nowhere: a new algorithm for infinite-domain CSPs". In: *Proceedings of ICALP'24*. Leibniz International Proceedings in Informatics (LIPIcs) 297, art. 148. LZI. (cit. on p. 60)

Oum, Sang-il (2008). "Approximating rank-width and clique-width quickly". In: *ACM Transactions on Algorithms* 5(1), art. 10. (cit. on p. 95)

Papadimitriou, Christos H. and Mihalis Yannakakis (1999). "On the complexity of database queries". In: *Journal of Computer and System Sciences* 58(3), pp. 407–427. (cit. on p. 59)

Robertson, Neil and Paul D. Seymour (1986a). "Graph minors. II. Algorithmic aspects of tree-width". In: *Journal of Algorithms* 7(3), pp. 309–322. (cit. on p. 93)

——— (1986b). "Graph minors. V. Excluding a planar graph". In: *Journal of Combinatorial Theory, Series B* 41(1), pp. 92–114. (cit. on p. 97)

Seese, Detlef (1991). "The structure of the models of decidable monadic theories of graphs". In: *Annals of Pure and Applied Logic* 53(2), pp. 169–195. (cit. on p. 97)

Stockmeyer, Larry and Uzi Vishkin (1984). "Simulation of parallel random access machines by circuits". In: *SIAM Journal on Computing* 13(2), pp. 409–422. (cit. on p. 46)

Stockmeyer, Larry J. (1976). "The polynomial-time hierarchy". In: *Theoretical Computer Science* 3(1), pp. 1–22. (cit. on pp. 22, 23, 45)

Strozecki, Yann (2019). "The algorithmics column. Enumeration complexity". In: *Bulletin of the EATCS* 129. (cit. on p. 47)

Tarjan, Robert E. and Mihalis Yannakakis (1984). "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs". In: *SIAM Journal of Computing* 13(3), pp. 566–579. (cit. on p. 56)

Tarski, Alfred (1935). "Der Wahrheitsbegriff in den Formalisierten Sprachen". In: *Studia Philosophica* 1, pp. 261–405 (cit. on p. 11)

Trakhtenbrot, Boris A. (1950). "The impossibility of an algorithm for the decidability problem on finite classes". In: *Proceedings of the USSR Academy of Sciences* 70(4). Russian Original: Трахтенброт, Б. А. (1950). "Невозможность алгорифма для проблемы разрешимости на конечных классах." In: *Доклады Академии наук СССР* 70(4), pp. 569–572 (cit. on pp. 13, 109)

Vardi, Moshe Y. (1982). "The complexity of relational query languages (extended abstract)". In: *Proceedings of STOC'82*, pp. 137–146. ACM. (cit. on p. 45)

———— (1995). "On the complexity of bounded-variable queries (extended abstract)". In: *Proceedings of PODS'95*, pp. 266–276. ACM. (cit. on p. 46)

Vianu, Victor (1997). "Databases and finite-model theory". In: *Proceedings of a DIMACS workshop on Descriptive Complexity and Finite Models*, pp. 97–148. AMS (cit. on p. 13)

Yannakakis, Mihalis (1981). "Algorithms for acyclic database schemes". In: *Proceedings of VLDB'81*, pp. 82–94. IEEE (cit. on pp. 49, 56, 57, 59, 83, 84, 90, 91, 107)

Zhuk, Dmitry (2017). "A proof of the CSP dichotomy conjecture". In: *Proceedings of FOCS'17*, pp. 331–342. IEEE. (cit. on p. 60)

List of Symbols

Index