Matching under Side Conditions in Description Logics

Diplomarbeit im Fach Informatik am
Lehr- und Forschungsgebiet Theoretische Informatik der
Rheinisch-Westfälischen Technischen Hochschule Aachen
Professor Dr.-Ing. Franz Baader

vorgelegt von Sebastian Philipp Brandt

Aachen, im August 2000

Matching under Side Conditions in Description Logics

Diplomarbeit
Ausgegeben und betreut von Professor Dr.-Ing. Franz Baader
Erstgutachter: Professor Dr.-Ing. Franz Baader
Zweitgutachter: Professor Dr. Wolfgang Thomas

Verfasser
Sebastian Philipp Brandt
Matrikelnummer 200 553
Geboren am 25. April 1974
Satz: TEX/LATEX

Erklärung
Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe. Aachen, den 21. August 2000.

Meiner Mutter und meinem Vater gewidmet.
Alles, was ich bin,
bin ich durch Euch.

A UN POETA MENOR DE 1899
Dejar un verso para la hora triste
Que en el confín del día nos acecha,
Ligar tu nombre a su doliente fecha
De oro y de vaga sombra. Eso quisiste.
¡Con qué pasión, al declinar el día,
Trabajarías el extraño verso
Que, hasta la dispersión del universo,
La hora de extraño azul confirmaría!
No sé si lo lograste ni siquiera.
Vago hermano mayor, si has existido,
Pero estoy solo y quiero que el olvido
Restituya a los días tu ligera
Sombra para este ya cansado alarde
De unas palabras en que esté la tarde.

EINEM MINDEREN DICHTER VON 1899
Der Stunde, die uns trist am Tagesrand
belauert, eine Zeile hinterlassen,
deinen Namen an ihre sieche Zeit
aus Gold und Dunkel heften war dein Wunsch.
Mit welcher Leidenschaft hast du an der Neige
des Abends an diesem sonderbaren Vers gefeilt,
daß er, bis zur Auflösung des Universums,
die sonderbare blaue Stunde berge!
Ich weiß nicht, ob er dir gelang, nicht einmal,
du ungewisser älterer Bruder, ob du lebtest,
aber ich bin allein und wünschte, das
Vergessen gäbe den Tagen deinen leichten
Schatten zurück für diese müde Reihe
von Wörtern, daß in ihnen dieser Abend sei.

JORGE LUIS BORGES

Danksagung

Mit großer Freude versehe ich die Pflicht, hier all denen meinen Dank auszusprechen, die Anteil hatten am Zustandekommen der vorliegenden Arbeit. Insbesondere danke ich Professor Dr. Franz Baader für seine Betreuung und für das Vertrauen, das er in mich und in das Gelingen dieser Arbeit gesetzt hat. Es wird mir ein Anliegen sein zu zeigen, daß dieses Vertrauen nicht unnütz investiert war. Meinem Mentor Professor Dr. Klaus Indermark verdanke ich, daß mein Studium einen sicheren Weg genommen hat und daß die RWTH Aachen sich mir von Beginn an von einer persönlichen und verantwortungsvollen Seite gezeigt hat. Für die Geduld, für die Unterstützung und für die unzähligen Gespräche über mein Thema danke ich Dipl.-Inform. Ralf Küsters, dessen Engagement diese Arbeit mitbestimmt hat.
Ich danke Neil Stewart M.A. für seinen Rat als "native speaker" und für seine Bereitschaft, auch einen so derart "off-topic" gelegenen Aufsatz mit der Sorgfalt zu behandeln, die ihn auszeichnet. Vielen Dank an Sebastian Niesen, der mir als Administrator den Luxus ermöglicht hat, mich gänzlich auf diese Arbeit zu konzentrieren, ohne Betriebssysteme warten zu müssen. Für ihre großzügige Unterstützung danke ich außerdem Mathias Christoph Brandt und Young-Im Yang, ohne deren Fürsorge die zurückliegenden Monate viel entbehrungsreicher verlaufen wären.

Sebastian Philipp Brandt

Contents

CHAPTER 1

INTRODUCTION

Description Logics (DL) form a category of knowledge-representation formalisms used to represent terminological knowledge of a given application domain in a structured and well-defined way. As common characteristics, they employ concept-*descriptions* for the representation of notions relevant in the application domain, and provide a model-theoretic semantics closely related to first-order *logics*. Concept descriptions are built from atomic concepts and atomic roles, i.e. from unary and binary predicates respectively, using concept constructors provided by the DL language. Atomic concepts and concept descriptions represent sets of individuals, whereas atomic roles represent binary relations between individuals [BN98a]. Consider the following example, which is inspired by [BN98a]. Assuming an atomic concept Human representing human beings, an atomic concept Female for all female beings, and an atomic role hasChild specifying parent-child relations, we can represent the general concepts of women and women having only daughters:

$$\begin{aligned} \mathit{Woman} &:= \mathsf{Human} \sqcap \mathsf{Female} \\ W &:= \mathit{Woman} \sqcap \forall \mathsf{hasChild}.\mathit{Woman} \end{aligned}$$

The symbol ($\sqcap$) stands for the conjunction of concept descriptions. Thus, a *Woman* is a female human being and W denotes the concept of women such that all their children are again women. The example also illustrates how concept descriptions are built up from atomic concepts.
In this work we will be concerned with the DL language $\mathcal{ALN}$, which also allows the imposition of number restrictions on atomic roles. If a number restriction for some role is included in a concept description, then certain limitations regarding the number of successors in respect to this role are imposed on all instances of this concept description. The idea is illustrated by extending the above example. In the definition of the concept description W it is not stated that there actually are any daughters. When specifying the general concept of a mother, however, we do not only require every child to be human, but we especially want to ensure that at least one such child exists. Utilizing number restrictions, this notion can be represented by the following concept description.

$$\begin{aligned} \mathit{Mother} &:= \mathit{Woman} \sqcap \forall \mathsf{hasChild}.\mathsf{Human} \sqcap (\geq 1\ \mathsf{hasChild}) \\ M &:= \mathit{Mother} \sqcap \forall \mathsf{hasChild}.(\leq 0\ \mathsf{hasChild}) \end{aligned}$$

Every individual represented by *Mother* therefore is in hasChild-relation to at least 1 other individual represented by Human, i.e. there exists at least one daughter or son. The definition of the concept description M is interesting, because here a number restriction occurs inside a role restriction. M represents the concept of a mother who is not a grandmother. This holds, since all the children of individuals represented by M are required to

have at most 0 children themselves, i.e. there are no children in the second generation. Observe that the at-least restriction in M does not interfere with the at-most restriction included in *Mother*, since it occurs on a different level in regard to the role hasChild. A formal definition of the language $\mathcal{ALN}$ including all available constructs will be provided in Chapter 3. The above examples may suffice at this point in order to give a rough impression of the capabilities of DL languages.

Subsumption and *equivalence* are distinguished as relations between concept descriptions of a DL language. If one concept description is subsumed by a second one, then these two are in subconcept–superconcept relation. Thus, the second is a superconcept or generalization of the first one. In this case, the individuals represented by the first concept description always form a subset of those of the second one. For instance in the above example the concept description M is subsumed by *Mother*, which itself is subsumed by *Woman*, since every mother is a woman. Concept descriptions are regarded as equivalent if they always represent the same set of individuals.

The subsumption-relation induces a hierarchy on the set of concept descriptions which is desirable for structuring the notions relevant for an application domain. On the other hand, identifying equivalent concept descriptions allows to avoid redundancies when augmenting an existing set of concept descriptions [BN98a]. Nevertheless, subsumption- and equivalence-relations must be decidable in order to take advantage of them in DL systems. Deciding subsumption or equivalence of concept descriptions are standard inference problems which have been examined for a variety of DL languages. For many of them, upper and lower complexity bounds have been obtained and matching algorithms have been proposed (e.g. [HNSS90, HB91, DLNN91]. See [BKBM99] for further references).

However, when DL languages are employed for large-scale knowledge bases, standard inference algorithms do not perform satisfactory for building and maintaining purposes. It has been shown in [MPS98] that non-standard inferences like *learning* and *matching* can be used to improve this. In this work we will restrict our attention to the latter.

In order to address matching we need to introduce *concept patterns*. These extend the notion of concept descriptions by allowing for *variables*, which can be substituted by concept descriptions. Matching a concept pattern against a concept description means finding a substitution for the occurring variables such that both expressions become equivalent. This is called matching modulo equivalence. Matching modulo subsumption on the other hand aims at merely making the concept pattern subsume the concept description. Consider the following example of a concept pattern, which again refers to the concept descriptions introduced above:

$$P := \mathit{Woman} \sqcap \forall \mathsf{hasChild}.X$$

When matching (modulo equivalence) the concept pattern P against the concept description W of our example the variable X is substituted by an expression equivalent to *Woman*. For matching modulo subsumption, assigning X with Human would already be sufficient. We shall see later on that it is desirable to find substitutions which are minimal in respect to subsumption, ensuring that the obtained result is as specific as possible.

The idea of matching can be refined by admitting *side conditions* which impose further constraints on the substitution sought. Side conditions can be defined for every variable occurring in a concept pattern and demand that the solution for this variable be subsumed by another concept pattern. In this way, side conditions form a system of subsumption conditions, which can either be *acyclic* or *cyclic*. We distinguish *non-strict* and *strict* side conditions, depending on whether subsumption or strict subsumption is required. With the help of side conditions it is possible to avoid trivial matches occurring as solutions to matching problems or to find solutions at a certain position in the concept hierarchy.

1.1 The motive for matching

Matching was motivated by the idea of pruning large concept descriptions which are likely to occur in real-world knowledge representation systems based on DL languages. Pruning means that, instead of printing concept descriptions in full length, only those aspects are printed which are relevant under current circumstances. For this task matching algorithms can be used, as the following example may illustrate.

Consider an application domain where description logics are employed to represent the properties of certain components interacting with each other for some purpose. Under certain circumstances only the dependency of a component on others might be of interest, whereas all the other properties are irrelevant. Instead of manually retrieving the relevant details in a concept description C defining a certain type of component, one could match C against a concept pattern D of the following form.

$$D := \forall \mathsf{dependsOnService}.X \sqcap \forall \mathsf{providesService}.Y$$

Provided that the dependency relations in such a context are represented by atomic roles like dependsOnService and providesService the matching result returns exactly the relevant aspects of the component C. It was pointed out that implementing pruning strategies on user interface level of knowledge representation systems entails disadvantages in comparison to including them in the underlying DL language. Especially, concept patterns like D used to provide a pruned view of concept descriptions can be stored, organized, and re-used [BKBM99].

As an extension of the above example, side-conditions could be used to restrict the solutions obtained when matching the concept description C against the concept pattern D. Assume that the atomic concept ServiceTypeA represents a certain subclass of services provided by our components. By including a non-strict side condition of the form

$$Y \sqsubseteq^{?} \mathsf{ServiceTypeA}$$

in the matching problem, only those services provided by C are returned, which are subsumed by ServiceTypeA, i.e. we obtain only services of type A. In this case, side conditions are used to obtain more specific results for matching problems. This is especially useful for matching modulo subsumption, where trivial solutions exist for every solvable matching problem. It should be noted, however, that side conditions can also be utilized to prevent solutions to matching problems from becoming too specific. Consequently, side conditions provide a powerful means for the refinement of matching problems.

Matching algorithms have already been employed successfully in professional and academic knowledge representation systems. In the Classic system developed at AT&T [BMS+91], matching is used to prune irrelevant information in the context of explanation facilities designed to make deductions explicit to the user. Another example comes from the domain of process engineering [BS96], where there are plans to utilize matching in order to avoid introducing redundancies in very large knowledge bases maintained by several persons over a longer period of time. Apart from these applications, matching can also be used when integrating knowledge bases [BK00b].

In [BKBM99], Baader, Küsters, Borgida, and McGuinness have proposed algorithms to solve matching problems without side conditions in $\mathcal{ALN}$ and three of its sublanguages, namely $\mathcal{FL}_0$, $\mathcal{FL}_\bot$, and $\mathcal{FL}_\neg$. These are introduced mainly for didactic reasons, allowing to develop the solution for $\mathcal{ALN}$ step by step. It should be noted that positive results in $\mathcal{ALN}$ are not automatically inherited by the sublanguages, which makes it necessary to consider each sublanguage individually. The authors have also given proofs regarding the computational complexity of these algorithms, showing that solutions are computed

in polynomial time. Nevertheless, the properties of one construct used within these proofs have been stated without proof.

The objective of this work is twofold. Firstly, we will confirm the results proposed for matching without side conditions by giving a formal proof of the properties used informally in [BKBM99]. Secondly, we will show how matching problems with non-strict side conditions can be solved in polynomial time as well. The following section will give a brief overview of the respective chapters.

1.2 The structure of this work

Chapter 2 introduces basic notions related to formal languages and finite automata. Many of the properties proposed in this context may appear very familiar, but are of crucial importance for our reasoning in the following chapters. In Chapter 3, description logics and matching problems are introduced formally.

Chapter 4 is concerned with the results on matching without side conditions obtained in [BKBM99] in recourse to an intuitive definition of so-called "treelike automata". We propose a formal definition for them and examine their properties in detail. This will allow us to confirm the respective results, i.e. matching problems without side conditions can be solved in polynomial time in $\mathcal{ALN}$ and its sublanguages. In Chapter 5, side conditions are taken into consideration. Two approaches are discussed to reduce matching problems with acyclic side conditions to such without them. A straightforward strategy originally mentioned in [BKBM99] is shown to fail, while an alternative one, utilizing finite automata, will succeed. Both approaches, however, are limited to acyclic side conditions.

This is overcome in Chapter 6, where we present a solution comprising a fixed point algorithm applicable to acyclic as well as cyclic side conditions. It will be shown that by this approach matching problems with non-strict side conditions in $\mathcal{ALN}$ as well as its sublanguages can be solved in polynomial time. Proving these claims will be simplified by the introduction of normal forms for concept descriptions, a representation which is unique with respect to equivalence.

In the last chapter we give a summary and very briefly mention two open problems for which the results of this work might be valuable. These problems are matching under strict side conditions and matching in description logics other than $\mathcal{ALN}$.

CHAPTER 2

PRELIMINARIES

In this chapter we introduce basic notions relating to sets, formal languages, and finite automata. Furthermore, some properties of finite automata are discussed, which will prove useful in the later chapters.

Let us first explain some typographic conventions throughout this work. The end of the body of every definition, of every proof, and of every example is indicated by a box (□) at the right-hand side of the column. The notions newly introduced in a definition are set in *italic* type. If the assertion of a lemma comprises an enumeration of several claims, then black triangles (▶) at the left-hand side of the column are used to structure the proof accordingly. When several cases are distinguished in a proof, light triangles (▷) are employed to indicate the beginning of every case. We hope that these visual markers make reading more convenient.

Throughout this work, the word "iff" is used as an equivalent to "if and only if". It should also be noted that we include 0 in the set of natural numbers, i.e. $\mathbb{N}$ is defined as $\mathbb{N} := \{0, 1, \ldots\}$. Our first definition specifies our notation for the power set and the cardinality of sets.

Definition 2.1 Notation for sets
For every set S, denote by $\mathfrak{P}(S)$ the *power set of* S, i.e. $\mathfrak{P}(S) := \{T \mid T \subseteq S\}$. The *cardinality of* S is denoted by $|S|$. □

2.1 Formal languages

We are now ready to introduce formal languages and discuss some of their properties. We will make use of the notation introduced in [HU80], where the subject is studied in depth.

Definition 2.2 Formal languages
Let Σ be a finite nonempty set. Σ is called *alphabet* and its elements are called *characters*, which we regard as atomic symbols. A finite sequence of characters is called a *word over* Σ. The length of a word w is denoted by $|w|$. The word consisting of 0 characters is denoted by ε, the *empty word*. For two words w and w', w' is called a *prefix of* w iff there exists another word w'' such that $w = w'w''$. In this case, w'' is called a *suffix of* w and w is called a *continuation of* w'. The notion of prefixes induces a strict order over the set of words over Σ in the following way: Two words w and v are in *strict prefix order* (denoted by $w <_{pr} v$), iff w is a prefix of v, and v is not equal to w.
Denote by '$\cdot$' the *concatenation* of words, i.e. the expression $w \cdot v$ represents the character sequence wv for every word w and v. The empty word is neutral in respect to the concatenation.

A set of words over Σ is called a *(formal) language over* Σ. A language is called *regular*, if it can be represented by a grammar of type 3 in the Chomsky Hierarchy. The notion of concatenation is extended to languages in the following way: For languages L, L', the concatenation $L{\cdot}L'$ is defined by concatenating all possible pairs of words, which yields $L{\cdot}L' := \{w{\cdot}w' \,|w \in L, w' \in L'\}$. For the iterated concatenation, the following notation is defined inductively. For every language L and for $n \in \mathbb{N}$, define:

$$L^0 := \{\varepsilon\}$$
$$L^{n+1} := L{\cdot}L^n$$

For every language L, the expression L^* is defined as $L^* := \bigcup_{n\in\mathbb{N}} L^n$. Similarly, L^+ is defined by excepting the case $n = 0$, i.e. $L^+ := L^* \setminus L^0$. The Σ^*-*closure* of L is defined by $L{\cdot}\Sigma^*$. □

Note that the alphabet Σ can be regarded as a language itself. Every word over Σ is an element of Σ^* and every language over Σ is a subset of Σ^*.
For formal languages, the operations left and the right quotient are defined as follows:

Definition 2.3 Left and right quotients
Let L be a language over the alphabet Σ, let $w \in \Sigma^*$ be a word over Σ. The *left quotient* of L in respect to w is defined as $w^{-1}{\cdot}L := \{v \in \Sigma^* | wv \in L\}$. The *right quotient* of L in respect to w is defined as $L{\cdot}w^{-1} := \{v \in \Sigma^* | vw \in L\}$. □

Thus, if a word in a formal language L begins with w, then the remainder of this word is an element of the left-quotient of w and L. The idea for the right-quotient is analogous. The cardinality $|L|$ denotes the number of words contained in a formal language L. To include the length of words into a measure for L, we introduce the notion of the size of formal languages:

Definition 2.4 Size of formal languages
Let $|\cdot|$ be the ordinary length-function for words over Σ. For every finite language $L \subseteq \Sigma^*$, define the *size of* S by:

$$\|L\| := \sum_{w\in L} |w| \qquad \square$$

The size $\|L\|$ of a language L corresponds to the amount of storage necessary to represent L explicitly. Thus, it is an appropriate measure when studying the computational complexity of algorithms over formal languages.

2.2 Finite automata

Finite automata are well known constructs for the representation of regular languages. We first address nondeterministic finite automata and then define the deterministic case as a specialization. Finite automata are studied exhaustively in [HU80], where our basic definitions originate.

2.2.1 Nondeterministic finite automata

Definition 2.5 Nondeterministic finite automata
Let Σ be a finite alphabet. A *nondeterministic finite automaton (NFA)* $\mathcal{B}$ *over* Σ is defined as $\mathcal{B} := \langle Q, \Sigma, \delta, q_0, F\rangle$, where

- $Q \neq \emptyset$ denotes a finite set of *states*,
- $F \subseteq Q$ is the set of *accepting states*,

- $q_0 \in Q$ denotes the *initial state* and
- $\delta\colon Q \times \Sigma \to \mathfrak{P}(Q)$ is a *non-deterministic transition function.*

The transition function is extended to words of arbitrary length by the notion of the *extended transition function* $\hat{\delta}$. For every $q \in Q$, $w \in \Sigma^*$, and for every $s \in \Sigma$, define $\hat{\delta}$ inductively as follows:

$$\begin{aligned} \hat{\delta}\colon Q \times \Sigma^* &\to \mathfrak{P}(Q) \\ q\,,\,\varepsilon &\mapsto \{q\} \\ q\,,\,ws &\mapsto \{p \mid \exists r \in \hat{\delta}(q,w)\colon p \in \delta(r,s)\} \end{aligned}$$

The *language accepted by* $\mathcal{B}$ now can be defined as $L(\mathcal{B}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0,w) \cap F \neq \emptyset\}$. The size $|\mathcal{B}|$ of an automaton $\mathcal{B}$ is defined by the number of states it has, i.e. $|\mathcal{B}| := |Q|$. □

Since we will employ automata for the representation of formal languages, appropriate operations are necessary to compute the complement, the intersection, and the union of nondeterministic finite automata. Note that the complement cannot be computed efficiently in the nondeterministic case. However, the other operations can be realized in polynomial time. The following definition provides a construction for the intersection-automaton of two given automata:

Definition 2.6 Product automata
For nondeterministic finite automata $\mathcal{B}_i := \langle Q_i, \Sigma, q_{0i}, \delta_i, F_i\rangle$ $(i \in \{1,2\})$ with disjoint sets of states, define the product automaton of $\mathcal{B}_1$ and $\mathcal{B}_2$ as follows:
$\mathcal{B}_1 \cap \mathcal{B}_2 := \langle Q_1 \times Q_2, (q_{01}, q_{02}), \delta, F_1 \times F_2\rangle$ with: For all $(q_1,q_2) \in Q_1 \times Q_2$, and for every $s \in \Sigma$, the transition function δ is defined by: $\delta((q_1,q_2),s) := (\delta_1(q_1,s), \delta_2(q_2,s))$. □

The product automaton simply runs both input automata in parallel and accepts the input, iff both automata independently accept it. The correctness of this construction is stated in the next lemma. We omit a proof, since the results below are probably well known.

Lemma 2.7 Properties of product automata
Let $\mathcal{B}_1, \mathcal{B}_2 \in \mathit{NFA}(\Sigma)$, where $\mathcal{B}_i := \langle Q_i, \Sigma, q_{0i}, \delta_i, F_i\rangle$ for $i \in \{1,2\}$. Let $Q_1 \cap Q_2 = \emptyset$. Then

1. $L(\mathcal{B}_1 \cap \mathcal{B}_2) = L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$
2. $|\mathcal{B}_1 \cap \mathcal{B}_2|$ is polynomial in $|\mathcal{B}_1|$ and $|\mathcal{B}_2|$.
3. $\mathcal{B}_1 \cap \mathcal{B}_2$ can be computed in polynomial time in $|\mathcal{B}_1|$ und $|\mathcal{B}_2|$.

Recall that computing the union of finite automata is particularly simple in the nondeterministic case. Provided disjoint sets of states we can define the union of n automata by merely adding a new initial state connected to the n former initial states by ε-transitions. This construction is employed in Kleene's Theorem. The size of the resulting automaton exceeds the sum of the sizes of the original automata only by a constant.
Automata are intended not only as a representation of formal languages, but also as a means of deciding certain properties of them. Later on in this work, especially two questions must be answered efficiently. Firstly, is the language accepted by a given automaton empty; and secondly, is a certain word contained in this language. Both problems in fact can be decided in polynomial time for nondeterministic automata. The following lemma is stated without proof, since its assertions are well known.

Lemma 2.8 Decision problems
Let $\mathcal{B} \in NFA(\Sigma)$ be an *NFA* over Σ and let w be a word over Σ. Then
1. $L(\mathcal{B}) =^? \emptyset$, i.e. the $\emptyset$-problem, can be decided in polynomial time in $|\mathcal{B}|$.
2. $w \in^? L(\mathcal{B})$, i.e. the word-problem, can be decided in polynomial time in $|\mathcal{B}|$ and $|w|$.

Regular languages can be represented by nondeterministic automata. We will now see that such a representation can be computed in polynomial time. Given a finite language, we can efficiently construct an appropriate nondeterministic automaton. This is solved similar to the construction of Kleene's Theorem. Nevertheless, we can avoid introducing intermediate states in our construction which is briefly described in the next lemma.

Lemma 2.9 Accepting finite languages
Let $L \subseteq \Sigma^*$ be a finite language over Σ. Then there is a nondeterministic finite automaton $\mathcal{B} \in NFA(\Sigma)$ with:
1. $L(\mathcal{B}) = L$, i.e. $\mathcal{B}$ accepts L
2. $|\mathcal{B}| \leq \|L\| + 1$, i.e. the size of $\mathcal{B}$ exceeds the size of L only by one.
3. $\mathcal{B}$ can be constructed in polynomial time in $\|L\|$.

Proof.
For every $w \in L$, generate an appropriate automaton to accept $\{w\}$ only. Such automata can be constructed easily by merely concatenating states in a linear fashion, labelling the edges with the appropriate characters of the words to accept. The union of these automata is then constructed by combining the initial states of all the automata constructed so far to one inital state.
It is not difficult to see that the resulting automaton has the desired properties. □

2.2.2 Deterministic finite automata

In the deterministic case, the transition function of a finite automaton returns exactly one state for every input. Therefore, we might simply define deterministic finite automata by demanding that $|\delta(q, s)| = 1$ for every state q and every character s. However, this limitation can be utilized to simplify the definition of the accepted language. The next definition therefore introduces a slightly different transition function.

Definition 2.10 Deterministic finite automata
Let Σ be a finite alphabet. A *deterministic finite automaton* $\mathcal{B} := \langle Q, \Sigma, \delta, q_0, F \rangle$ *over* Σ is defined analogous to a nondeterministic finite automaton except for the the transition function δ. Here, $\delta(q, s)$ represents exactly one state in Q and not a subset of it. Thus, the deterministic transition function δ is of the form $\delta \colon Q \times \Sigma \to Q$. This entails a simplified definition of the extended transition function $\hat{\delta}$. For every $q \in Q$, $w \in \Sigma^*$, and for every $s \in \Sigma$, define $\hat{\delta}$ by:

$$\begin{aligned} \hat{\delta} \colon Q \times \Sigma^* &\to Q \\ q\,,\, \varepsilon &\mapsto q \\ q\,,\, ws &\mapsto \delta(\hat{\delta}(q, w), s) \end{aligned}$$

The definition of the accepted language is defined analogous to the nondeterministic case. The same holds for the size of $\mathcal{B}$. □

CHAPTER 3

Matching in Description Logics

In this chapter, we introduce the framework necessary to formally express the subject of this work. We need to define the description logics of interest and the classes of problems to be examined within these logics. For this, the following sections start by merely repeating the basic definitions given in [BKBM99].

3.1 Description logics

Throughout this work, we will refer to the following sets of atomic concepts, roles and variables, which are necessary for the definition of description logics. Let $\mathcal{C}$, $\mathcal{R}$, and $\mathcal{X}$ be mutually disjoint finite sets. Denote by $\mathcal{C}$ an arbitrary but fixed set of *atomic concepts* and denote by $\mathcal{R}$ an arbitrary but fixed set of *atomic roles*. Every formal language $L \subseteq \mathcal{R}^*$ is referred to as *role language*.

The description logics $\mathcal{ALN}$ and three of its sublanguages, $\mathcal{FL}_0$, $\mathcal{FL}_\bot$, and $\mathcal{FL}_\neg$, are now defined by specifying the syntax of its concept descriptions first and defining a model theoretic semantics afterwards.

Definition 3.1 Syntax of concept descriptions
A $(\geq)$-*number restriction* is of the form $(\geq nR)$, where $n \in \mathbb{N}$ and $R \in \mathcal{R}$. Similarly, a $(\leq)$-*number restriction* is of the form $(\leq nR)$. Denote by $\mathcal{N}_\geq$ an arbitrary but fixed finite set of $(\geq)$-number restrictions, denote by $\mathcal{N}_\leq$ an arbitrary but fixed finite set of $(\leq)$-number restrictions. The set $dom(\mathcal{ALN})$ of $\mathcal{ALN}$-*concept descriptions* over $\mathcal{C}$, $\mathcal{R}$, $\mathcal{N}_\geq$, and $\mathcal{N}_\leq$ is inductively defined by the following rules.

1. Every atomic concept $A \in \mathcal{C}$ and the symbol $\top$ ("top-concept") are concept descriptions.
2. If C and D are concept descriptions, then $C \sqcap D$ is as well.
3. If C is a concept description and $R \in \mathcal{R}$ is an atomic role, then $\forall R.C$ is a concept description.
4. The symbol $\bot$ ("bottom-concept") is a concept description.
5. For every atomic concept $A \in \mathcal{C}$, $\neg A$ is a concept description.
6. Every number restriction in $\mathcal{N}_\geq$ or $\mathcal{N}_\leq$ is a concept description.

The sets of $\mathcal{FL}_0$-, $\mathcal{FL}_\bot$- and $\mathcal{FL}_\neg$-concept descriptions are defined as subsets of $dom(\mathcal{ALN})$. For the set $dom(\mathcal{FL}_0)$ of $\mathcal{FL}_0$-concept descriptions only rules (1)–(3) are admitted, for $dom(\mathcal{FL}_\bot)$ only rules (1)–(4), and for $dom(\mathcal{FL}_\neg)$ only rules (1)–(5). □

The model-theoretic semantics of $\mathcal{ALN}$ and its sublanguages is defined by specifying a domain and an interpretation function mapping every concept description onto a subset of this domain.

Definition 3.2 Semantics
Let Δ^I be a non-empty set. Define an *interpretation* I by its *domain* Δ^I and its *interpretation function* $\cdot^I\colon dom(\mathcal{ALN}) \to \mathfrak{P}(\Delta^I) \cup \mathfrak{P}(\Delta^I \times \Delta^I)$ in such a way that $A^I \subseteq \Delta^I$ for all $A \in \mathcal{C}$ and $R^I \subseteq \Delta^I \times \Delta^I$ for all $R \in \mathcal{R}$. The interpretation function is then extended to complex concept descriptions by the following rules.

- $\bot^I := \emptyset$, $\top^I := \Delta^I$ (bottom,top)
- $(\neg A)^I := \Delta^I \setminus A^I$ (atomic negation)
- $(C \sqcap D)^I := C^I \cap D^I$ (conjunction)
- $(\forall R.C)^I := \{d \in \Delta^I \mid \forall e \in \Delta^I\colon (d,e) \in R^I \Rightarrow e \in C^I\}$ (role restriction)
- $(\leq nR)^I := \{d \in \Delta^I \mid |\{e \in \Delta^I \mid (d,e) \in R^I\}| \leq n\}$ ($\leq$-number restricion)
- $(\geq nR)^I := \{d \in \Delta^I \mid |\{e \in \Delta^I \mid (d,e) \in R^I\}| \geq n\}$ ($\geq$-number restriction)

where $A \in \mathcal{C}$, $R \in \mathcal{R}$, $C, D \in dom(\mathcal{ALN})$, $(\geq nR) \in \mathcal{N}_\geq$, and $(\leq nR) \in \mathcal{N}_\leq$. □

In [BN98a], Baader and Narendran have introduced the concept centered normal form, which can be used to represent concept descriptions in a standardized manner. It has been refined further in [BKBM99], yielding the $\mathcal{FL}_0$-normal form, which is applicable to $\mathcal{ALN}$-concept descriptions as well as to any of its three sublaguages considered here. The next definition introduces $\mathcal{FL}_0$-normal forms along with variable names used to denote the occurring role languages.

Definition 3.3 $\mathcal{FL}_0$-normal form
Denote by L an arbitrary identifier. For every $H \in \{\bot\} \cup \mathcal{C} \cup \{\neg A \mid A \in \mathcal{C}\} \cup \mathcal{N}_\leq \cup \mathcal{N}_\geq$, let the decoration L_H of L denote a finite role language. Define the L-*labelled* $\mathcal{FL}_0$-*normal form* of an $\mathcal{ALN}$-concept description C as follows:

$$C := \forall \mathsf{L}_\bot.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall \mathsf{L}_A.A \sqcap \bigsqcap_{A \in \mathcal{C}} \forall \mathsf{L}_{\neg A}.\neg A \sqcap \bigsqcap_{(\geq nR) \in \mathcal{N}_\geq} \forall \mathsf{L}_{(\geq nR)}.(\geq nR) \sqcap \bigsqcap_{(\leq nR) \in \mathcal{N}_\leq} \forall \mathsf{L}_{(\leq nR)}.(\leq nR)$$

Define $\forall\{\varepsilon\}.D := D$ and $\forall\emptyset.D := \top$ for every $\mathcal{ALN}$-concept description D. The L-labelled $\mathcal{FL}_0$-normal form of an arbitrary $\mathcal{FL}_\neg$-concept description can now be defined by requiring the role languages $\mathsf{L}_{(\geq mR)}$ and $\mathsf{L}_{(\leq nR)}$ to be empty for every number restriction $(\geq mR) \in \mathcal{N}_\geq$ and $(\leq nR) \in \mathcal{N}_\leq$. For the L-labelled $\mathcal{FL}_0$-normal form of any $\mathcal{FL}_\bot$-concept description, we additionally demand that $\mathsf{L}_{\neg A}$ is empty for every atomic concept $A \in \mathcal{C}$. Finally, for the L-labelled $\mathcal{FL}_0$-normal form of an $\mathcal{FL}_0$-concept descriptions, the language $\mathsf{L}_\bot$ is empty as well. □

The above notation has the advantage that every atomic concept, every negated atomic concept, and every number restriction from the specified sets occurs exactly once. It should be noted that the absence of an atomic concept in a concept description easily can be expressed by choosing $\emptyset$ for the respective role language. In this case, the respective

expression becomes equivalent to the top-concept, the occurrence of which does not alter the interpretation of any concept description. We can now define subsumption and equivalence of concept descriptions.

Definition 3.4 Inference problems
Subsumption ($\sqsubseteq$), *equivalence* ($\equiv$), and *strict subsumption* ($\sqsubset$) are defined as binary relations $(\sqsubseteq), (\equiv), (\sqsubset) \subseteq dom(\mathcal{ALN})^2$. For any $C, D \in dom(\mathcal{ALN})$ define:

- $C \sqsubseteq D$ (C "is subsumed by" D) iff $C^I \subseteq D^I$ for all interpretations I;
- $C \equiv D$ (C "is equivalent to" D) iff $C^I = D^I$ for all interpretations I;
- $C \sqsubset D$ (C "is strictly subsumed by" D) iff $C \sqsubset D$ and $C \not\equiv D$ for all interpretations I.

Since $dom(\mathcal{FL}_0) \subset dom(\mathcal{FL}_\bot) \subset dom(\mathcal{FL}_\neg) \subset dom(\mathcal{ALN})$, the notion of (strict) subsumption and equivalence is implicitly defined for the sublanguages of $\mathcal{ALN}$. □

The $\mathcal{FL}_0$-normal form as introduced in Definition 3.3 can be used to characterize subsumption and equivalence of concept descriptions. We first introduce the notion of excluding words, which is required for the characterization.

Definition 3.5 Excluding words
Let C be an $\mathcal{ALN}$-concept description. Let D be an $\mathcal{FL}_\neg$-concept description in U-labelled $\mathcal{FL}_0$-normal form. The set of *C-excluding words* is defined by:

$$E_C := \{w \in \Sigma^* | \, C \sqsubseteq \forall w.\bot\}$$

For D, define the role language $\widehat{U}_\bot$ as follows:

$$\widehat{U}_\bot := U_\bot \cup \bigcup_{A \in \mathcal{C}} (U_A \cap U_{\neg A})$$ □

It can be shown that $E_D = \widehat{U}_\bot \cdot \Sigma^*$ for every $\mathcal{FL}_\neg$-concept description D in U-labelled $\mathcal{FL}_0$-normal form. Thus, for $\mathcal{FL}_\neg$-concept descriptions in $\mathcal{FL}_0$-normal form the notion of excluding words can be characterized by $\widehat{U}_\bot$. We shall see later on, that a characterization of excluding words for $\mathcal{ALN}$-concept descriptions in $\mathcal{FL}_0$-normal form is more complex. Subsumption in $\mathcal{ALN}$ was characterized by Küsters in [Küs98], yielding the following result:

Lemma 3.6 Characterization of subsumption in $\mathcal{ALN}$
Let C, D be $\mathcal{ALN}$-concept descriptions. Let C be in U-labelled $\mathcal{FL}_0$-normal. Let D be in V-labelled $\mathcal{FL}_0$-normal form. Then $C \sqsubseteq D$ iff all of the following conditions hold.

1. $E_C \supseteq E_D$
2. $U_A \cup E_C \supseteq V_A \cup E_D$ for all $A \in \mathcal{C}$
3. $U_{\neg A} \cup E_C \supseteq V_{\neg A} \cup E_D$ for all $A \in \mathcal{C}$
4. $\bigcup_{m \geq n} U_{(\geq mR)} \cup E_C \supseteq \bigcup_{m \geq n} V_{(\geq mR)} \cup E_D$ for all $(\leq nR) \in \mathcal{N}_\leq$ with $n \geq 1$
5. $\bigcup_{m \leq n} U_{(\leq mR)} \cup E_C \cdot R^{-1} \supseteq \bigcup_{m \leq n} V_{(\leq mR)} \cup E_D \cdot R^{-1}$ for all $(\geq nR) \in \mathcal{N}_\geq$

Similar characterizations can be obtained for the sublanguages of $\mathcal{ALN}$. The following results for $\mathcal{FL}_0$, $\mathcal{FL}_\bot$ and $\mathcal{FL}_\neg$ can be obtained from [BKBM99].

Lemma 3.7 Characterization of subsumption in $\mathcal{FL}_0$
Let C and D be $\mathcal{FL}_0$-concept descriptions. Let C be in U-labelled $\mathcal{FL}_0$-normal form and let D be in V-labelled $\mathcal{FL}_0$-normal form. Then $C \sqsubseteq D$ iff $U_A \supseteq V_A$ for all $A \in \mathcal{C}$.

Lemma 3.8 Characterization of subsumption in $\mathcal{FL}_\bot$
Let C and D be $\mathcal{FL}_\bot$-concept descriptions. Let C be in U-labelled $\mathcal{FL}_0$-normal form and let D be in V-labelled $\mathcal{FL}_0$-normal form. Then $C \sqsubseteq D$ iff the following two conditions hold:

1. $U_\bot \cdot \Sigma^* \supseteq V_\bot \cdot \Sigma^*$
2. $U_A \cup U_\bot \cdot \Sigma^* \supseteq V_A \cup V_\bot \cdot \Sigma^*$ for all $A \in \mathcal{C}$

Lemma 3.9 Characterization of subsumption in $\mathcal{FL}_\neg$
Let C and D be $\mathcal{FL}_\neg$-concept descriptions. Let C be in U-labelled $\mathcal{FL}_0$-normal form and let D be in V-labelled $\mathcal{FL}_0$-normal form. Then $C \sqsubseteq D$ iff the following two conditions hold:

1. $\widehat{U}_\bot \cdot \Sigma^* \supseteq \widehat{V}_\bot \cdot \Sigma^*$
2. $U_A \cup \widehat{U}_\bot \cdot \Sigma^* \supseteq V_A \cup \widehat{V}_\bot \cdot \Sigma^*$ for all $H \in \mathcal{C} \cup \{\neg A | A \in C\}$

It has to be noted that [BKBM99] actually characterize equivalence and not subsumption. However, it can be shown with little effort that the above results are correct. Characterizations of equivalence can be derived easily from the above results. According to Definition 3.2, equivalence of concept descriptions is equivalent to mutual subsumption. In order to characterize equivalence it is therefore sufficient to replace all ($\supseteq$)-relations by ($=$) in the above four lemmae. The notion of subsumption is illustrated by the following example.

Example 3.10 Subsumption in $\mathcal{FL}_\neg$
Assume $\Sigma := \{R, S\}$ as the alphabet of roles. Consider the following $\mathcal{ALN}$-concept descriptions:

$$C := \forall\{R,S\}.B \sqcup \forall\{RR\}.(\geq 2S)$$
$$D := \forall\{R,S\}.\bot \sqcup \forall\{RR\}.(\geq 3S)$$
$$E := \forall\{\varepsilon\}.\bot \sqcup \forall\{R,S\}.A$$

Then E is subsumed by all the other concept descriptions, because it is equivalent to the bottom-concept. D is strictly subsumed by C, because it forbids R- and S-role successors instead of requiring B for them and imposes stronger number restrictions on RR-role successors. □

3.2 Matching problems

In order to define matching problems, we first need to introduce the notion of concept patterns. Intuitively, concept patterns extend concept descriptions by admitting variables. For this purpose, denote by $\mathcal{X}$ an arbitrary but fixed set of *variables*. For the sake of consistent notation throughout this work, let $\mathcal{X} =: \{X_1, \ldots, X_\ell\}$ for some $\ell \in \mathbb{N}$.

Definition 3.11 Concept patterns
Let $\mathcal{L} \in \{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. The set $dom_\mathcal{X}(\mathcal{L})$ of *$\mathcal{L}$-concept patterns* ist inductively defined as follows:

1. Every concept description $C \in dom(\mathcal{L})$ is a concept pattern.
2. Every concept variable $X \in \mathcal{X}$ is a concept pattern.
3. If C and D are concept patterns, then $C \sqcap D$ is as well.

4. If C is a concept pattern and $R \in \mathcal{R}$, then $\forall R.C$ is a concept pattern. □

The $\mathcal{FL}_0$-normal form for concept descriptions can be extended to concept patterns by treating variables as special atomic concepts.

Definition 3.12 $\mathcal{FL}_0$-normal form for concept patterns
Denote by L, L' arbitrary but distinct identifiers. For every $j \in \{1, \ldots, \ell\}$ and for every $H \in \{\bot\} \cup \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\} \cup \mathcal{N}_{\leq} \cup \mathcal{N}_{\geq}$, let decorations L_H and L'_j of L and L' denote a finite role language. Define the $(\mathsf{L}, \mathsf{L}')$-*labelled* $\mathcal{FL}_0$-*normal form* of an $\mathcal{ALN}$-concept pattern D as follows:

$$\begin{aligned} D := \; & \forall \mathsf{L}_{\bot}.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall \mathsf{L}_A.A \sqcap \bigsqcap_{A \in \mathcal{C}} \forall \mathsf{L}_{\neg A}.\neg A \\ & \sqcap \bigsqcap_{(\geq nR) \in \mathcal{N}_{\geq}} \forall \mathsf{L}_{(\geq nR)}.(\geq nR) \sqcap \bigsqcap_{(\leq nR) \in \mathcal{N}_{\leq}} \forall \mathsf{L}_{(\leq nR)}.(\leq nR) \\ & \sqcap \bigsqcap_{j=1}^{\ell} \mathsf{L}'_j.X_j \end{aligned}$$ □

For the assignment of concept descriptions to concept variables, we introduce substitutions over $\mathcal{ALN}$ and its sublanguages.

Definition 3.13 Substitution
Let $\mathcal{L} \in \{\mathcal{FL}_0, \mathcal{FL}_{\bot}, \mathcal{FL}_{\neg}\}$. Define a *substitution* σ *over* $\mathcal{L}$ as a mapping from the set of variables $\mathcal{X}$ to $dom(\mathcal{L})$. σ is extended to a function $\hat{\sigma}$ of the form $\hat{\sigma}\colon\ dom_{\mathcal{X}}(\mathcal{L}) \to dom(\mathcal{L})$, such that the following conditions hold for all $C, D \in dom_{\mathcal{X}}(\mathcal{L})$, for all $X \in \mathcal{X}$, $A \in \mathcal{C}$, $R \in \mathcal{R}$, and for all number restrictions $(\geq nR)$ and $(\leq nR)$.

- $\hat{\sigma}(X) = \sigma(X)$
- $\hat{\sigma}(A) = A,\ \hat{\sigma}(\neg A) = \neg A$
- $\hat{\sigma}(\bot) = \bot,\ \hat{\sigma}(\top) = \top$
- $\hat{\sigma}(C \sqcap D) = \hat{\sigma}(C) \sqcap \hat{\sigma}(D)$
- $\hat{\sigma}(\forall R.C) = \forall R.\hat{\sigma}(C)$
- $\hat{\sigma}(\leq nR) = (\leq n\, R),\ \hat{\sigma}(\geq n\, R) := (\geq n\, R)$

For a simpler notation, we will not distinguish between a substitution σ and its extension $\hat{\sigma}$ in the remainder of this work and denote both by σ. For substitutions σ and σ' sharing the same domain $\mathcal{X}$, we define the following relations.

- $\sigma \sqsubseteq \sigma'$ iff $\sigma(X) \sqsubseteq \sigma'(X)$ for all concept variables $X \in \mathcal{X}$.
- $\sigma \sqsubset \sigma'$ iff $\sigma \sqsubseteq \sigma'$ and there is a variable $X \in \mathcal{X}$ with $\sigma(X) \sqsubset \sigma'(X)$.
- $\sigma \equiv \sigma'$ iff $\sigma \sqsubseteq \sigma'$ and $\sigma' \sqsubseteq \sigma$. □

The purpose of these relations is to provide a means to determine whether the values assigned by one substitution are more general (in respect to subsumption) than another. We are now ready to define matching problems.

Definition 3.14 Matching problems
Let $\mathcal{L} \in \{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let C denote an $\mathcal{L}$-concept description. Let D denote an $\mathcal{L}$-concept pattern. Referring to Definition 3.4, we distinguish two different kinds of matching problems.

- An *$\mathcal{L}$-matching problem modulo subsumption* is of the form $C \sqsubseteq^? D$. A *solution* to this problem is a substitution σ over $\mathcal{L}$ with $C \sqsubseteq \sigma(D)$.
- An *$\mathcal{L}$-matching problem modulo equivalence* is of the form $C \sqsubseteq^? D$. A *solution* to this problem is a substitution σ over L with $C \equiv \sigma(D)$.
- A *system of $\mathcal{L}$-matching problems* is of the form $\{P_i | 1 \leq i \leq n\}$, where n is a positive integer and for every i, P_i is an $\mathcal{L}$-matching problem modulo equivalence or modulo subsumption. A *solution* to this system is a substitution which solves P_i for every i.

The notion of $\mathcal{FL}_0$-normal forms is extended to $\mathcal{L}$-matching problems as follows. An $\mathcal{L}$-matching problem is in $(\mathsf{L}, \mathsf{L}', \mathsf{L}'')$-*labelled $\mathcal{FL}_0$-normal form* if and only if C is in L-labelled $\mathcal{FL}_0$-normal form and D is in $(\mathsf{L}', \mathsf{L}'')$-labelled $\mathcal{FL}_0$-normal form for distinct identifiers L, L', and L''. □

It is shown in [BKBM99] that matching problems modulo subsumption and systems of matching problems can be reduced to matching modulo equivalence. The following lemma merely summarizes the respective results and may therefore be stated without proof.

Lemma 3.15 Representation of matching problems
Let $\mathcal{L} \in \{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Then
1. For every $\mathcal{L}$-matching problem modulo subsumption there is a polynomially large $\mathcal{L}$-matching problem modulo equivalence with the same set of solutions.
2. For every system of $\mathcal{L}$-matching problems there is a polynomially large $\mathcal{L}$-matching problem modulo equivalence with the same set of solutions.
3. Both reductions can be computed in polynomial time.

The idea of combining a system of matching problems into a single matching problem modulo equivalence is illustrated by the next example, where a simple system of matching problems is considered.

Example 3.16 Representation of matching problems
Let C, E be $\mathcal{L}$-concept descriptions and let D, F be $\mathcal{L}$-concept patterns. Let $R_1, R_2 \in \Sigma$ be distinct atomic roles. Let $P := \{C \equiv^? D, E \sqsubseteq^? F\}$ be a system of $\mathcal{L}$-matching problems. Then P has the same set of solutions as the following matching problem modulo equivalence: $\forall R_1.C \sqcap \forall R_2.E \equiv^? \forall R_1.C \sqcap \forall R_2.(E \sqcap F)$.

Due to the results of the above lemma it is sufficient to examine single matching problems modulo equivalence. A matching problem can be specified further by stating additional requirements for the solution. This leads to a definition central for this work.

Definition 3.17 Matching problems with side conditions
Let $\mathcal{L} \in \{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let C denote an $\mathcal{L}$-concept description. Let D denote an $\mathcal{L}$-concept pattern. For every $j \in \{1, \ldots, \ell\}$ let E_j denote an $\mathcal{ALN}$-concept pattern. For the definition of matching problems with side conditions, we first need to introduce the notion of subsumption conditions. Again, we define a strict and a non-strict version.

- An *$\mathcal{L}$-side condition* for $X \in \mathcal{X}$ is of the form $X \sqsubseteq^? D$. A solution to this condition is a substitution σ over $\mathcal{L}$ with $\sigma(X) \sqsubseteq \sigma(D)$.

- A *strict $\mathcal{L}$-side condition* for $X \in \mathcal{X}$ is of the form $X \sqsubset^? D$. A solution to this condition is a substitution σ over $\mathcal{L}$ with $\sigma(X) \sqsubset \sigma(D)$.

Matching problems with side conditions can now be defined as a tuple consisting of a matching problem and a set of side conditions.

- An *$\mathcal{L}$-matching problem modulo equivalence with (non-strict) side conditions* is of the form $(C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$. A solution to this problem is a substitution σ over $\mathcal{L}$ with $C \equiv \sigma(D)$ and $\sigma(X_j) \sqsubseteq \sigma(E_j)$ for every $j \in \{1, \ldots, \ell\}$.
- An *$\mathcal{L}$-matching problem modulo equivalence with strict side conditions* is of the form $(C \equiv^? D, \{X_j \, \rho_j^? \, E_j | 1 \leq j \leq \ell\})$, where $\rho_j \in \{\sqsubset, \sqsubseteq\}$ for all $j \in \{1, \ldots, \ell\}$. A solution to this problem is a substitution σ over $\mathcal{L}$ with $C \equiv \sigma(D)$ and $\sigma(X_j) \, \rho_j \, \sigma(E_j)$ for every $j \in \{1, \ldots, \ell\}$.
- Side conditions are called *acyclic* iff the variables $X_j, \ldots, X_\ell$ do not occur in E_j for every $j \in \{1, \ldots, \ell\}$.

An $\mathcal{L}$-matching problem with side conditions is in $(\mathsf{L}, \mathsf{L}', \mathsf{L}'')$-*labelled $\mathcal{FL}_0$-normal form* iff for unique identifiers L, L', and L'', C is in L-labelled $\mathcal{FL}_0$-normal form, D is in $(\mathsf{L}', \mathsf{L}'')$-labelled $\mathcal{FL}_0$-normal form, and E_j is in $(\mathsf{L}'_j, \mathsf{L}''_j)$-labelled $\mathcal{FL}_0$-normal form for every index $j \in \{1, \ldots, \ell\}$. □

CHAPTER 4

Solving Matching Problems

This chapter presents methods to solve matching problems modulo equivalence without side conditions in $\mathcal{ALN}$ and its sublanguages. In the first section we give a summary of results on this subject, which has been studied extensively in [BKBM99]. The solution strategies proposed there rely on the informally introduced notion of "treelike automata". In the second section, we formally define treelike automata and discuss their complexity. This allows a formal verification of the complexity results stated in [BKBM99], which will be given in the third section. We will see that matching problems modulo equivalence in fact can be solved in polynomial time.

4.1 Results from previous work

In [BKBM99] and [Küs98], matching modulo equivalence in $\mathcal{FL}_\bot$, $\mathcal{FL}_\neg$ and $\mathcal{ALN}$ is reduced to solving equations over formal languages, which we will refer to as "solvability equations". The following four definitions and the following three lemmata summarize the results of the respective articles.

Definition 4.1 Solvability equations in $\mathcal{FL}_\bot$
Let $(C \equiv^? D)$ be an $\mathcal{FL}_\bot$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Define the following formal language equations:

$$U_\bot \cdot \Sigma^* = V_\bot \cdot \Sigma^* \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,\bot} \cdot \Sigma^* \qquad (\bot)$$

$$U_A \cup U_\bot \cdot \Sigma^* = V_A \cup U_\bot \cdot \Sigma^* \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,A} \qquad (A)$$

for all $A \in \mathcal{C}$. □

Solvability of the above system of equations is decided by assigning appropriate formal languages to the occurring variables. The following lemma specifies these formal languages.

Lemma 4.2 Testing solvability in $\mathcal{FL}_\bot$
Let $(C \equiv^? D)$ be an $\mathcal{FL}_\bot$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Then the system of equations $(\bot), ((A) | A \in \mathcal{C})$ has a solution iff:

1. For every $j \in \{1, \ldots, \ell\}$, replacing the expression $X_{j,\bot} \cdot \Sigma^*$ by the set $\widehat{L}_{j,\bot} := \bigcap_{w \in W_j} w^{-1} \cdot (U_\bot \cdot \Sigma^*)$ solves equation $(\bot)$.

2. For every $A \in \mathcal{C}$ and for every $j \in \{1, \ldots, \ell\}$, replacing the variable $X_{j,A}$ by the set $\widehat{L}_{j,A} := \bigcap_{w \in W_j} w^{-1} \cdot (U_A \cup U_\bot \cdot \Sigma^*)$ solves equation (A).

Similar results are obtained for $\mathcal{FL}_\neg$. Here, we have the following solvability equations.

Definition 4.3 Solvability equations in $\mathcal{FL}_\neg$
Let $(C \equiv^? D)$ be an $\mathcal{FL}_\neg$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Define the following formal language equations.

$$\widehat{U}_\bot \cdot \Sigma^* = V_\bot \cdot \Sigma^* \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,\bot} \cdot \Sigma^* \cup \bigcup_{A \in \mathcal{C}} \mathit{Int}(A, \neg A) \cdot \Sigma^* \qquad (\bot)$$

$$U_A \cup \widehat{U}_\bot \cdot \Sigma^* = V_A \cup \widehat{U}_\bot \cdot \Sigma^* \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,A} \qquad (A)$$

$$U_{\neg A} \cup \widehat{U}_\bot \cdot \Sigma^* = V_{\neg A} \cup \widehat{U}_\bot \cdot \Sigma^* \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,\neg A} \qquad (\neg A)$$

for all $A \in \mathcal{C}$, where

$$\mathit{Int}(A, \neg A) := (V_A \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,A}) \cap (V_{\neg A} \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,\neg A}). \qquad \square$$

Though still only dependent on the set $\mathcal{C}$ of atomic concepts, the number of equations has increased, because negated atomic concepts need to be dealt with separately. Observe that in the solvability equations for $\mathcal{FL}_\bot$, the equation $(\bot)$ was completely independent of role languages referring to atomic concepts $A \in \mathcal{C}$. For $\mathcal{FL}_\neg$ this is no longer the case, because the conjunction of an atomic concept and its negation is inconsistent. For that reason, the expression *Int* is included in equation $(\bot)$. The following lemma provides a test for solvability in $\mathcal{FL}_\neg$.

Lemma 4.4 Testing solvability in $\mathcal{FL}_\neg$
Let $(C \equiv^? D)$ be an $\mathcal{FL}_\neg$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Then the system of equations $(\bot), ((A) | A \in \mathcal{C}), ((\neg A) | A \in \mathcal{C})$ has a solution iff:

1. For every $A \in \mathcal{C}$ and for every $j \in \{1, \ldots, \ell\}$, replacing the variable $X_{j,A}$ by the set $\widehat{L}_{j,A} := \bigcap_{w \in W_j} w^{-1} \cdot (U_A \cup \widehat{U}_\bot \cdot \Sigma^*)$ solves equation (A).

2. For every $A \in \mathcal{C}$ and for every $j \in \{1, \ldots, \ell\}$, replacing the variable $X_{j,\neg A}$ by the set $\widehat{L}_{j,\neg A} := \bigcap_{w \in W_j} w^{-1} \cdot (U_{\neg A} \cup \widehat{U}_\bot \cdot \Sigma^*)$ solves equation (A).

3. For every $j \in \{1, \ldots, \ell\}$, replacing the variables $X_{j,\bot} \cdot \Sigma^*$ by the expression $\widehat{L}_{j,\bot} := \bigcap_{w \in W_j} w^{-1} \cdot (U_\bot \cdot \Sigma^*)$ together with the assignments proposed in (1) and (2) solves equation $(\bot)$.

Note that condition three requires "together with the assignments proposed in (1) and (2)". This is necessary because of the expression *Int*, by which equation $(\bot)$ becomes dependent on the other assignments. For $\mathcal{ALN}$, we have to introduce some notation first.

Definition 4.5 Notation
Let $(C \equiv^? D)$ be an $\mathcal{ALN}$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. The following tuples of variables are defined for the sake of readability.

$$\begin{aligned}
X_\bot &:= (X_{j,\bot} | 1 \leq j \leq \ell) \\
X_\mathcal{C} &:= (X_{j,A} | 1 \leq j \leq l, A \in \mathcal{C}) \\
X_\neg &:= (X_{j,\neg A} | 1 \leq j \leq \ell, A \in \mathcal{C}) \\
X_\geq &:= (X_{j,(\geq nR)} | 1 \leq j \leq \ell, (\geq nR) \in \mathcal{N}_\geq) \\
X_\leq &:= (X_{j,(\leq nR)} | 1 \leq j \leq \ell, (\leq nR) \in \mathcal{N}_\leq)
\end{aligned}$$

Denote by α an arbitrary assignment of finite languages to the variables contained in the tuples, i.e. $\alpha(X_{i,H}) = L_{i,H}$ for all $i \in \{1, \ldots, \ell\}$ and $H \in \{\bot\} \cup \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\} \cup \mathcal{N}_\leq \cup \mathcal{N}_\geq$. Let σ be the substitution corresponding to α, so that for every $j \in \{1, \ldots, \ell\}$ we have:

$$\begin{aligned}
\sigma(X_j) := \; & \forall \alpha(X_{j,\bot}).\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall \alpha(X_{j,A}).A \sqcap \bigsqcap_{A \in \mathcal{C}} \forall \alpha(X_{j,\neg A}).\neg A \\
& \sqcap \bigsqcap_{(\geq nR) \in \mathcal{N}_\geq} \forall \alpha(X_{j,(\geq nR)}).(\geq nR) \sqcap \bigsqcap_{(\leq nR) \in \mathcal{N}_\leq} \forall \alpha(X_{j,(\leq nR)}).(\leq nR)
\end{aligned}$$

Denote by $E_D(X_\bot, X_\mathcal{C}, X_\neg, X_\geq, X_\leq)$ the set of excluding words obtained for D relative to the assignment α. Thus, let

$$E_D(\alpha(X_\bot), \alpha(X_\mathcal{C}), \alpha(X_\neg), \alpha(X_\geq), \alpha(X_\leq)) := E_{\sigma(D)},$$

yielding the set of $\sigma(D)$-excluding words after assigning the occurring variables. □

The above construct is necessary, because the set of excluding words is defined only for concept descriptions and not for concept patterns. Consequently, we must assume some assignment of the concept variables occurring on the right-hand side of the matching problem. With these preparations, the following solvability equations are provided.

Definition 4.6 Solvability equations in $\mathcal{ALN}$
Let $(C \equiv^? D)$ be an $\mathcal{ALN}$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. With the notation of the above definition, define the following formal language equations.

$$E_C = E_D(X_\bot, X_\mathcal{C}, X_\neg, X_\geq, X_\leq) \tag{$\bot$}$$

$$U_A \cup E_C = V_A \cup E_C \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,A} \tag{A}$$

$$U_{\neg A} \cup E_C = V_{\neg A} \cup E_C \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,\neg A} \tag{$\neg A$}$$

$$\bigcup_{m \geq n} U_{(\geq mR)} \cup E_C = V_{(\geq mR)} \cup E_C \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,(\geq nR)} \tag{$\geq nR$}$$

$$\bigcup_{m \leq n'} U_{(\leq mR)} \cup E_C \cdot R^{-1} = V_{(\leq mR)} \cup E_C \cdot R^{-1} \cup \bigcup_{j=1}^{\ell} W_j \cdot X_{j,(\geq n'R)} \tag{$\leq n'R$}$$

for all $\mathcal{A} \in \mathcal{C}$, $n \in \mathbb{N} \setminus \{0\}$, $n' \in \mathbb{N}$, $(\geq nR) \in \mathcal{N}_\geq$, and $(\leq n'R) \in \mathcal{N}_\leq$. □

Here, the number of equations additionally depends on the number restrictions occurring in the matching problem. Again, equation ($\bot$) takes into account role languages referring to other concepts than the $\bot$-concept. However, this property is syntactically hidden in the constructs E_C and E_D, which are defined as $\{w \in \Sigma^* | C \sqsubseteq \forall w.\bot\}$ and analogously for E_D, as we know from Chapter 2.

Lemma 4.7 Testing solvability in $\mathcal{ALN}$
Let $(C \equiv^? D)$ be an $\mathcal{ALN}$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Furthermore, let $\widehat{L}_{j,\bot} := \bigcap_{w \in W_j} w^{-1} \cdot E_C$. Then there exists a finite set $L_{j,\bot}$ of polynomial size in the input matching problem with $L_{j,\bot} \cdot \Sigma^* = \widehat{L}_{j,\bot}$. As mentioned previously, this is shown in [BKBM99]. The system of equations $(\bot), ((A)|A \in \mathcal{C}), ((\neg A)|A \in \mathcal{C}), ((\geq nR)|(\geq nR) \in \mathcal{N}_{\geq}), ((\leq nR)|(\leq nR) \in \mathcal{N}_{\leq})$ then has a solution iff:

1. For every $j \in \{1, \ldots, \ell\}$ and $A \in \mathcal{C}$, replacing the variable $X_{j,A}$ by the set $L_{j,A} := (\bigcap_{w \in W_j} w^{-1} \cdot (U_A \cup E_C)) \setminus \widehat{L}_{j,\bot}$ solves equation (A).
2. For every $j \in \{1, \ldots, \ell\}$ and $A \in \mathcal{C}$, replacing the variable $X_{j,\neg A}$ by the set $L_{j,\neg A} := (\bigcap_{w \in W_j} w^{-1} \cdot (U_{\neg A} \cup E_C)) \setminus \widehat{L}_{j,\bot}$ solves equation $(\neg A)$.
3. For every $j \in \{1, \ldots, \ell\}$ and $(\geq nR) \in \mathcal{N}_{\geq}$, replacing the variable $X_{j,\neg A}$ by the set $L_{j,(\geq nR)} := (\bigcap_{w \in W_j} w^{-1} \cdot (\bigcup_{m \geq n} U_{(\geq nR)} \cup E_C)) \setminus \widehat{L}_{j,\bot}$ solves equation $(\geq nR)$.
4. For every $j \in \{1, \ldots, \ell\}$ and $(\leq nR) \in \mathcal{N}_{\leq}$, replacing the variable $X_{j,\neg A}$ by the set $L_{j,(\leq nR)} := (\bigcap_{w \in W_j} w^{-1} \cdot (\bigcup_{m \leq n} U_{(\leq nR)} \cup E_C \cdot \mathcal{R}^{-1})) \setminus \widehat{L}_{j,\bot}$ solves equation $(\leq nR)$.
5. For every $j \in \{1, \ldots, \ell\}$, replacing the variable $X_{j,\bot}$ by the set $L_{j,\bot}$ together with the assignments proposed in (1)–(4) solves equation $(\bot)$.

Observe that in the above conditions a finite alternative to $\widehat{L}_{j,\bot}$ is provided and that $\widehat{L}_{j,\bot}$ is subtracted from the other languages, thus producing polynomially large languages as solutions to the equations. This is an immediate consequence of [BKBM99], where it was shown that the above solution languages can be computed in polynomial time.
By inserting the languages specified in the previous lemmata into the referring solvability equations, we obtain variable-free formal language equations, which are valid if and only if the original matching problem is solvable. Note that this reduces the decision problem from solvability to equality. Therefore, we only need a method to decide equality for the equations obtained in this way. In [BKBM99], the notion of treelike automata is introduced to facilitate this task. The next section formally defines them and discusses their properties.

4.2 Treelike automata

We need to decide equality for the variable-free variants of the solvability equations introduced in the last section. Why not employ ordinary finite automata for this decision? In the general case, repeatedly intersecting deterministic finite automata may produce exponentially large results [YZ91]. Nondeterministic finite automata, on the other hand, cannot be complemented efficiently. Since both intersection and complement are essential operations for our decision problem, we need to define automata suiting our requirements better—treelike automata. Their main objective is to support efficient operations for the left quotient, the complement, the intersection, and the union.

4.2.1 Basic definitions

In order to define treelike automata as a special class of deterministic finite automata it would be sufficient to restrict the set of states and the transition function in a certain way. However, we want to stress the analogy to trees and therefore define treelike automata inductively. Furthermore, this will enable us to define operations on treelike automata in a particularly simple fashion.

Definition 4.8 Syntax of treelike automata
Let Σ be a finite alphabet and $\mathit{Type} := \{\underline{\mathit{Nor}}, \underline{\mathit{Fin}}\}$ a set of labels. The set $\mathit{Treelike}(\Sigma)$ of *treelike automata over* Σ is inductively defined by:

$$\mathit{Treelike}(\Sigma) := \{\underline{\mathit{Nil}}, \underline{\mathit{Cyc}}\} \cup (\mathit{Type} \times (\mathit{Treelike}(\Sigma)^{|\Sigma|}))$$

The inductive definition of $\mathit{Treelike}(\Sigma)$ corresponds to the structure of a tree with nodes of the form $(\underline{\mathit{Nor}}, (\cdot))$ and $(\underline{\mathit{Fin}}, (\cdot))$ respectively. The second component of a node is a tuple of the dimension $|\Sigma|$, representing the list of successors of that node. Since we assume the alphabet Σ to be ordered, successors for certain elements of Σ are simply inserted in the appropriate places of the tuple. Thus, each node has one successor for every character of Σ. The idea is that having a successor for some character s_i is analogous to a directed edge labelled s_i in an ordinary finite automaton.
There are two special constructors, $\underline{\mathit{Nil}}$ and $\underline{\mathit{Cyc}}$. $\underline{\mathit{Nil}}$ is supposed to be the automaton accepting no input whatsoever, $\underline{\mathit{Cyc}}$ the one accepting any input. In a tree representation, $\underline{\mathit{Nil}}$ and $\underline{\mathit{Cyc}}$ appear as leaf-nodes. However, they deviate from this notion in one respect which will become clear when defining the transition function. Both nodes have themselves as successor for every character of Σ. In this respect treelike automata differ from the intuition of trees. □

In an ordinary tree, leaf-nodes like $\underline{\mathit{Nil}}$ and $\underline{\mathit{Cyc}}$ pointing to themselves do not exist. It might therefore appear negligent to refer to trees when illustrating the structure of treelike automata. We will nevertheless do so for two reasons. Firstly, the existence of unusual leaf-nodes does not violate the analogy to trees severely; secondly, the terminology existing for trees will prove suitable to explain our ideas.

Let us first introduce some abbreviations for treelike automata.

Definition 4.9 Notation

- For $(\underline{\mathit{Nor}}, (\mathcal{T}_1, \ldots, \mathcal{T}_n))$ write $(\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$;
  for $(\underline{\mathit{Fin}}, (\mathcal{T}_1, \ldots, \mathcal{T}_n))$ write $(\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$.
- Write $(\mathcal{T}_1, \ldots, \mathcal{T}_n)_*$, if $(\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ as well as $(\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$ are referred to, i.e. $(\cdot)_*$ means "$(\cdot)_N$ *or* $(\cdot)_F$" in existentially quantified statements and "$(\cdot)_N$ *and* $(\cdot)_F$" in for-all-quantified statements. □

For the rest of this chapter, we shall regard Σ as an arbitrary but fixed finite alphabet with n unique elements for some positive integer n. Therefore, let $\Sigma := \{s_1, \ldots, s_n\}$.
Like deterministic finite automata, treelike automata are equipped with a transition function and a set of accepting states. Unlike deterministic finite automata, both transition function and accepting states will be defined identical for every treelike automaton. This is possible, because the behaviour of the automaton is implicitly contained in its inductive structure.

Definition 4.10 Semantics of treelike automata
For any treelike automata $\mathcal{T}_1, \ldots, \mathcal{T}_n \in \mathit{Treelike}(\Sigma)$ and for any $s_i \in \Sigma$, the transition function δ is inductively defined by:

$$\begin{aligned} \delta\colon \mathit{Treelike}(\Sigma) \times \Sigma &\to \mathit{Treelike}(\Sigma) \\ \underline{\mathit{Nil}}\,,\, s_i &\mapsto \underline{\mathit{Nil}} \\ \underline{\mathit{Cyc}}\,,\, s_i &\mapsto \underline{\mathit{Cyc}} \\ (\mathcal{T}_1, \ldots, \mathcal{T}_n)_*\,,\, s_i &\mapsto \mathcal{T}_i \end{aligned}$$

Observe that $\underline{Nil}$ and $\underline{Cyc}$ point to themselves as successors, thus deviating from the idea of tree-nodes. δ is now generalized in the usual way to accept words of length greater than one and the empty word ε. For $\mathcal{T} \in \mathit{Treelike}(\Sigma)$, for $s_i \in \Sigma$, and for any $v \in \Sigma^*$, define inductively:

$$\begin{aligned} \hat{\delta}\colon \mathit{Treelike}(\Sigma) \times \Sigma^* &\to \mathit{Treelike}(\Sigma) \\ \mathcal{T}\,, \varepsilon &\mapsto \mathcal{T} \\ \mathcal{T}\,, s_i v &\mapsto \hat{\delta}(\delta(\mathcal{T}, s_i), v) \end{aligned}$$

One can see that $\hat{\delta}(\underline{Nil}, w)$ equals $\underline{Nil}$ for any word w. The same holds for $\underline{Cyc}$. The difference between $\underline{Nil}$ and $\underline{Cyc}$ becomes apparent when defining accepting conditions. The set Acc of accepting states is the same for any treelike automaton. It is defined as infinite set of the form:

$$Acc := \{\underline{Cyc}\} \cup (\{\underline{Fin}\} \times (\mathit{Treelike}(\Sigma)^{|\Sigma|}))$$

Observe that for a given node $\mathcal{T}$, membership in Acc can be tested in constant time. It is sufficient to test whether or not $\mathcal{T}$ equals $\underline{Cyc}$ or begins with $(\underline{Fin}, \ldots)$. The language accepted by a treelike automaton $\mathcal{T}$ now can be defined as:

$$L(\mathcal{T}) := \{w \in \Sigma^* |\ \hat{\delta}(\mathcal{T}, w) \in Acc\}$$

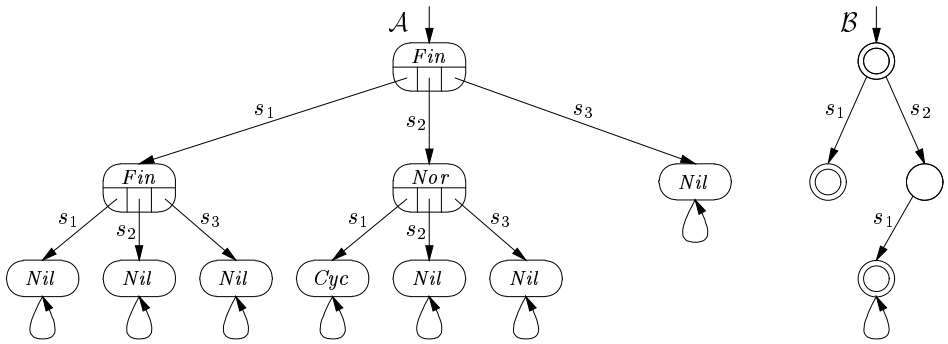
Equivalence is defined as usual for automata. Treelike automata $\mathcal{S}$ and $\mathcal{T}$ are equivalent ($\mathcal{S} \equiv \mathcal{T}$), iff they accept the same language, i.e. $L(\mathcal{S}) = L(\mathcal{T})$. □

The following example illustrates the structure of treelike automata in comparison to deterministic finite automata.

Example 4.11 Treelike automata
Let $\Sigma := \{s_1, s_2, s_3\}$ and let $\mathcal{A}$ be a treelike automaton over Σ such that:

$$\mathcal{A} := (\underline{Fin}, ((\underline{Fin}, (\underline{Nil}, \underline{Nil}, \underline{Nil})), (\underline{Nor}, (\underline{Cyc}, \underline{Nil}, \underline{Nil})), \underline{Nil}))$$

Then $\mathcal{A}$ can be represented by a transition tree as shown on the left-hand side below. In this tree every node represents a state of the automaton. The upper half denotes the label of the state, the lower half lists the successors for every character of Σ. The topmost state is the initial state. A directed edge labelled s_i corresponds to a transition of the automaton upon input s_i. Cyclic edges at the leave-nodes apply to every character of Σ.



For instance, upon input $s_2 s_1 s_3$ the automaton $\mathcal{A}$ reaches the state $\underline{Cyc}$. According to the definition, the initial state, its leftmost direct successor, and the leaf-node labelled $\underline{Cyc}$ are accepting states. We thus find that $\mathcal{A}$ accepts the language $L(\mathcal{A}) = \{\varepsilon, s_1\} \cup \{s_2 s_1\} \cdot \Sigma^*$.

Observe that the size of $\mathcal{A}$ depends on the cardinality of Σ. On the right-hand side, we have contrasted a deterministic finite automaton $\mathcal{B}$ accepting the same language. Obviously, $\mathcal{B}$ is significantly smaller that $\mathcal{A}$. When introducing operations on treelike automata, however, we shall see that this overhead yields advantages. □

Exploiting the analogy to trees, we define some further notions for treelike automata:

Definition 4.12 Subtrees and trimmed automata
$\mathcal{T}, \mathcal{S} \in \mathit{Treelike}(\Sigma)$

- $\mathcal{S}$ is a *subtree of* $\mathcal{T}$ iff there is a word $w \in \Sigma^*$ with $\hat{\delta}(\mathcal{T}, w) = \mathcal{S}$.
- $\mathcal{T}$ is *trimmed* iff $\mathcal{T}$ has no subtree of the form $(\underline{Nil}, \ldots, \underline{Nil})_N$, i.e. all leave-nodes in the tree represented by $\mathcal{T}$ are accepting states of the form $(\underline{Nil}, \ldots, \underline{Nil})_F$ or $\underline{Cyc}$. □

The notion of a subtree corresponds to the intuitive idea of a subtree in the tree represented by a treelike automaton.
Trimming treelike automata aims at ruling out certain irregularities. Nodes of the form $(\underline{Nil}, \ldots, \underline{Nil})_N$ do not contribute to the language accepted by the automaton. In certain contexts it will be necessary to modify automata in such a way that no leave-node accepts only the empty set.
The canonical definition of the accepted language in Definition 4.10 makes it easy to see the connection between treelike and deterministic finite automata. For practical purposes, however, we can take advantage of the rather simple structure of treelike automata to propose an alternative definition of the accepted language.

Definition 4.13 Language function
For treelike automata $\mathcal{T}_1, \ldots, \mathcal{T}_n \in \mathit{Treelike}(\Sigma)$ the function *lang* is inductively defined as follows.

$$\begin{aligned} \mathit{lang}\colon \mathit{Treelike}(\Sigma) &\to \mathfrak{P}(\Sigma^*) \\ \underline{Nil} &\mapsto \emptyset \\ \underline{Cyc} &\mapsto \Sigma^* \\ (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N &\mapsto \bigcup_{i=1}^{n} \{s_i\} \cdot \mathit{lang}(\mathcal{T}_i) \\ (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F &\mapsto \{\varepsilon\} \cup \mathit{lang}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N) \end{aligned}$$ □

The function *lang* is intended to simplify the handling of treelike automata. Nevertheless, we still have to prove that the above definition in fact is equivalent to the previous one. This will be shown in the next section.
Since we will be concerned with complexity issues, a means of quantifying the size of a treelike automaton is required. The next definition provides such a measure.

Definition 4.14 Size of treelike automata
For treelike automata $\mathcal{T}_1, \ldots, \mathcal{T}_n \in \mathit{Treelike}(\Sigma)$ the function $\|\cdot\|$ is inductively defined by:

$$\begin{aligned} \|\cdot\|\colon \mathit{Treelike}(\Sigma) &\to \mathbb{N} \\ \underline{Nil} &\mapsto 1 \\ \underline{Cyc} &\mapsto 1 \\ (\mathcal{T}_1, \ldots, \mathcal{T}_n)_* &\mapsto 1 + \sum_{i=1}^{n} \|\mathcal{T}_i\| \end{aligned}$$ □

$\|\mathcal{T}\|$ corresponds to the amount of space necessary to represent $\mathcal{T}$.

The definitions stated so far make treelike automata appear fairly similar to deterministic finite automata. It is the task of the next section to answer two questions: How much do treelike automata and *DFA* have in common and what class of formal languages can be represented by treelike automata.

4.2.2 Properties

At first, we have to verify the correctness of the alternative definition given in the previous section for the language accepted by treelike automata. In most cases, using the function *lang* will prove simpler than the original definition.

Lemma 4.15 Correctness of *lang*
Let $\mathcal{T} \in \mathit{Treelike}(\Sigma)$ be a treelike automaton. Then $L(\mathcal{T}) = \mathit{lang}(\mathcal{T})$, i.e. *lang* is an equivalent definition of the language accepted by $\mathcal{T}$.

Proof.
We prove the claim by induction over the structure of $\mathcal{T}$.
$\triangleright$ $\mathcal{T} = \underline{\mathit{Nil}}$ or $\mathcal{T} = \underline{\mathit{Cyc}}$: By Definition 4.10, $L(\underline{\mathit{Nil}}) = \{w \in \Sigma^* | \hat{\delta}(\underline{\mathit{Nil}}, w) \in \mathit{Acc}\}$. It holds that $\hat{\delta}(\underline{\mathit{Nil}}, w)$ equals $\underline{\mathit{Nil}}$ for any word w. As $\underline{\mathit{Nil}}$ is no element of *Acc*, we have $L(\underline{\mathit{Nil}}) = \emptyset$. By Definition 4.13, this equals $\mathit{lang}(\underline{\mathit{Nil}})$. The case $\mathcal{T} = \underline{\mathit{Cyc}}$ is similar. The only difference is that $\underline{\mathit{Cyc}} \in \mathit{Acc}$, so that we gain Σ^* as accepted language which matches the definition of $\mathit{lang}(\underline{\mathit{Cyc}})$.

$\triangleright$ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$: Then $L((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N) = \{w \in \Sigma^* | \hat{\delta}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N, w) \in \mathit{Acc}\}$. Nodes of the form $(\cdot)_N$ do not occur in *Acc*. Thus, ε is not in $L((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)$. We can thus rewrite $L((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)$ as $\{s_i v | s_i \in \Sigma, v \in \Sigma^*, \hat{\delta}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N, s_i v) \in \mathit{Acc}\}$. By applying distributivity over the union we obtain

$$L((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N) = \bigcup_{i=1}^{n} \{s_i\} \cdot \{v \in \Sigma^* | \hat{\delta}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N, s_i v) \in \mathit{Acc}\}.$$

According to Definition 4.10, the condition $\hat{\delta}(\delta((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N, s_i), v) \in \mathit{Acc}$ is equivalent to $\hat{\delta}(\mathcal{T}_i, v) \in \mathit{Acc}$, which by induction is equivalent to $v \in \mathit{lang}(\mathcal{T}_i)$. Consequently, we have

$$L((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N) = \bigcup_{i=1}^{n} \{s_i\} \cdot \mathit{lang}(\mathcal{T}_i),$$

which equals definition of $\mathit{lang}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)$.

$\triangleright$ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$: The argument for nodes marked $\underline{\mathit{Fin}}$ is similar to the previous case. Here, the automaton will accept ε, so that with the arguments from (3) above, we can infer

$$L((\mathcal{T}_1, \ldots, \mathcal{T}_n)_F) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{s_i\} \cdot \mathit{lang}(\mathcal{T}_i),$$

which equals definition of $\mathit{lang}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_F)$. □

We will now show that—not surprisingly—every treelike automaton can be represented by a deterministic finite automaton. The class $\mathit{Treelike}(\Sigma)$ therefore is a special representation of a subclass of $\mathit{DFA}(\Sigma)$.

Lemma 4.16 Relation between treelike and finite automata
Let $\mathcal{T} \in \mathit{Treelike}(\Sigma)$ be a treelike automaton. Then there is a deterministic finite automaton $\mathcal{A} \in \mathit{DFA}(\Sigma)$ with $L(\mathcal{T}) = L(\mathcal{A})$.

Proof.
Define $\mathcal{A} \in \mathit{DFA}(\Sigma)$ as $\mathcal{A} := \langle Q, \Sigma, \delta_{\mathcal{A}}, q_{\mathcal{T}}, F\rangle$, where

- $Q := \{q_{\mathcal{S}} | \mathcal{S}$ subtree of $\mathcal{T}\}$ is the set of states with initial state $q_{\mathcal{T}}$,
- $F := \{q_{\mathcal{S}} \in Q | \mathcal{S} \in \mathit{Acc}\}$ is the set of finite states, and
- for $q_{\mathcal{S}} \in Q$ and $s_i \in \Sigma$ the transition function $\delta_{\mathcal{A}}$ is defined by: $\delta_{\mathcal{A}}(q_{\mathcal{S}}, s_i) := q_{\delta(\mathcal{S}, s_i)}$.

Observe that Q is finite and closed under $\delta_{\mathcal{A}}$. $\delta_{\mathcal{A}}$ is a deterministic transition function. For the extended transition function $\overline{\delta_{\mathcal{A}}}\colon Q \times \Sigma^* \to Q$ we can derive by induction over the length of w that: $\overline{\delta_{\mathcal{A}}}(q_{\mathcal{S}}, w) = q_{\hat{\delta}(\mathcal{S}, w)}$.
We now prove the equivalence of the *DFA* and the treelike automaton. According to the common definition of *DFA*, the language accepted by $\mathcal{A}$ is $\{w \in \Sigma^* | \overline{\delta_{\mathcal{A}}}(q_{\mathcal{T}}, w) \in F\}$. We have mentioned that $\overline{\delta_{\mathcal{A}}}(q_{\mathcal{T}}, w) = q_{\hat{\delta}(\mathcal{S}, w)}$ for any word w. Since F is defined as $\{q_{\mathcal{S}} \in Q | \mathcal{S} \in \mathit{Acc}\}$, we obtain

$$L(\mathcal{A}) = \{w \in \Sigma^* | q_{\hat{\delta}(\mathcal{S}, w)} \in \{q_{\mathcal{S}} \in Q | \mathcal{S} \in \mathit{Acc}\}\}.$$

The condition for the set holds iff $\hat{\delta}(\mathcal{S}, w) \in \mathit{Acc}$, which is equivalent to $w \in L(\mathcal{T})$. □

The directed graph representing the automaton introduced in the above lemma does not necessarily form a tree. In treelike automata, identical subtrees can occur at different positions in the tree. The definition of the set of states Q above automatically maps identical states onto one state. Consequently, the size of the equivalent *DFA* may be smaller than that of original treelike automaton. Note also that the deterministic finite automaton $\mathcal{B}$ proposed in Example 4.11 is not obtained from the analogous treelike automaton $\mathcal{A}$ by performing the construction of Lemma 4.16.

We now introcude auxiliary functions to simplify the construction of treelike automata for the recognition of two simple classes of languages.

Definition 4.17 Induced treelike automata
The functions *ind* and *ind'* are inductively defined as follows.

$$\begin{aligned}
\mathit{ind}\colon \Sigma^* &\to \mathit{Treelike}(\Sigma) \\
\varepsilon &\mapsto (\underline{\mathit{Nil}}, \ldots, \underline{\mathit{Nil}})_F \\
s_i v &\mapsto (\underbrace{\underline{\mathit{Nil}}, \ldots, \underline{\mathit{Nil}}}_{i-1}, \mathit{ind}(v), \underbrace{\underline{\mathit{Nil}}, \ldots, \underline{\mathit{Nil}}}_{n-i})_N \\
\mathit{ind}'\colon \Sigma^* &\to \mathit{Treelike}(\Sigma) \\
\varepsilon &\mapsto \underline{\mathit{Cyc}} \\
s_i v &\mapsto (\underbrace{\underline{\mathit{Nil}}, \ldots, \underline{\mathit{Nil}}}_{i-1}, \mathit{ind}'(v), \underbrace{\underline{\mathit{Nil}}, \ldots, \underline{\mathit{Nil}}}_{n-i})_N
\end{aligned}$$

□

Given a word w, the function *ind* is supposed to return a treelike automaton accepting the language $\{w\}$. Similarly, $\mathit{ind}'(w)$ is supposed to accept $\{w\}\cdot\Sigma^*$. In the next lemma, we will prove that the functions *ind* and *ind'* in fact have the desired property.

Lemma 4.18 Properties *ind* and *ind′*
Let $w \in \Sigma^*$ be a word over Σ. Then

1. $L_{ind(w)} = \{w\}$ and $L_{ind'(w)} = \{w\}\cdot\Sigma^*$
2. The size $\|ind(w)\|$ of $ind(w)$ is linear in $|w|$. The execution of $ind(w)$ requires linear time in $|w|$.
3. The equivalent claim holds for ind'.

Proof.
▶ 1. Due to Lemma 4.15, it is sufficient to consider $lang(\mathcal{T})$ instead of $L(\mathcal{T})$. Proof by induction over the length of w.

▷ $|w| = 0$: Then: $w = \varepsilon$. We have $ind(\varepsilon) = (\underline{Nil}, \ldots, \underline{Nil})_F$ according to Definition 4.17. $lang((\underline{Nil}, \ldots, \underline{Nil})_F)$ equals $\{\varepsilon\} \cup lang((\underline{Nil}, \ldots, \underline{Nil})_N)$ and finally, $lang((\underline{Nil}, \ldots, \underline{Nil})_N)$ equals $\bigcup_{i=1}^{n}\{s_i\}\cdot lang(\underline{Nil})$. By definition, $lang(\underline{Nil})$ is empty. Thus, for $lang(ind(\varepsilon))$, we end up with $\{\varepsilon\}$.

The reasoning for $ind'(\varepsilon)$ is similar. $ind'(\varepsilon)$ returns $\underline{Cyc}$ and $lang(\underline{Cyc}) = \Sigma^*$. Since $\Sigma^* = \{\varepsilon\}\cdot\Sigma^*$, this case is correct.

▷ $|w| > 0$: Then there exist $s_i \in \Sigma$ and $v \in \Sigma^*$ with $w = s_i v$. By definition, $lang(ind(s_i v))$ equals $lang((\underline{Nil}, \ldots, \underline{Nil}, ind(v), \underline{Nil}, \ldots, \underline{Nil})_N)$. We have already seen in ($|w| = 0$) that all leave-nodes marked $\underline{Nil}$ do not contribute to the accepted language. Thus, he have $lang(ind(s_i v)) = \{s_j\}\cdot lang(ind(v))$. By induction, this equals $\{s_j\}\cdot\{v\}$, which completes the argument.

For $ind'(s_i v)$, the proof is identical except for the induction argument. Here we can assume that $lang(ind'(v))$ equals $\{v\}\cdot\Sigma^*$. The rest of the conclusion remains the same.

▶ 2. Upon input $w \in \Sigma^*$, exactly one character of w is removed by ind in every step of recursion. Simultaneously, the automaton to be assembled is expanded by a constant amount of space. That amount is linear in the size $|\Sigma|$ of the alphabet and is thus constant in $|w|$. Therefore, the expansion of the automaton costs a constant amount of time in every step. The number of steps is linear in $|w|$. Thus, we require only linear time in $|w|$ to assemble $ind(w)$.

▶ 3. Analogous to (2). □

We have seen as a consequence of Lemma 4.16 that the language accepted by a treelike automaton is regular, since one can always construct an equivalent *DFA*. We will now examine further the structure of languages accepted by treelike automata and show that there are regular languages which cannot be accepted by any of them.

Lemma 4.19 Structure of $\mathcal{L}(\mathcal{T})$

1. For every treelike automaton $\mathcal{T} \in Treelike(\Sigma)$ there exist finite languages $L, L' \subseteq \Sigma^*$ with: $L_{\mathcal{T}} = L \cup L'\cdot\Sigma^*$.
2. For any finite languages $L, L' \subseteq \Sigma^*$ there is a treelike automaton $\mathcal{T} \in Treelike(\Sigma)$ with: $L_{\mathcal{T}} = L \cup L'\cdot\Sigma^*$.

Proof.
▶ 1. Because of 4.15 we can again resort to $lang(\mathcal{T})$ instead of examining $L(\mathcal{T})$. Proof by structural induction over $\mathcal{T}$.

▷ $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$: Per definition, $lang(\underline{Nil}) = \emptyset = \emptyset \cup \emptyset \cdot \Sigma^*$. Consequently, with $L = L' = \emptyset$ the claim holds. Similar for $\underline{Cyc}$: $lang(\underline{Cyc}) = \Sigma^* = \emptyset \cup \{\varepsilon\} \cdot \Sigma^*$. By defining $L = \emptyset$ and $L' = \{\varepsilon\}$, we again have the desired result.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$: $lang((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)$ equals $\bigcup_{i=1}^{n} \{s_i\} \cdot lang(\mathcal{T}_i)$. By induction, we may assume that there exist finite languages L and L' with $lang(\mathcal{T}_i) = (L_i \cup L_i' \cdot \Sigma^*)$. Applying distributivity over union then yields

$$lang((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N) = \bigcup_{i=1}^{n} \{s_i\} \cdot L_i \cup \bigcup_{i=1}^{n} \{s_i\} \cdot L_i' \cdot \Sigma^*$$

Therefore, by choosing $L = \bigcup_{i=1}^{n} \{s_i\} \cdot L_i$ and $L' = \bigcup_{i=1}^{n} \{s_i\} \cdot L_i'$ the argument is complete.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$: This case is equivalent to the previous one. The only difference is that now $\{\varepsilon\}$ is included in $lang((\mathcal{T}_1, \ldots, \mathcal{T}_n)_F)$. thus, we can choose $L = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{s_i\} \cdot L_i$ and $L' = \bigcup_{i=1}^{n} \{s_i\} \cdot L_i'$ to succeed.

► 2. According to Lemma 4.18 for every $w \in \Sigma^*$ there exist treelike automata $\mathcal{T}, \mathcal{T}'$ with: $L_{\mathcal{T}} = \{w\}$ and $L_{\mathcal{T}'} = \{w\} \cdot \Sigma^*$. The languages L and L' are finite. Moreover, in the next section we will show by Lemma 4.20 that $Treelike(\Sigma)$ is closed under finite union. Taking into account these two arguments, we can draw the proposed conclusion. □

So far we have no reasonable methods to construct more complex treelike automata. In the last lemma it became apparent that a scheme to construct the union of treelike automata is desirable. In the next section, we will introduce appropriate operations for treelike automata to facilitate this.

4.2.3 Operations on treelike automata

In Definition 4.12 the notion of trimmed treelike automata was introduced. Now we will propose methods for actually trimming an automaton. However, there is a second class of irregularities we seek to eliminate. Apart from nodes which contribute nothing to the accepted language, it is possible in a treelike automaton that large subtrees merely acccept every input whatsoever. In such a case, it would be appropriate to replace the entire subtree by a $\underline{Cyc}$-node, thus minimizing the automaton.

At this point it is not clear why minimized automata are considered important for our reasoning. At last, we want to use treelike automata to establish an algorithm for solving matching problems which meets certain complexity bounds. For that purpose we have to guarantee that, when constructing a treelike automaton for a certain language, the size of the automaton is limited in the size of the language to represent. The auxiliary functions introduced next will prove to eliminate that problem.

For any treelike automata $\mathcal{T}_1, \ldots, \mathcal{T}_n$ the operations trim (*trim*) and simplify (*simp*) are inductively defined as follows.

$$\begin{aligned}
trim\colon\ Treelike(\Sigma) &\to Treelike(\Sigma) \\
\underline{Nil} &\mapsto \underline{Nil} \\
\underline{Cyc} &\mapsto \underline{Cyc} \\
(\mathcal{T}_1, \ldots, \mathcal{T}_n)_N &\mapsto \begin{cases} \underline{Nil} & \text{if for all } i\colon trim(\mathcal{T}_i) = \underline{Nil} \\ (trim(\mathcal{T}_1), \ldots, trim(\mathcal{T}_n))_N & \text{otherwise} \end{cases} \\
(\mathcal{T}_1, \ldots, \mathcal{T}_n)_F &\mapsto (trim(\mathcal{T}_1), \ldots, trim(\mathcal{T}_n))_F
\end{aligned}$$

trim is intended to remove all nodes of the form $(\underline{Nil},\dots,\underline{Nil})_N$ and replace them by $\underline{Nil}$ which is equivalent in regard to the accepted language. However, this process has to be done recursively. The replacement of one node by $\underline{Nil}$ may change the predecessor node to $(\underline{Nil},\dots,\underline{Nil})_N$, which then has to be replaced as well. The idea is to remove all nodes contributing nothing to the accepted language. Nodes of the form $(\cdot)_F$ are never removed because they accept $\{\varepsilon\}$.
Like *trim*, the purpose of *simp* is to remove subtrees and replace them by simple equivalents. Nodes of the form $(\underline{Cyc},\dots,\underline{Cyc})_F$ accept Σ^* and can be replaced by $\underline{Cyc}$. Again, that modification is carried out recursively to eliminate all such cases.

$$
\begin{aligned}
\mathit{simp}\colon \mathit{Treelike}(\Sigma) &\to \mathit{Treelike}(\Sigma)\\
\underline{Nil} &\mapsto \underline{Nil}\\
\underline{Cyc} &\mapsto \underline{Cyc}\\
(\mathcal{T}_1,\dots,\mathcal{T}_n)_N &\mapsto (\mathit{simp}(\mathcal{T}_1),\dots,\mathit{simp}(\mathcal{T}_n))_N\\
(\mathcal{T}_1,\dots,\mathcal{T}_n)_F &\mapsto \begin{cases} \underline{Cyc} & \text{if for all } i\colon \mathit{simp}(\mathcal{T}_i) = \underline{Cyc}\\ (\mathit{simp}(\mathcal{T}_1),\dots,\mathit{simp}(\mathcal{T}_n))_F & \text{otherwise}\end{cases}
\end{aligned}
$$

Taking advantage of these functions we can now introduce operations on treelike automata. Most of them are intended to refer to set-theoretic operations on the sets represented by treelike automata, i.e. the complement of a treelike automata $\overline{\mathcal{A}}$ is intended to accept the complemented language of $\mathcal{A}$.
For any treelike automata $\mathcal{T}_1,\dots,\mathcal{T}_n,\mathcal{S}_1,\dots,\mathcal{S}_n \in \mathit{Treelike}(\Sigma)$ and $w \in \Sigma^*$ the operations left quotient $(w^{-1}(\cdot))$, complement (c), trimmed complement $(\overline{\cdot})$, intersection $(\cap')$, trimmed intersection $(\cap)$, and union $(\cup)$ are defined inductively.

$$
\begin{aligned}
w^{-1}\colon \mathit{Treelike}(\Sigma) &\to \mathit{Treelike}(\Sigma)\\
\mathcal{T}_1 &\mapsto \hat{\delta}(\mathcal{T}_1, w)
\end{aligned}
$$

$$
\begin{aligned}
c\colon \mathit{Treelike}(\Sigma) &\to \mathit{Treelike}(\Sigma)\\
\underline{Nil} &\mapsto \underline{Cyc}\\
\underline{Cyc} &\mapsto \underline{Nil}\\
(\mathcal{T}_1,\dots,\mathcal{T}_n)_N &\mapsto (c(\mathcal{T}_1),\dots,c(\mathcal{T}_n))_F\\
(\mathcal{T}_1,\dots,\mathcal{T}_n)_F &\mapsto (c(\mathcal{T}_1),\dots,c(\mathcal{T}_n))_N
\end{aligned}
$$

Let us first consider the unary operations. The left quotient is supposed to return the subtree of a treelike automaton after reading a word w. The state of an automaton is an automaton itself, so we just have to return the result of the transition function. The complement also works recursively. Since $\underline{Nor}$- and $\underline{Fin}$-nodes differ in accepting ε or not, the labels of these nodes have to be exchanged.
For the sake of brevity, all binary operations are defined commutatively without explicitly repeating symmetric patterns.

$$
\begin{aligned}
\cap'\colon \mathit{Treelike}(\Sigma) \times \mathit{Treelike}(\Sigma) &\to \mathit{Treelike}(\Sigma)\\
\underline{Nil}\,,\, \mathcal{T} &\mapsto \underline{Nil}\\
\underline{Cyc}\,,\, \mathcal{T} &\mapsto \mathcal{T}\\
(\mathcal{T}_1,\dots,\mathcal{T}_n)_N\,,\, (\mathcal{S}_1,\dots,\mathcal{S}_n)_* &\mapsto (\mathcal{T}_1 \cap' \mathcal{S}_1,\dots,\mathcal{T}_n \cap' \mathcal{S}_n)_N\\
(\mathcal{T}_1,\dots,\mathcal{T}_n)_F\,,\, (\mathcal{S}_1,\dots,\mathcal{S}_n)_F &\mapsto (\mathcal{T}_1 \cap' \mathcal{S}_1,\dots,\mathcal{T}_n \cap' \mathcal{S}_n)_F
\end{aligned}
$$

When regarding $\underline{Nor}$ as "False" and $\underline{Fin}$ as "True", the intersection takes the logical conjunction as resulting label. The union operator, as defined below, takes the disjunction.

This is not surprising, because in the intersection-automaton only those states may be accepting states which have been accepting states in both input automata. An analogous argument applies to the union-automaton.

$$\begin{aligned}
\cup\colon \mathit{Treelike}(\Sigma) \times \mathit{Treelike}(\Sigma) &\to \mathit{Treelike}(\Sigma)\\
\underline{\mathit{Nil}}\,,\, \mathcal{T} &\mapsto \mathcal{T}\\
\underline{\mathit{Cyc}}\,,\, \mathcal{T} &\mapsto \underline{\mathit{Cyc}}\\
(\mathcal{T}_1,\ldots,\mathcal{T}_n)_F\,,\, (\mathcal{S}_1,\ldots,\mathcal{S}_n)_* &\mapsto (\mathcal{T}_1\cup\mathcal{S}_1,\ldots,\mathcal{T}_n\cup\mathcal{S}_n)_F\\
(\mathcal{T}_1,\ldots,\mathcal{T}_n)_N\,,\, (\mathcal{S}_1,\ldots,\mathcal{S}_n)_N &\mapsto (\mathcal{T}_1\cup\mathcal{S}_1,\ldots,\mathcal{T}_n\cup\mathcal{S}_n)_N
\end{aligned}$$

Observe that an explicit definition of the union is in fact not necessary because we have introduced the complement and the intersection. The alternative proposed above, however, may prove more efficient when actually implenting algorithms based on treelike automata. The complexity class, as we will see, is not affected.

When intersecting trimmed automata, the trimming-property can get lost. The complement has the same disadvantage. To overcome this we introduce modified versions of these operations.

$$\begin{aligned}
\cap &: \mathit{Treelike}(\Sigma) \to \mathit{Treelike}(\Sigma)\\
\cap &:= \mathit{trim} \circ \cap'\\
\overline{\cdot} &: \mathit{Treelike}(\Sigma) \to \mathit{Treelike}(\Sigma)\\
(\overline{\cdot}) &:= \mathit{trim} \circ c
\end{aligned}$$

Thus, the correction is achieved by simply applying the *trim* operation at the end.
The left-quotient of a treelike automaton $\mathcal{T}$ is denoted as application of a function, i.e. $w^{-1}(\mathcal{T})$, and not like a product. In this respect we deviate from the notation for the left quotient of formal languages in Definition 2.3. A notation like $w^{-1}\cdot\mathcal{T}$ could mislead to the impression that there is a concatenation for treelike automata.
Observe that in most of the cases the operations defined above could be realized by merely changing node labels in an appropriate way. Nevertheless, it has not been shown yet that these operations yield the desired results. We will give a proof of correctness first and then examine the complexity of the operations.

Lemma 4.20 Correctness of the operations
Let $w \in \Sigma^*$ be a word. and let $\mathcal{T}, \mathcal{S} \in \mathit{Treelike}(\Sigma)$ be treelike automata. Then

1. $\mathit{trim}(\mathcal{T})$ is trimmed and $L(\mathit{trim}(\mathcal{T})) = L(\mathcal{T})$.
2. $L(\mathit{simp}(\mathcal{T})) = L(\mathcal{T})$
3. $L(w^{-1}(\mathcal{T})) = w^{-1}\cdot L(\mathcal{T})$
4. $L(\overline{\mathcal{T}}) = \overline{L(\mathcal{T})}$
5. $L(\mathcal{T}\cap\mathcal{S}) = L(\mathcal{T}) \cap L(\mathcal{S})$
6. $L(\mathcal{T}\cup\mathcal{S}) = L(\mathcal{T}) \cup L(\mathcal{S})$

Proof.
▶ 1. Prove trimming-property: $\mathcal{T}$ is trimmed if and only if no subtree of $\mathcal{T}$ is of the form $(\underline{\mathit{Nil}},\ldots,\underline{\mathit{Nil}})_N$. One can see, that *trim* recursively removes subtrees of that form, replacing then with $\underline{\mathit{Nil}}$. Thus, the resulting tree has the desired property.

Proof of the equality of the accepted languages by structural induction over $\mathcal{T}$.

▷ $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$: Trivial, because *trim* does not change $\underline{Nil}$ or $\underline{Cyc}$.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ or $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$: First case: $trim(\mathcal{T}_i) = \underline{Nil}$ for all i. Then: $lang(trim((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)) = lang(\underline{Nil}) = \emptyset$ by definition of *trim*. By induction, we obtain $lang(trim(\mathcal{T}_i)) = lang(\mathcal{T}_i)$, which equals $lang(\underline{Nil})$ according to the assumption. Since $lang(\underline{Nil}) = \emptyset$, we obtain by definition of *lang*: $lang((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N) = \bigcup_{i=1}^{n} \{s_i\} \cdot \emptyset$, which simplifies to $\emptyset$.

Second case: There is an i with $trim(\mathcal{T}_i) \neq \underline{Nil}$. If not all $\mathcal{T}_i$ are trimmed to $\underline{Nil}$, *trim* does not affect the label of the root node. By induction, we have $lang(trim(\mathcal{T}_i)) = lang(\mathcal{T}_i)$. Consequently, *trim* does not change the accepted langauge.

If $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$, *trim* does not affect the root node by definition. Thus, the argument of the second case applies again.

▶ 2. Taking advantage of the analogous definitions of *trim* and *simp*, we can proof the correctness of *simp* in the same fashion as for *trim*. Here the criterion is not evaluating to $\underline{Nil}$, but evaluationg to $\underline{Cyc}$. Moreover, this time nodes marked $\underline{Nor}$ remain unchanged by instead of $\underline{Fin}$ in the second case above.

▶ 3. For $w = \varepsilon$, the left quotient has no effect. Furthermore, $(wv)^{-1} \cdot L = v^{-1} \cdot (w^{-1} \cdot L)$ for $v \in \Sigma^*$ and for every language L. Consequently, left quotients for longer words w can be obtained by successively applying the left quotient for only one character. It is therefore sufficient to consider only words w of length 1 in our proof, i.e. $w = s_i$. Then we have: $L(s_i^{-1}(\mathcal{T})) = \{v \in \Sigma^* | \hat{\delta}(s_i^{-1}(\mathcal{T}), v) \in Acc\}$. According to the definition of the left quotient, $\hat{\delta}(s_i^{-1}(\mathcal{T}), v) = \hat{\delta}(\hat{\delta}(\mathcal{T}, s_i), v)$, which by definition of $\hat{\delta}$ equals $\hat{\delta}(\mathcal{T}, s_i v)$. Hence, taking all words v for which $\hat{\delta}(\mathcal{T}, s_i v) \in Acc$ is equivalent to taking the left quotient $s^{-1} \cdot L(\mathcal{T})$.

▶ 4. Due to (1), it is sufficient to prove the proposition for the complement (c) instead of the trimmed complement. Proof by structural induction over $\mathcal{T}$.

▷ $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$: According to *lang*, we have $lang(\underline{Nil}) = \emptyset$ and $lang(\underline{Cyc}) = \Sigma^*$. $\underline{Nil}$ and $\underline{Cyc}$ are complementary with regard to (c), their languages are complementary with regard to the complement of formal languages. Thus, c is correct for $\underline{Nil}$ and $\underline{Cyc}$.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ or $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$: ($c$) changes the node label and proceeds to the successors. We thus obtain $lang(c((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{s_i\} \cdot lang(c(\mathcal{T}_i))$ by definition of *lang*, which by induction can be replaced by $\{\varepsilon\} \cup \bigcup_{i=1}^{n} \{s_i\} \cdot \overline{lang(\mathcal{T}_i)}$. We now have to show that this equals $\overline{lang((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)}$.

For a word $w \in \Sigma^*$, it holds that $w \notin lang((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)$ iff $w \notin \bigcup_{i=1}^{n} \{s_i\} \cdot lang(\mathcal{T}_i)$. This is equivalent to $w = \varepsilon$ or, for some character $s_i \in \Sigma$ and $v \in \Sigma^*$, $w = s_i v$ such that $v \notin lang(\mathcal{T}_i)$. Therefore, w is an element of $\{\varepsilon\} \cup \bigcup_{i=1}^{n} \{s_i\} \cdot \overline{lang(\mathcal{T}_i)}$, which is the complement of $lang((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)$ according to the definition of (c).

The proof for $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$ is identical except for the empty word ε missing in the accepted language.

▶ 5. Because of (1), we will consider the intersection ($\cap'$) and not the trimmed intersection ($\cap$). Proof by induction over the structure of $\mathcal{T}$ and $\mathcal{S}$.

▷ $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$: The intersection of $lang(\underline{Nil})$ with any other language is empty. By definition, $\underline{Nil} \cap' \mathcal{S}$ is $\underline{Nil}$ for any treelike automaton $\mathcal{S}$, so that $lang(\underline{Nil} \cap' \mathcal{S})$ is empty as well. Similarly, intersecting $lang(\underline{Cyc})$ with any other language yields Σ^*. As $\underline{Cyc} \cap' \mathcal{S} = \underline{Cyc}$, the language accepted by $\underline{Cyc} \cap' \mathcal{S}$ is also Σ^*.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ and $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)_N$: By definition, we have that $\mathcal{T} \cap' \mathcal{S}$ is equal to $(\mathcal{T}_1 \cap' \mathcal{S}_1, \ldots, \mathcal{T}_n \cap' \mathcal{S}_n)_N$. Therefore, $lang(\mathcal{T} \cap' \mathcal{S})$ equals $\bigcup_{i=1}^{n} \{s_i\} \cdot lang(\mathcal{T}_i \cap' \mathcal{S}_i)$. By induction, $\{s_i\} \cdot lang(\mathcal{T}_i \cap' \mathcal{S}_i)$ equals $\{s_i\} \cdot (lang(\mathcal{T}_i) \cap lang(\mathcal{S}_i))$. The characters of the alphabet Σ are assumed to be uniqe. This allows us to apply distributivity over the intersection, yielding $\bigcup_{i=1}^{n} \{s_i\} \cdot lang(\mathcal{T}_i) \cap \bigcup_{i=1}^{n} \{s_i\} \cdot lang(\mathcal{S}_i)$, which matches the definition of $lang(\mathcal{T}) \cap lang(\mathcal{S})$.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ and $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)_F$: Similar to the previous case. Here ε is an element of $lang(\mathcal{S})$. It disappears when applying the intersection operation ($\cap'$) as well as when intersecting the actual accepted languages of $\mathcal{T}$ and $\mathcal{S}$. Thus, the same argument holds.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$, $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)_F$: Analogous to the previous cases. This time the empty word ε appears in the languages of both $\mathcal{T}$ and $\mathcal{S}$.

▶ 6. Proof by induction over the structure of $\mathcal{T}$ and $\mathcal{S}$. Due to the great similarity to the intersection operation we will not give the proof in full detail. Whereas previously the intersection with Σ^* causes no change, here the union with $\emptyset$ changes nothing. The same analogy exists between intersecting with $\emptyset$ an uniting with Σ^*. Therefore, for case $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$, the same arguments hold.

The other cases are also obtained in the way seen above. Here, however, we do not even require the argument of Σ consisting of distinct characters. Distributivity over the union could be applied even without this. □

Let us now study the complexity of treelike automata. In the next lemma, we discuss the effect of the above operations on the size of the resulting automata. We will see that the size of the resulting automaton never exceeds the sum of the sizes of the original automata.

Lemma 4.21 Size of the constructed automata
Consider a word $w \in \Sigma^*$ and treelike automata $\mathcal{T}, \mathcal{S} \in Treelike(\Sigma)$. Then

1. $\|trim(\mathcal{T})\| \leq \|\mathcal{T}\|$ and $\|simp(\mathcal{T})\| \leq \|\mathcal{T}\|$
2. $\|w^{-1}(\mathcal{T})\| \leq \|\mathcal{T}\|$
3. $\|\overline{\mathcal{T}}\| \leq \|\mathcal{T}\|$
4. $\|\mathcal{T} \cap \mathcal{S}\| < \|\mathcal{T}\| + \|\mathcal{S}\|$ and $\|\mathcal{T} \cup \mathcal{S}\| < \|\mathcal{T}\| + \|\mathcal{S}\|$

Proof.
▶ 1. Applied on $\underline{Nil}$ or $\underline{Cyc}$, *trim* does not change anything. Applied on nodes of the form $(\mathcal{T}_1, \ldots, \mathcal{T}_n)_*$ however, it replaces subtrees of size greater than 1 by $\underline{Nil}$, which is of size 1. Therefore, the size cannot increase when applying *trim*. The same argument holds for *simp*.

▶ 2. The left quotient operation by definition returns a subtree of the automaton it is applied to. Obviously, a treelike automaton is never of a smaller size than one of its subtrees.

▶ 3. Because of the result of (1), it is sufficient to consider (c) instead of $(\overline{\cdot})$. By induction over the structure of $\mathcal{T}$, we prove that $|c(\mathcal{T})| = |\mathcal{T}|$.

▷ $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$: $\underline{Nil}$ and c are complementary in regard to (c). Moreover, the size of both is the same which implies that it remains the same when applying (c).

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ or $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$: By definition of c, $\|(\mathcal{T}_1, \ldots, \mathcal{T}_n)_N\|$ is equal to $\|(c(\mathcal{T}_1), \ldots, c(\mathcal{T}_n))_N\|$, which simplifies to $1+\sum_{i=1}^{n} \|c(\mathcal{T}_i)\|$. By induction, $\|c(\mathcal{T}_i)\| = \|\mathcal{T}_i\|$, so that $\|c(\mathcal{T})\| = 1 + \sum_{i=1}^{n} |\mathcal{T}_i|$ which matches the definition of $\|\mathcal{T}\|$.

▶ 4. Again, we only need to give a proof for the intersection ($\cap'$) without trimming. Proof by induction over the structure of $\mathcal{T}$.

▷ $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$: Then $\mathcal{T} \cap' \mathcal{S} \in \{\mathcal{T}, \mathcal{S}\}$ according to the definition. Both $\|\mathcal{T}\|$ and $\|\mathcal{S}\|$ are integers greater than 0. Consequently, the size of the intersection automaton must be smaller than the sum of the sizes of $\mathcal{T}$ and $\mathcal{S}$.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_*$ and $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)_*$: Applying the definition of the intersection and that of $\|\cdot\|$, we obtain that $\|\mathcal{T} \cap' \mathcal{S}\|$ equals $1+\sum_{i=1}^{n} \|\mathcal{T}_i \cap' \mathcal{S}_i\|$. By induction, this is less than $1+\sum_{i=1}^{n} (\|\mathcal{T}_i\|+\|\mathcal{S}_i\|)$. Adding 1 and splitting the sum yields $1+\sum_{i=1}^{n} \|\mathcal{T}_i\| + 1+\sum_{i=1}^{n} \|\mathcal{S}_i\|$, which is equivalent to $\|\mathcal{T}\| + \|\mathcal{S}\|$.

Because of the symmetric definitions of the intersection and the union, the same argument holds for the union of $\mathcal{T}$ and $\mathcal{S}$. □

Regarding time complexity, we can establish similar results. The next lemma will show that all operations take only linear time in the size of the input automata.

Lemma 4.22 Time complexity
Let $w \in \Sigma^*$ be a word and let $\mathcal{T}, \mathcal{S} \in \mathit{Treelike}(\Sigma)$ be treelike automata. Then the following operations require only linear time in the size of the input automata:

1. *trim*, *simp*, and $(\overline{\cdot})$,
2. the left quotient $(w^{-1}(\cdot))$,
3. trimmed intersection $(\cap)$ and union $(\cup)$.

Proof.
▶ 1. Proof by induction over the structure of $\mathcal{T}$.

▷ $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$: Upon input $\underline{Nil}$ or $\underline{Cyc}$, the three functions *trim*, *simp*, and c immediately return $\underline{Nil}$ or $\underline{Cyc}$ als result. For all of them it takes constant time to identify the input automaton as one of $\underline{Nil}$ or $\underline{Cyc}$. Generating and returning the result also requires only constant time. The trimmed complement operation $(\overline{\cdot})$ is defined as concatenation of *trim* and c and therefore also is finished in constant time in the size of the input.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_*$: Consider $\mathit{trim}(\mathcal{T})$. By induction, it requires only linear time in $\|\mathcal{T}_i\|$ to compute the results for $\mathit{trim}(\mathcal{T}_i)$, $\mathit{simp}(\mathcal{T}_i)$, and $c(\mathcal{T}_i)$ respectively. The test for $\mathit{trim}(\mathcal{T}_i) = \underline{Nil}$ requires only constant time for all i. Since Σ is assumed to be constant and since the number of subtrees equals $|\Sigma|$, testing all $\mathcal{T}_i$ costs only constant time in $\|(\mathcal{T}_1, \ldots, \mathcal{T}_n)_*\|$. The final result of $\mathit{trim}(\mathcal{T})$ can be computed from the results of $\mathit{trim}(\mathcal{T}_i)$ in constant time. Altogether, we can infer linear time in the sum of all $\|\mathcal{T}_i\|$ and thus linear time in $\|(\mathcal{T}_1, \ldots, \mathcal{T}_n)_*\|$.

For *simp* and c, the same argument holds.

▶ 2. For $w \in \Sigma^*$, the computation of $\hat{\delta}(\mathcal{T}, w)$ costs only linear time in $|w|$ and $\|\mathcal{T}\|$. Returning the resulting subtree also requires only linear time. Altogether, we obtain linear time complexity.

▶ 3. Taking advantage of (1), it is sufficient to consider ($\cap'$) instead of ($\cap$). Proof by induction over the structure of $\mathcal{T}$.

▷ $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$: $\underline{Nil} \cap' \mathcal{S}$ and $\underline{Cyc} \cap' \mathcal{S}$ return either $\underline{Nil}$ or $\underline{Cyc}$. Since the test for $\mathcal{T} = \underline{Nil}$ or $\mathcal{T} = \underline{Cyc}$ requires only constant time, the result can be computed in constant time. The same argument applies to the union operation ($\cup$).

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_*$: By induction, for all i the results $\mathcal{T}_i \cap' \mathcal{S}_i$ can be computed in linear time in $\|\mathcal{T}_i\|$ and $\|\mathcal{S}_i\|$. The time necessary to assemble the final results from that is linear in $|\Sigma|$ and therefore constant in $\|(\mathcal{T}_1, \ldots, \mathcal{T}_n)_*\|$. Consequently, we obtain linear time complexity in $\|(\mathcal{T}_1, \ldots, \mathcal{T}_n)_*\|$ for the intersection.

The same argument holds for the union ($\cup$). □

The advantage of *trim* and *simp* is to provide a method for generating a better representation for an automaton accepting some language. In the view of the rather simple structure of treelike automata the question arises if there is a unique representation for every automaton, i.e. a normal form which cannot be simplified further. The next lemma shows that such a representation can be defined using *trim* and *simp*.

Lemma 4.23 Normal forms
Let $\mathcal{T}, \mathcal{S} \in \mathit{Treelike}(\Sigma)$ be treelike automata. Then

1. $\mathcal{T} \equiv \mathcal{S}$ iff $\mathit{simp}(\mathit{trim}(\mathcal{T})) = \mathit{simp}(\mathit{trim}(\mathcal{S}))$
2. Testing for equivalence requires only linear time in $\|\mathcal{T}\| + \|\mathcal{S}\|$.

Proof.
▶ 1. ("⇒") Proof by induction over the structure of $\mathcal{T}$ and $\mathcal{S}$.

▷ $\mathcal{T}, \mathcal{S} \in \{\underline{Nil}, \underline{Cyc}\}$: Since $\mathit{simp}(\mathit{trim}(\underline{Nil})) = \underline{Nil}$ and $\mathit{simp}(\mathit{trim}(\underline{Cyc})) = \underline{Cyc}$ and since $\mathit{lang}(\underline{Nil}) \neq \mathit{lang}(\underline{Cyc})$, the proposition follows immediately.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ and $\mathcal{S} = \underline{Nil}$: Then $\mathcal{T}$ only accepts $\emptyset$ and we can derive $\mathit{lang}(\mathcal{T}_i) = \emptyset$ for every i, which means that all $\mathcal{T}_i$ are equivalent to $\underline{Nil}$. By induction, we then obtain $\mathit{simp}(\mathit{trim}(\mathcal{T}_i)) = \mathit{simp}(\mathit{trim}(\underline{Nil}))$, which equals $\underline{Nil}$ by definition of *simp* and *trim*. If $\mathit{simp}(\mathit{trim}(\mathcal{T}_i)) = \underline{Nil}$, then already $\mathit{trim}(\underline{Nil})$ must have been $\underline{Nil}$. If this is the case, then by definition $\mathit{trim}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N)$ is $\underline{Nil}$ and therefore $\mathit{simp}(\mathit{trim}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N))$ is as well. Utilizing that $\underline{Nil} = \mathit{simp}(\mathit{trim}(\underline{Nil}))$ we end up with $\mathit{simp}(\mathit{trim}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N))$ equal to $\mathit{simp}(\mathit{trim}(\underline{Nil}))$, which was to be shown.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$ and $\mathcal{S} = \underline{Cyc}$: Then $\mathit{lang}(\mathcal{T}) = \Sigma^*$ and so $\mathit{lang}(\mathcal{T}_i) = \Sigma^*$ for every i. This implies that every $\mathcal{T}_i$ is equivalent to $\underline{Cyc}$ which by induction yields the equality of $\mathit{simp}(\mathit{trim}(\mathcal{T}_i))$ and $\mathit{simp}(\mathit{trim}(\underline{Cyc}))$. Thus, $\mathit{simp}(\mathit{trim}(\mathcal{T}_i)) = \underline{Cyc}$ for every i.
As $\mathcal{T}$ is labelled $\underline{Fin}$, $\mathit{simp}(\mathit{trim}(\mathcal{T}))$ simplifies to $\mathit{simp}((\mathit{trim}(\mathcal{T}_1), \ldots, \mathit{trim}(\mathcal{T}_n))_F)$. Since always $\mathit{simp}(\mathit{trim}(\mathcal{T}_i)) = \underline{Cyc}$, the whole expression simplifies to $\underline{Cyc}$ by definition of *simp*. Exploiting again that $\underline{Cyc} = \mathit{simp}(\mathit{trim}(\underline{Cyc}))$ we obtain that $\mathit{simp}(\mathit{trim}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_F))$ equals $\mathit{simp}(\mathit{trim}(\underline{Cyc}))$.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ and $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)_N$: By definition of *lang*, we have that $\mathit{lang}(\mathcal{T}_i)$ equals $\mathit{lang}(\mathcal{S}_i)$ for all i, implying $\mathcal{T}_i \equiv \mathcal{S}_i$. By induction, we can infer that $\mathit{simp}(\mathit{trim}(\mathcal{T}_i))$ equals $\mathit{simp}(\mathit{trim}(\mathcal{S}_i))$ for all i.

First case: for all i it holds that $\mathit{trim}(\mathcal{T}_i) = \underline{Nil}$. Then $\mathit{simp}(\mathit{trim}(\mathcal{T}_i)) = \underline{Nil}$, so that by the above equation $\mathit{simp}(\mathit{trim}(\mathcal{S}_i)) = \underline{Nil}$. Therefore, $\mathit{trim}(\mathcal{S}_i) = \underline{Nil}$ for all i as well. As both $\mathcal{T}$ and $\mathcal{S}$ are labelled $\underline{Nor}$, by definition of *trim* this yields that $\mathit{simp}(\mathit{trim}((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N))$ equals $\mathit{simp}(\underline{Nil})$ and analogously $\mathit{simp}(\mathit{trim}((\mathcal{S}_1, \ldots, \mathcal{S}_n)_N))$ is equal to $\mathit{simp}(\underline{Nil})$. Consequently, $\mathit{simp}(\mathit{trim}(\mathcal{T}))$ equals $\mathit{simp}(\mathit{trim}(\mathcal{S}))$.

Second case: there is an i with $trim(\mathcal{T}_i) \neq \underline{Nil}$. Then *trim* will reduce neither $\mathcal{T}$ nor $\mathcal{S}$ to $\underline{Nil}$. As *simp* by definition does not reduce nodes labelled $\underline{Nor}$, the proposition is obtained by merely applying the definitions of *trim* and *simp*: $simp(trim((\mathcal{T}_1, \ldots, \mathcal{T}_n)_N))$ equals $simp((trim(\mathcal{T}_1), \ldots, trim(\mathcal{T}_n))_N)$ which equals $(simp(trim(\mathcal{T}_1)), \ldots, simp(trim(\mathcal{T}_n)))_N$. This is equivalent to $(simp(trim(\mathcal{S}_1)), \ldots, simp(trim(\mathcal{S}_n)))_N$, which can again be simplified to $simp((trim(\mathcal{S}_1), \ldots, trim(\mathcal{S}_n))_N)$, resulting in $simp(trim((\mathcal{S}_1, \ldots, \mathcal{S}_n)_N))$.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_F$ and $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)_F$: This case is fairly analogous to the previous one. We again can infer $simp(trim(\mathcal{T}_i))$ being equal to $simp(trim(\mathcal{S}_i))$ for all i and then distinguish two cases.

First case: for all i : $simp(trim(\mathcal{T}_i)) = \underline{Cyc}$. As *trim* does not reduce nodes marked $\underline{Fin}$, $simp(trim((\mathcal{T}_1, \ldots, \mathcal{T}_n)_F))$ simplifies to $simp((trim(\mathcal{T}_1), \ldots, trim(\mathcal{T}_n))_F)$. Since always $simp(trim(\mathcal{T}_i)) = \underline{Cyc}$, *simp* reduces the expression to $\underline{Cyc}$. The same transformation applies to $\mathcal{S}$, so that we obtain equality.

Second case: there is an i with: $simp(trim(\mathcal{T}_i)) \neq \underline{Cyc}$. In this case, neither *trim* nor *simp* reduce the root node. Similar to the second case above we can therefore prove the proposition by merely applying the definitions of *trim* and *simp*.

▷ $\mathcal{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)_N$ and $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)_F$: Then $\mathcal{T}$ and $\mathcal{S}$ cannot be equivalent because they do not agree upon accepting ε or not.

▶ 1. ("⇐") Assume $simp(trim(\mathcal{T})) = simp(trim(\mathcal{S}))$. According to Lemma 4.20, *simp* and *trim* do not alter the accepted language, so that $L(\mathcal{T})$ equals $L(simp(trim(\mathcal{T})))$ and $L(\mathcal{S})$ equals $L(simp(trim(\mathcal{S})))$. This implies $L(\mathcal{T}) = L(\mathcal{S})$, completing the proof.

▶ 2. According to Lemma 4.22, computing $simp(trim(\mathcal{T}))$ and $simp(trim(\mathcal{S}))$ requires only linear time in $\|\mathcal{T}\| + \|\mathcal{S}\|$. Due to Lemma 4.21, treelike automata cannot increase in size when trimming or simplifying them. It is therefore sufficient to find a strategy to syntactically compare the automata in linear time in the size of the automata.

Such a strategy can be defined easily. For instance, a simultaneous depth-first search over the trees corresponding to the automata has the required properties. □

Because of the properties shown in the previous lemma, $simp(trim(\mathcal{T}))$ could serve as a normal-form for the automaton $\mathcal{T}$. It is particularly interesting that such a normal form can be computed in linear time in the size of the automaton.

In the end we want to use treelike automata to represent regular languages occuring in the solvability equations introduced in the first section of this chapter. We have already seen that treelike automata can represent any language of the form $L \cup L' \cdot \Sigma^*$, where L and L' are finite languages. We have not yet provided a scheme to actually construct such an automaton, given languages L and L'. In the next lemma this will be provided.

Lemma 4.24 Representing languages by treelike automata
Let $L, L' \subseteq \Sigma^*$ be finite languages over Σ. Then there is a automaton $\mathcal{T} \in \mathit{Treelike}(\Sigma)$ which accepts von $L \cup L' \cdot \Sigma^*$ with:

1. $\|\mathcal{T}\| \in \mathcal{O}(\|L\| + \|L'\|)$.
2. The construction of $\mathcal{T}$ takes only linear time in $\|L\| + \|L'\|$.

Proof.
Construct the automaton $\mathcal{T}$ for in the following way:

- For every $w \in L$, construct $ind(w)$ which accepts $\{w\}$, for every $w' \in L'$; analogously construct $ind'(w')$ accepting $\{w'\} \cdot \Sigma^*$.

- Construct the union over all automata $ind(w)$ and $ind'(w')$ constructed before.

Formally, $\mathcal{T}$ can be denoted as $(\bigcup_{w\in L} ind(w)) \cup (\bigcup_{w'\in L'} ind'(w'))$. Taking advantage of Lemma 4.18 and of Lemma 4.20 it is not difficult to see that $\mathcal{T}$ accepts $L \cup L'\cdot\Sigma^*$ in accordance with the proposition. We now show that $\mathcal{T}$ meets properties (1) and (2).

▶ 1. In Lemma 4.18 we have seen than for every word $w \in \Sigma^*$ the size $\|ind(w)\|$ and $\|ind'(w)\|$ of the induced automata are in $\mathcal{O}(|w|)$. From Lemma 4.21 we furthermore know that for every treelike automata $\mathcal{T}_1$ and $\mathcal{T}_2$, the size $\|\mathcal{T}_1 \cup \mathcal{T}_2\|$ of the unified automata is less than the sum $\|\mathcal{T}_1\| + \|\mathcal{T}_2\|$ of the original sizes. With these results we can infer the following result for the size of $\mathcal{T}$:

$|\mathcal{T}|$ by definition equals $\|(\bigcup_{w\in L} ind(w)) \cup (\bigcup_{w'\in L'} ind'(w'))\|$. Because of the properties of the union, this is in $\mathcal{O}((\sum_{w\in L} \|ind(w)\|) + (\sum_{w'\in L'} \|ind'(w')\|))$ which is limited by $\mathcal{O}((\sum_{w\in L} |w|) + (\sum_{w'\in L'} |w'|))$ according to the properties of ind. The definition of the size of formal languages implies that this is equivalent to $\mathcal{O}(\|L\| + \|L'\|)$.

▶ 2. For the time complexity, an argument similar to (1) can be devised. We know from Lemma 4.18 that ind and ind' require only linear time in the size of the input.

Unifying treelike automata can also be done in linear time, as shown in Lemma 4.22. Now it is very important to take into account that after each unification the size of the resulting automaton does not exceed the sum of the sizes of the input automata. This guarantees that when repeatedly unifying automata the size of the arguments is always the sum of all automata so far unified. With this we come to the following conclusion.

When executed naively, the union over all automata of the form $ind(w)$ and $ind'(w)$ costs quadratic time in the size of L and L', as can be illustrated easily: unifying $ind(w_1)$ and $ind(w_2)$ costs linear time in $|w_1| + |w_2|$. But then unifying the resulting automaton with $ind(w_3)$ additionally requires linear time in $|w_1| + |w_2| + |w_3|$, so that for the final automaton the costs are $2 \cdot (|w_1| + |w_2|) + |w_3|$. This intuition implies a quadratic result in $\|L\| + \|L'\|$ for the overall time necessary to construct $\mathcal{T}$. It should be noted that this result is sufficient for the argument in the following sections. We will in fact require no more than polynomial time complexity.

However, a more efficient strategy can be found. Instead of unifying two automata in every successive step, the union over all automata can be computed simultaneously. This strategy avoids re-reading the resulting automaton in every step and can thus be realized in linear time in the sum $\|L\| + \|L'\|$ of the sizes of the input languages. □

Observe that the reverse task, i.e. reading off the language accepted by a given treelike automaton, can be solved easily in linear time in the size of the automaton. We only need to perform a depth-first traversal of the automaton and memorize the word read on the path from the root-node to the current node. Whenever visiting a node marked *Fin*, the current word is added to a first language L; whenever visiting *Cyc*, the current word is added to a second language L'. It can be shown that then the automaton examined in that way accepts the language $L \cup L'\cdot\Sigma^*$.
The complexity of standard automata-theoretic problems for treelike automata has not yet been considered. For the purpose of verifying solvability equations, we need two of them at most: the emptyness- and the word-problem. Both can be solved easily for treelike automata, as we will see next.

Lemma 4.25 Decision problems
Let $\mathcal{T} \in \mathit{Treelike}(\Sigma)$ be a treelike automata and let $w \in \Sigma^*$ be a word. Then the following problems are solvable in linear time in the size $\|\mathcal{T}\|$ of the automaton and in the size of w:

1. $L(\mathcal{T}) =^? \emptyset$, i.e. the $\emptyset$-problem

2. $w \in^? L(\mathcal{B})$, i.e. the word-problem

Proof.
▶ 1. As seen in Lemma 4.23, the language accepted by $\mathcal{T}$ is empty if and only if the normalized automaton $simp(trim(\mathcal{T}))$ equals $\underline{Nil}$.

Computing *simp* and *trim* does not increase the size of an automaton. These operations also take only linear time in the size of the argument. We have seen this in Lemma 4.21 and Lemma 4.22. Furthermore, the test of equality to $\underline{Nil}$ costs only constant time. We therefore end up with linear time complexity for the $\emptyset$-problem.

▶ 2. According to the definition of $L(\mathcal{T})$, w is accepted by $\mathcal{T}$ iff $\hat{\delta}(\mathcal{T}, w) \in Acc$. By definition of the left quotient, this is equivalent to $w^{-1}(\mathcal{T}) \in Acc$. According to Lemma 4.22, computing the left quotient costs only linear time in $\|\mathcal{T}\|$ and $|w|$. The left quotient of $\mathcal{T}$ of course is not greater in size than $\mathcal{T}$. Finally, testing for $w^{-1}(\mathcal{T}) \in^? Acc$ takes only constant time, since the left quotient either has to be $\underline{Cyc}$ or it has to be labelled $\underline{Fin}$. Thus, the word problem is of linear time complexity as well. □

There are two operations occurring in our solvability equations which have not yet been defined for treelike automata—concatenation and the right quotient. The right quotient occurs in the equations referring to $\leq$-number restrictions for $\mathcal{ALN}$. However, as can be seen in Definition 4.6, it is applied only to sets of excluding words and it is applied only for single atomic roles. We will see later on that in this case the right quotient can be replaced by a simple expression requiring only known operations.

Nevertheless, treelike automata cannot be concatenated efficiently. When just linking two treelike automata via ε-transitions, the deterministic behaviour as well as the tree structure might get lost. Merely copying the second automaton to every finite state would also violate the tree structure if the first automaton contained inner nodes marked $\underline{Fin}$. In addition copying would be inefficient for successive concatenations (which do not occur in the equations, though).

To solve the problem of concatenation, we have to resort to general nondeterministic finite automata. These, however, cannot be complemented efficiently. Observe that no solvability equation requires concatenation on both sides. This suggests the following strategy: We can represent the left-hand side of an equation by a treelike automaton and the right-hand side—requiring concatenation—by an *NFA*. We can then still compute the complement treelike automaton and intersect it with the *NFA*. This strategy is successful if two conditions hold. Firstly, the intersection of two *NFA* must be efficient. Secondly, the $\emptyset$-problem for *NFA* must be decidable in polynomial time. We know from Lemma 2.8 that these conditions can be met.

In the next section, we will give strategies to decide solvability in $\mathcal{FL}_\bot$, $\mathcal{FL}_\neg$, and $\mathcal{ALN}$. Finally, we shall summarize the results on matching in the last section of this chapter. Matching problems in $\mathcal{FL}_0$ will not be examined in detail, because deciding solvability and computing least solutions there is comparatively simple. When summarizing results, however, we will briefly address this case.

4.3 Deciding solvability

We are now ready to actually decide the solvability of the solvability equations in $\mathcal{FL}_\bot$. Thus, we insert the languages provided in Lemma 4.2 into the equations of Definition 4.1. The resulting equations are then tested for equality using a strategy introduced in the next lemma. Now is the time for deploying the capabilities of treelike automata we have

taken so much care to specify in the last section. The possibility to efficiently compute the complement of a treelike automaton will prove especially useful.

Lemma 4.26 Testing solvability in $\mathcal{FL}_\bot$
Let $C \equiv^? D$ be an $\mathcal{FL}_\bot$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Then

1. Solvability of equation $(\bot)$ as introduced in Definition 4.1 can be decided in polynomial time in the size of the equation.
2. The same holds for equation (A) for every $A \in \mathcal{C}$.

Proof.
▶ 1. According to Definition 4.1 and Lemma 4.2, we have to decide if

$$U_\bot \cdot \Sigma^* = V_\bot \cdot \Sigma^* \cup \bigcup_{j=1}^{\ell} \underbrace{W_j \cdot \bigcap_{w \in W_j} w^{-1}(U_\bot \cdot \Sigma^*)}_{=:L'_j} . \qquad (\bot')$$

For every j it can be shown that L'_j is a subset of $U_\bot \cdot \Sigma^*$ and is Σ^*-closed. For the above equation to hold the following conditions are therefore sufficient:

- $V_\bot \subseteq U_\bot \cdot \Sigma^*$
- For all $u \in U_\bot$: $u \in V_\bot \cdot \Sigma^*$ or there exists a j with $u \in L'_j$.

▷ Testing $V_\bot \subseteq U_\bot \cdot \Sigma^*$: The idea is to construct a nondeterministic finite automaton $\mathcal{B}_0$ accepting the language $V_\bot \cap \overline{U_\bot \cdot \Sigma^*}$. The $\emptyset$-problem is then decided for $\mathcal{B}_0$.

As shown in Lemma 4.24, we can construct a treelike automaton $\mathcal{T}$ for $U_\bot \cdot \Sigma^*$ which is of polynomial size in $\|U_\bot\|$ and which can be obtained in polynomial time. According to Lemmata 4.21 and 4.22, it furthermore takes only polynomial time in the size of $\mathcal{A}$ to construct the complement automaton $\overline{\mathcal{T}}$. Especially, the size of $\overline{\mathcal{T}}$ does not exceed that of $\mathcal{T}$. We can similarly construct a treelike automaton $\mathcal{S}$ for $V_\bot$. As the intersection of two treelike automata takes only polynomial time (Lemma 4.22), we can easily produce an intersection automaton $\mathcal{I} := \overline{\mathcal{T}} \cap \mathcal{S}$. Lemma 4.21 again guarantees that $\mathcal{I}$ is of polynomial size in the sizes of $\mathcal{T}$ and $\mathcal{S}$. Finally, deciding the $\emptyset$-problem for treelike automata takes only polynomial time in the size of the automata. This was shown in Lemma 4.25 when introducing general nondeterministic automata. Altogether, we can decide the first condition in polynomial time, since all the role languages involved are limited in size by the input matching problem.

▷ Testing the second condition: At first, observe that every L'_j is of the form $L \cdot \Sigma^*$, which allows us to except the Σ^*-closure of $U_\bot$ when selecting elements u for our test. Since $U_\bot$ is part of the input matching problem, it is no problem to examine every $u \in U_\bot$ as long as each requires only polynomial time. Thus, consider one such u. Testing whether $u \in V_\bot \cdot \Sigma^*$ can be realized similar to the first condition. We can compute an appropriate treelike automaton to represent $V_\bot \cdot \Sigma^*$. As seen in Lemma 4.25, solving the word problem then costs only polynomial time. Thus, we still have to test if u is an element of $W_j \cdot w^{-1}(U_\bot \cdot \Sigma^*)$ for every j.

For a given j we can test this as follows. W_j is part of the input and thus of polynomial size. Consequently, we can construct a treelike automaton for $U_\bot \cdot \Sigma^*$ and then compute the left quotient for every $w \in W_j$ in polynomial time. We can then establish the intersection automaton over all automata computed that way. The properties of the operations

for treelike automata guarantee that this takes only polynomial time in the size of the matching problem and results in a treelike automaton $\mathcal{T}_j$ of polynomial size.

Furthermore, we can construct a treelike automaton $\mathcal{S}_j$ accepting W_j. The concatenation, however, is not available for treelike automata. We can nevertheless establish a nondeterministic finite automaton representing L'_j by linking $\mathcal{S}_j$ and $\mathcal{T}_j$ in the way proposed in Kleene's theorem for the concatenation automaton: Add ε-transitions from every accepting state of $\mathcal{S}_j$ to the initial state of $\mathcal{T}_j$. This operation obviously does not increase the size of the resulting automaton severely. It results in an *NFA* $\mathcal{B}_j$ accepting L'_j which is polynomial in the size of the matching problem. The treelike property, however, is lost over that operation. This implies that we cannot compute the complement of $\mathcal{B}_j$ efficiently. Fortunately, we only have to solve the word problem for every j, which can be decided in polynomial time. This is a result of Lemma 2.8.

Putting the above arguments together, we can decide in polynomial time in the size of the input matching problem whether equation $(\bot')$ is valid or not.

▶ 2. Equation (A) for every $A \in \mathcal{C}$ holds if and only if:

$$\begin{aligned} U_A \cup U_\bot{\cdot}\Sigma^* &= V_A \cup U_\bot{\cdot}\Sigma^* \;\cup\; \bigcup_{j=1}^{\ell} W_j{\cdot} \bigcap_{w\in W_j} w^{-1}(U_A \cup U_\bot{\cdot}\Sigma^*) \qquad (A') \\ &= V_A \cup U_\bot{\cdot}\Sigma^* \cup \\ &\quad \underbrace{\bigcup_{j=1}^{\ell} W_j{\cdot} \bigcap_{w\in W_j} w^{-1}(U_A)}_{=:L_j} \;\cup\; \underbrace{\bigcup_{j=1}^{\ell} W_j{\cdot} \bigcap_{w\in W_j} w^{-1}(U_\bot{\cdot}\Sigma^*)}_{=:L'_j} \end{aligned}$$

For every j it holds that: $L_j \subseteq U_A$ und $L'_j \subseteq U_\bot{\cdot}\Sigma^*$. For the validity of equation (A') the following two conditions are therefore sufficient:

- $V_A \subseteq U_A \cup U_\bot{\cdot}\Sigma^*$
- For all $u \in U_A$: $u \in V_A \cup U_\bot{\cdot}\Sigma^*$ or there exists a j mit $u \in L_j$ or $u \in L'_j$.

The first condition is treated similarly to (1). We can construct in polynomial time a treelike automaton of polynomial size representing $V_A \cap \overline{U_A \cup U_\bot{\cdot}\Sigma^*}$. For this automaton, again the $\emptyset$-problem can be decided in polynomial time

For the second condition the strategy is almost identical to the one introduced in (1). Especially, testing whether $u \in L_j$ or $u \in L'_j$ requires exactly the same steps as seen above. □

We shall see in the following lemma that the above scheme can easily be generalized for $\mathcal{FL}_\neg$. Here, additional equations for negated atomic concept have to be included in the test. Their structure, however, is identical to the respective non-negated version except for two new aspects. The occurrence of role languages of the form $\widehat{U}_\bot$ instead of $U_\bot$, and the usage of the function *Int* in equation $(\bot)$. We will see that both problems can be solved without altering the overall decision strategy proposed in the last lemma.

Lemma 4.27 Testing solvability in $\mathcal{FL}_\neg$
Let $C \equiv^? D$ be an $\mathcal{FL}_\neg$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Then

1. Solvability of equation $(\bot)$ as introduced in Definition 4.3 can be decided in polynomial time in the size of the equation.
2. The same holds for equations (A) and $(\neg A)$ for every $A \in \mathcal{C}$.

Proof.
▶ 1. Due to Definition 4.3 and Lemma 4.4, we obtain the following equation for the test if equation ($\bot$) is solvable:

$$\widehat{U}_\bot\cdot\Sigma^* = V_\bot\cdot\Sigma^* \cup \underbrace{\bigcup_{j=1}^{\ell} W_j\cdot \bigcap_{w\in W_j} w^{-1}(U_\bot\cdot\Sigma^*)\cdot\Sigma^*}_{=:L_j} \cup \underbrace{\bigcup_{A\in\mathcal{C}} Int(A,\neg A)\cdot\Sigma^*}_{=:L'_A} \qquad (\bot')$$

where inserting the languages specified in Lemma 4.4 into the definition of *Int* yields:

$$\begin{aligned} Int(A,\neg A) = (V_A \cup \bigcup_{j=1}^{\ell} W_j\cdot \bigcap_{w\in W_j} w^{-1}(U_A) \cup \bigcup_{j=1}^{\ell} W_j\cdot \bigcap_{w\in W_j} w^{-1}(\widehat{U}_\bot\cdot\Sigma^*)) \\ \cap\ (V_{\neg A} \cup \bigcup_{j=1}^{\ell} W_j\cdot \bigcap_{w\in W_j} w^{-1}(U_{\neg A}) \cup \bigcup_{j=1}^{\ell} W_j\cdot \bigcap_{w\in W_j} w^{-1}(\widehat{U}_\bot\cdot\Sigma^*)) \end{aligned}$$

Combining the above results, we end up with an equation considerably more complex than that discussed for $\mathcal{FL}_\bot$ in Lemma 4.26. Nevertheless, we can follow a similar scheme to verify the validity of equation ($\bot'$) for $\mathcal{FL}_\neg$. We prove the inclusion from left to right ($\subseteq$) by solving the word problem for an appropriate nondeterministic finite automaton and the inclusion from right to left ($\supseteq$) by deciding the $\emptyset$-problem for the intersection of a treelike and a nondeterministic finite automaton.

▷ ($\subseteq$): Like for $\mathcal{FL}_\bot$, the entire right side of ($\bot'$) is Σ^*-closed. It is therefore again sufficient to test if every word $w \in \widehat{U}_\bot$ can be found in $V_\bot\cdot\Sigma^*$ or in the remaining expression on the right side. Firstly, $V_\bot\cdot\Sigma^*$ can be represented by a treelike automaton. Secondly, following the strategy of Lemma 4.26, we can construct nondeterministic finite automata for every L_j. Thus, we only have to show that there are appropriate automata for deciding the word problem for L'_A. In the above equation for $Int(A,\neg A)$, two expressions of the form $V_A \cup \bigcup W_j\cdot\bigcap_{w\in W} w^{-1}(U_A\cup\widehat{U}_0\cdot\Sigma^*)$ are intersected. Following the construction for the solvability equations (A) in $\mathcal{FL}_\bot$, these expressions can be represented by nondeterministic finite automata. The intersection automaton of these expressions is polynomial in the size of the original automata, as can be seen in Definition 2.5. Thus, we have constructed an *NFA* representing $Int(A,\neg A)$. It takes only linear time compute the Σ^*-closure of that automaton—we just have to add edges from every accepting state pointing to themselves. Consequently, we can provide a polynomially large *NFA* for the representation of L'_A for every $A \in \mathcal{C}$.

▷ ($\supseteq$): We have already seen that we can represent $\widehat{U}_\bot\cdot\Sigma^*$ by a treelike automaton. The idea now is to represent the entire right-hand side of equation ($\bot'$) except the union of L_j by a nondeterministic automaton $\mathcal{B}$. The complement of the treelike automaton for $\widehat{U}_\bot\cdot\Sigma^*$ can then be intersected with $\mathcal{B}$. Testing the result of this for emptyness is equivalent to the inclusion we want to decide.

We do not need to include the union of L_j into the construction of $\mathcal{B}$, since obviously every L_j is already a subset of $U_\bot\cdot\Sigma^*$, which is a subset of the right-hand side of the equation. We know from the previous part, that $V_\bot\cdot\Sigma^*$ easily can be represented by a treelike automaton as well as every L'_A can be represented by a polynomially large *NFA*. The only step missing now is to compute the union of all these automata. We have already seen that unifying nondeterministic finite automata only costs a constant amount of additional space and can be done in linear time. Therefore, we require only linear time to obtain the desired automaton. We can now compute the complement automaton for the

left-hand side of the equation and test the intersection automaton for emptyness. Since the intersection automaton is polynomial in the size of the original automata according to Definition 2.5 and since the $\emptyset$-problem can be decided in polynomial time in the size of the automaton, we end up with polynomial time complexity for the decision.

▶ 2. For every $A \in \mathcal{C}$, inserting the appropriate languages into solvability equation (A) yields the following equation:

$$U_A \cup \widehat{U}_\perp \cdot \Sigma^* = V_A \cup \widehat{U}_\perp \cdot \Sigma^* \cup \bigcup_{j=1}^{\ell} W_j \cdot \bigcap_{w \in W_j} w^{-1}(U_A \cup \widehat{U}_\perp \cdot \Sigma^*) \qquad (A')$$

We have already mentioned, that $\widehat{U}_\perp$ can be represented by a treelike automaton. Therefore, the above equation is merely a syntactic variant of the analogous equation for $\mathcal{FL}_\perp$. In consequence, we can decide equality with exactly the same strategy as introduced for equation (A') in Lemma 4.26. Due to the similarity of the equations, this argument also applies to equation $(\neg A')$ for every $A \in \mathcal{C}$. □

Most of the complexity of the solvability equations for $\mathcal{ALN}$ is hidden in the construct of excluded words, occurring as E_C and E_D. Thanks to the results presented in [BKBM99] and [Küs98], we need not resolve their structure in detail. Instead, we can rely on the fact that there exists an algorithm to compute the set of excluded words of a given concept description in polynomial time. Nevertheless, we must ensure one condition: inserting the languages proposed in Definition 4.7 into the right-hand side of equation $(\perp)$ may not blow up their size exponentially. On the other hand, once given a polynomial representation of E_C, the argument for the other equations is very similar to the approaches seen before for $\mathcal{FL}_\perp$ and $\mathcal{FL}_\neg$.

Lemma 4.28 Testing solvability in $\mathcal{ALN}$
Let $C \equiv^? D$ be an $\mathcal{ALN}$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Then

1. Solvability of equation $(\perp)$ as introduced in Definition 4.6 can be decided in polynomial time in the size of the equation.

2. The same holds for equations (A) and $(\neg A)$ for every $A \in \mathcal{C}$ as well as for equations $(\leq nR)$ and $(\geq nR)$ for every $(\leq nR) \in \mathcal{N}_\leq$ and $(\geq nR) \in \mathcal{N}_\geq$.

Proof.
▶ 1. According to Lemma 4.7, we have to decide if

$$E_C = E_D(X_\perp, X_\mathcal{C}, X_\neg, X_\geq, X_\leq)$$

It is stated in [BKBM99] that it takes only polynomial time in the size of C to compute a finite set U_{E_C} with $E_C = U_{E_C} \cdot \Sigma^*$. Therefore, it can be shown that we need only polynomial time to compute the solution languages introduced in Lemma 4.7. It can be shown further that these languages are only polynomially large in the size of the original matching problem. Therefore, inserting these languages into D yields a an $\mathcal{ALN}$-concept description of polynomial size in the size of D. According to [BKBM99], we can then in polynomial time compute a finite language U_{E_D} such that $E_C = U_{E_C} \cdot \Sigma^*$. Thus, we can construct treelike automata for the representation of both sides of the equation, which are polynomial in the size of the original matching problem. Equivalence therefore can be decided in two steps by testing mutual inclusion: Firstly, compute the complement of one automaton and then test the intersection with the other for emptyness. Secondly, perform the same test vice-versa with the automata exchanged.

▶ 2. Inserting the languages of Lemma 4.7 in the remaining solvability equations yields equations of the following type:

$$U_A \cup E_C = V_A \cup E_C \cup \bigcup_{j=1}^{\ell} W_j \cdot \bigcap_{w \in W_j} w^{-1}(U_{\neg A} \cup E_C) \qquad (A')$$

$$\bigcup_{m \geq n} U_{(\geq mR)} \cup E_C = V_{(\geq mR)} \cup E_C \cup \bigcup_{j=1}^{\ell} W_j \cdot \bigcap_{w \in W_j} w^{-1}(\bigcup_{m \geq n} U_{(\geq nR)} \cup E_C)$$

We have mentioned under (1) that for the representation of E_C and E_D polynomially large treelike automata can be constructed in polynomial time. Therefore, the first type of equations can be verified with the strategy introduced for the equations (A') for $\mathcal{FL}_\bot$ in Lemma 4.26. Taking into account that we can also compute treelike automata for the representation of the union $\bigcup_{m \geq n} U_{(\geq mR)}$ and $\bigcup_{m \geq n} U_{(\geq nR)} \cup E_C$, the same scheme can be employed for the equations referring to number restrictions. □

At this point, we know that matching problems modulo equivalence can be decided in polynomial time for $\mathcal{ALN}$ and its sublanguages. We still have to discuss how to compute the actual solution to a solvable matching problem. The next section gives a brief summary on this subject.

4.4 General result

Apart from testing solvability, [BKBM99] also proposes solutions to be assigned to the variables occurring in a matching problem and proves their correctness in detail. Using our results on the complexity of operations on treelike automata, we will now furthermore confirm that computing the actual solution to a solvable matching problem takes only polynomial time. Additionally, we recall three other properties of the solution strategy. Firstly, it is shown that it produces least solutions in regard to ($\sqsubseteq$); secondly, it introduces no new atomic concepts or number restrictions; and thirdly, it can handle systems of matching problems as well.

Lemma 4.29 Solving matching problems
Let $\mathcal{L}$ be a logic in $\{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let P be an $\mathcal{L}$-matching problem modulo equivalence as introduced in Definition 3.14. Then there exists an algorithm $match_\mathcal{L}$ with the following properties:

1. $match_\mathcal{L}(P)$ decides in polynomial time, whether the input matching problem P has a solution or not. If P is solvable, then $match_\mathcal{L}(P)$ in polynomial time in the size of P computes a solution σ which is minimal in regard to ($\sqsubseteq$).

2. $match_\mathcal{L}$ does not introduce atomic concepts or number restrictions which do not occur in the input matching problem P.

3. $match_\mathcal{L}$ also accepts a system of matching problems as introduced in Definition 3.14.

Proof.
▶ 1. In the previous sections we have shown that there exist strategies for deciding solvability of a given matching problem in polynomial time in the size of the problem. We have seen that such strategies can be found for $\mathcal{ALN}$ as well as for its three sublanguages considered here. We still have to make sure that computing the actual solution to a solvable matching problem also requires only polynomial time, which can be readlily inferred

utilizing the results obtained so far. [BKBM99] provides us with strategies to specify appropriate solution languages. Taking advantage of our results concerning the complexity properties of treelike automata we will show that these languages can be computed in polynomial time. We first give the prove for $\mathcal{ALN}$ and then consider its sublanguages separately.

▷ Solutions in $\mathcal{ALN}$: To show this for $\mathcal{ALN}$-matching problems, we only need to combine results we have already obtained. In [BKBM99] it is shown that the languages $L_{\cdot,\cdot}$ used for the solvability test in Lemma 4.7 in fact are least solutions to the matching problem. Therefore, a solution σ with the desired properties can be defined by assigning

$$X_j \longmapsto \forall L_{j,\perp}.\perp \sqcap \bigsqcap_{A\in\mathcal{C}} \forall L_{j,A}.A \sqcap \bigsqcap_{A\in\mathcal{C}} \forall L_{j,\neg A}.\neg A \\ \sqcap \bigsqcap_{(\leq nR)\in\mathcal{N}_{\leq}} \forall L_{j,(\leq nR)}.(\leq nR) \sqcap \bigsqcap_{(\geq nR)\in\mathcal{N}_{\geq}} \forall L_{j,(\geq nR)}.(\geq nR)$$

for every $j \in \{1,\ldots,\ell\}$. It can be shown that the assigned concept descriptions are of polynomial size in the size of the original matching problem. Since every role language of the form $L_{\cdot,\cdot}$ can be represented by a treelike automaton, it takes only polynomial time to read off the languages represented by these automata, i.e. to actually return the computed result.

▷ Solutions in $\mathcal{FL}_{\perp}$ and $\mathcal{FL}_{\neg}$: For these sublanguages of $\mathcal{ALN}$, we must first restrict the languages used in the solvability test to finite ones. The rest of the argument then is identical to that for $\mathcal{ALN}$. For $\mathcal{FL}_{\perp}$ and $\mathcal{FL}_{\neg}$, [BKBM99] again provides us with the necessary results: Finite solution languages $L_{j,A}$ can be obtained in the following way. Since $\widehat{L}_{j,\perp}$ can be represented by a treelike automaton for every j, we read off a finite language $L_{j,\perp}$ with $L_{j,\perp}\cdot\Sigma^* = \widehat{L}_{j,\perp}$. Analogous to the languages defined for $\mathcal{ALN}$ in Lemma 4.7 we now define languages $L_{j,A}$ by subtracting $\widehat{L}_{j,\perp}$ from $\widehat{L}_{j,A}$. We can then assign to the variable X_j the conjunction

$$X_j \longmapsto \forall L_{j,\perp}.\perp \sqcap \bigsqcap_{A\in\mathcal{C}} \forall L_{j,A}.A \sqcap \bigsqcap_{A\in\mathcal{C}} \forall L_{j,\neg A}.\neg A$$

for every $j \in \{1,\ldots,\ell\}$. Again, we yield a solution of polynomial size in polynomial time. The argument for $\mathcal{FL}_{\perp}$ is identical except for negated atomic concept missing in the concept descriptions finally assigned.

▷ Solutions in $\mathcal{FL}_0$: Two arbitrary $\mathcal{FL}_0$-concept descriptions are equivalent if and only if their $\mathcal{FL}_0$-normal forms agree on all role languages involved. Therefore, infinite sets are not necessary at any step when solving matching problems. It can be shown that the solvability equation and solution languages for $\mathcal{FL}_0$ are equivalent to those for $\mathcal{FL}_{\perp}$ after removing any constructs relating to the bottom-concept or its role languages. The task of deciding solvability and computing solutions to a given matching problem then quite apparently turns out to be of polynomial complexity.

▶ 2. It is shown in [BKBM99], that the solution specified above already has the desired property. Especially, this implies that the solution of a matching problem can be represented with the same set of role languages as the matching problem.

▶ 3. In Lemma 3.15, we have already seen that systems of matching equations can be represented by a single matching problem modulo subsumption which is polynomial in the size of the original system. Thus, with the results from (1) the proposition follows immediately. □

Our examination of matching problems modulo equivalence without side conditions is complete. We can decide and solve matching problems without side conditions in polynomial time. Furthermore, we can find minimal solutions without introducing new atomic concepts or number restrictions and we can admit systems of matching problems as input. The results obtained here will be of eminent importance for Chapter 5, where a solution strategy for matching problems with side conditions is introduced.

CHAPTER 5

Eliminating Side Conditions

In the previous chapter, an efficient solution strategy for matching problems without side conditions has been proposed. We now approach matching problems with acyclic non-strict side conditions. The idea is to reduce a matching problem with side conditions to an equivalent one without by augmenting the original matching equation by additional constraints. A strategy for this is discussed in [BKBM99]. However, it is also demonstrated that this might result in exponentially large matching equations. In the first section, we will briefly introduce the relevant reduction strategy. Moreover, we will show that an intuitive strategy to represent the resulting matching problem more compactly fails to avoid the exponential blow-up of role languages. In the second section, these problems are overcome by employing nondeterministic finite automata for the representation of role languages. It will be shown then that the matching algorithm introduced in the previous chapter can be modified to accept role languages represented by automata. The solution proposed in this chapter, however, is limited to acyclic side-conditions.

5.1 Reducing matching problems

The idea of reducing a matching problem with side conditions to an equivalent one without side conditions is introduced in [BKBM99]. The following substitution is defined to facilitate this reduction. In spite of syntactic similarity we call it "generalized substitution" because substitutions in Definition 3.13 have been defined to map concept patterns onto concept descriptions and not onto concept patterns again.

Definition 5.1 Generalized substitutions
Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be an $\mathcal{FL}_\bot$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. The generalized substitution θ is inductively defined as follows:

$$\theta(X_1) := Y_1 \sqcap E_1$$
$$\theta(X_j) := Y_j \sqcap \theta(E_j) \qquad \square$$

It is shown in [BKBM99] that a matching problem with acyclic side conditions of the form $(C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ is equivalent to $(C \equiv^? \theta(D))$, which is free of side conditions. Equivalent in this context means having the same solution. It is also shown that this modification can result in exponentially large role languages in the modified matching problem. The remedy suggested in this context is a "compact representation"

for role languages, which avoids making concatenations explicit. The following definition extends this idea to the notion of product form languages.

Definition 5.2 Product form languages
For all role languages U and V over Σ, *product form languages* are defined as follows:

- Every role language U is a product form language.
- If U and V are product form languages, so are $U \cdot V$ and $\{U, V\}$.

The semantics of product form languages is inductively defined over their structure. For product form languages U and V, the expression $U \cdot V$ represents the concatenation of the languages represented by U and V. Similarly, $\{U, V\}$ yields the union of the respective languages. A formal definition of the semantics of product form languages is omitted. □

The above definition does not only allow for a product representation, but also admits nested product forms by including the case $\{U, V\}$. The following example shows that product form languages in fact yield a more compact representation for formal languages.

Example 5.3 Product representation I
The language $\{RR, RS, SR, SS\}$ can be represented in product form by $\{R, S\} \cdot \{R, S\}$. However, the concatenation may not only occur at the outermost level. For instance, the language $\{RR, RSR, RSS, S\}$ can be represented in by $\{\{R\} \cdot \{R, \{S\} \cdot \{R, S\}\}, \{S\}\}$. □

However, problems may arise when employing product form languages for the elimination of side conditions. Consider the next example, where role languages produced by the generalized substitution θ are represented by product form languages.

Example 5.4 Product representation II
Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be an $\mathcal{FL}_\bot$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. For P, we specify the following side conditions:

$$
\begin{aligned}
E_1 &: X_1 \sqsubseteq^? A \\
E_2 &: X_2 \sqsubseteq^? \forall\{R, S\}.X_1 \\
E_j &: X_j \sqsubseteq^? \forall\{R\}.X_{j-1} \sqcap \forall\{Q, S\}.X_{j-2}
\end{aligned}
$$

where $j \in \{3, \ldots, \ell\}$. We have seen that the side conditions can be eliminated by replacing D with $\theta(D)$. The role languages of the resulting concept pattern are represented by product form languages. In the following, we restrict our attention to role languages referring to the atomic concept A. For the first four variables, we obtain according to the definition of θ:

$$
\begin{aligned}
\theta(X_1) &\equiv \ldots \sqcap A \\
\theta(X_2) &\equiv \ldots \sqcap \forall\{R, S\}.A \\
\theta(X_3) &\equiv \ldots \sqcap \forall\{\{R\} \cdot \{R, S\}, \{Q, S\}\}.A \\
\theta(X_4) &\equiv \ldots \sqcap \forall\{\{R\} \cdot \underbrace{\{\{R\} \cdot \{R, S\}, \{Q, S\}\}}_{L_1}, \{Q, S\} \cdot \underbrace{\{R, S\}}_{L_2}\}.A
\end{aligned}
$$

It is easy to see that the product form representation of the languages is indeed more compact. For instance, the resulting role language referring to the atomic concept A in $\theta(X_4)$ in explicit form is:

$$\theta(X_4) \equiv \ldots \sqcap \forall\{RRR, RRS, RQ, RS, QR, QS, SR, SS\}.A$$

However, product form role languages do not prevent the above concept patterns from growing exponentially large, as we will now see. In the above example, the sublanguage L_2 is identical to the respective result for X_2 and similarly, L_1 is identical to the result for X_3. This expansion carries on for all $j \in \{1, \ldots, \ell\}$. Thus, the size of the product form language referring to A for every concept variable X_j is greater than the sum of the sizes of the previous two results. This implies that the size of the product form representation of $\theta(X_j)$ increases faster than the Fibonacci Sequence which constitutes an exponential growth. □

It might be possible for every $\theta(X_j)$ to find another representation on the basis of product form languages, which is more compact than the one immediately produced by θ. The simplifications necessary for this, however, are greatly dependent on the individual structure of the side conditions and cannot be realized in an intuitive way. It is not clear whether an appropriate simplification can always be achieved in polynomial time.

5.2 Automata and acyclic side conditions

We have seen that it is difficult to find appropriate compact representations avoiding an exponential blow-up when eliminating acyclic side conditions. In this section, we will employ finite automata for the representation of role languages. To this end we study the structure of the role languages produced by the generalized substitution θ. The result will provide us with a strategy to compute appropriate nondeterministic finite automata. It is essential in this context to find a strategy which avoids copying identical structures when synthesizing an automaton. We have seen in Example 5.4 how language-related representation techniques are flawed by structure copying. Automata-theoretic approaches would be affected by the same problem.
Here, we therefore share or re-use sub-automata appearing at several positions in the construction, i.e. instead of using several instances of a sub-automaton only one instance is introduced, linked with all necessary states by appropriate edges. Since the side conditions are acyclic, we can use an inductive argument to find an appropriate construction.

Lemma 5.5 Automata and side conditions
Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be an $\mathcal{FL}_\bot$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form with non-strict acyclic side conditions. Then:

1. The role languages occurring in $\theta(X_j)$ can be represented by nondeterministic finite automata, which size is polynomial in the size of P and which can be computed in polynomial time. This holds for every $j \in \{1, \ldots, \ell\}$
2. The same holds for the role languages occurring in $\theta(D)$.

Proof.
▶ 1. It is sufficient to prove the assertion for $\sigma(X_\ell)$. For smaller values of j, remove the side conditions for $\{X_{j+1}, \ldots, X_\ell\}$. Due to the acyclic nature of the side conditions, the following argument can be employed for $\sigma(X_j)$ as well. Proof by induction over ℓ.

▷ $\ell = 1$: Trivial. Due to acyclic side conditions, it holds that: $\theta(X_1) = Y_1 \sqcap \bigsqcap_{A \in \mathcal{C}} \forall V_{1,A}.A$. The size of the role languages $V_{1,A}$ do not exceed the size of the input problem P. Therefore, according to Lemma 2.9 only polynomial time in the size of P is necessary to construct appropriate nondeterministic automata $\mathcal{B}_{1,A} \in \mathit{Treelike}(\Sigma)$ for the representation of $V_{1,A}$.

▷ $\ell > 1$: Due to induction, we may assume that the assertion holds for every $j < \ell$. Thus, for every $j \in \{1, \ldots, \ell - 1\}$, for every $j' \in \{1, \ldots, j - 1\}$ and for every $A \in \mathcal{C}$ there exist

polynomially large automata $\mathcal{B}_{j,A}$, $\mathcal{C}_{j,j'} \in NFA(\Sigma)$ such that $\theta(X_j)$ can be respresented as follows:

$$\theta(X_j) \equiv Y_j \sqcap \bigsqcap_{A \in \mathcal{C}} \forall L(\mathcal{B}_{j,A}).A \sqcap \bigsqcap_{j'=1}^{j-1} \forall L(\mathcal{C}_{j,j'}).Y_{j'}$$

Consider $\theta(X_\ell)$. According to the definition, this equals $Y_\ell \sqcap \theta(E_\ell)$. We may then expand E_ℓ according to Defintion 3.14 and apply the definition of θ. Due to induction, we obtain:

$$\theta(X_\ell) \equiv Y_\ell \sqcap \bigsqcap_{A \in \mathcal{C}} \forall V_{\ell,A}.A \sqcap \bigsqcap_{j=1}^{\ell-1} \forall W_{\ell,j}.\left(Y_j \sqcap \bigsqcap_{A \in \mathcal{C}} \forall L(\mathcal{B}_{j,A}).A \sqcap \bigsqcap_{j'=1}^{j-1} \forall L(\mathcal{C}_{j,j'}).Y_{j'}\right)$$

By sorting the role languages occurring in the above expression by respecting atomic concept or variable, we yield the $\mathcal{FL}_0$-normal of $\sigma(X_j)$:

$$\begin{aligned}\theta(X_\ell) \equiv \bigsqcap_{A \in \mathcal{C}} \forall(\underbrace{\{V_{\ell,A}\} \cup \{W_{\ell,j} \cdot L(\mathcal{B}_{j,A}) | 1 \leq j \leq \ell - 1\}}_{M_A}).A \\ \sqcap \bigsqcap_{j'=1}^{\ell-1} \forall(\underbrace{\{W_{\ell,j'}\} \cup \{W_{\ell,j} \cdot L(\mathcal{C}_{j,j'}) | j'+1 \leq j \leq \ell - 1\}}_{M'_{j'}}).Y_{j'} \\ \sqcap Y_\ell\end{aligned}$$

It has to be shown that for all A and j' there exist automata $\mathcal{B}_{\ell,A}, \mathcal{C}_{\ell,j'} \in NFA(\Sigma)$ of polynomial size in the size of P, such that $L(\mathcal{B}_{\ell,A}) = M_A$ and $L(\mathcal{C}_{\ell,j'}) = M'_{j'}$.

The role languages $V_{\ell,A}$ and $W_{\ell,j}$ do not exceed the size of the input matching problem. We know from Lemma 2.9 that we can construct appropriate nondeterministic finite automata for their representation, which exceed their respective language in size only by a constant. According to the induction hypothesis there exist polynomially large automata $\mathcal{B}_{j,A}$ and $\mathcal{C}_{j,j'}$ for the representation of $L(\mathcal{B}_{j,A})$ and $L(\mathcal{C}_{j,j'})$ respectively. Incidentally, these automata have already been constructed in the previous steps of the induction. For the representation of $W_{\ell,j} \cdot L(\mathcal{B}_{j,A})$ we therefore merely construct an automaton representing $W_{\ell,j}$ and link it to the already existing instance of $\mathcal{B}_{j,A}$ by an appropriate ε-transition. From this an automaton representing M_A is easily obtained. It suffices to introduce a new initial state which non-deterministically branches to the automaton for $V_{\ell,A}$ on the one hand and to that for $W_{\ell,j}$ on the other. An analogous procedure can be used to construct an automaton for the representation of $M'_{j'}$.

Following this strategy, we finally end up with polynomially large automata having the desired properties. As we use nondeterministic finite automata, the above construction takes only polynomial time. Especially the union of several *NFA* can be computed in polynomial time in the sum of the sizes of the original automata and results in an union-automaton of polynomial size in the size of the input matching problem.

▶ 2. For the concept pattern D, it holds due to the definition of θ that:

$$\theta(D) \equiv \bigsqcap_{A \in \mathcal{C}} \forall V_A.A \sqcap \bigsqcap_{j=1}^{l} \forall W_j.\theta(X_j)$$

We have seen in (1) that appropriate automata for the representation of all role languages occurring in $\theta(X_j)$ can be computed in polynomial time for all j. Including the remaining role languages requires a concatenation with the language W_j and—in some cases—a union with V_A. Both operations can be accomplished by the same scheme proposed in the above part. □

Due to this lemma, acyclic non-strict side conditions can be eliminated without an exponential blow-up of the resulting matching problem. The next sections are concerned with the question if the matching algorithms introduced in Lemma 4.29 can be modified to cope with the modifications introduced here.

5.3 Restricting large languages

Before modifying the original matching algorithm, we will prove that there is a strategy to avoid considering all words of the role languages occuring in an input matching problem. Note that the algorithm introduced in Lemma 4.29 in fact relies on this ability. Since the reduction by the generalized substitution θ only affects the right-hand side of the matching problem, it is not necessary to find a strategy for all role languages. We will see that only the intersection of left quotients appearing in the algorithm requires special attention. The following definition introduces some auxiliary notions. The intention behind their introduction will become clear in the next lemma.

Definition 5.6 Auxiliary languages
Let U, W be languages over Σ. Define the prefix closure $pre(U)$ and the auxiliary sets $pre(U)_W$, $\overline{pre}(U)_W$, $post(U)_W$, and $neg(U)_W$ as follows:

$$\begin{aligned} pre(U) &:= \{w \in \Sigma^* | \exists s \in \Sigma^* \colon ws \in U\} \\ pre(U)_W &:= pre(U) \cap W \\ \overline{pre}(U)_W &:= \overline{pre(U)} \cap W \\ post(U)_W &:= U{\cdot}\Sigma^* \cap W \\ neg(U)_W &:= \overline{pre(U) \cup U{\cdot}\Sigma^*} \cap W \end{aligned}$$

□

Observe that U can be expressed in terms of the above languages. It can be shown that $W = pre(U)_W \cup post(U)_W \cup neg(U)_W$, and similarly $W = pre(U)_W \cup \overline{pre}(U)_W$. For example, consider $\Sigma := \{R, S\}$, $U := \{RS\}$, and $W := \{R, RR\}$. Then we have $pre(U)_W = \{R\}$, $post(U)_W = \emptyset$, and $neg(U)_W = \{RR\}$. In the following lemma it is shown that these auxiliary languages can be used to simplify intersections of left quotients:

Lemma 5.7 Properties
Let $U, W \subseteq \Sigma^*$ be languages. Then

1. $\bigcap_{w \in post(U)_W} w^{-1}{\cdot}(U{\cdot}\Sigma^*) = \Sigma^*$
2. If $neg(U)_W \neq \emptyset$, then $\bigcap_{w \in W} w^{-1}{\cdot}(U{\cdot}\Sigma^*) = \emptyset$
3. If $neg(U)_W = \emptyset$, then $\bigcap_{w \in W} w^{-1}{\cdot}(U{\cdot}\Sigma^*) = \bigcap_{w \in pre(U)_W} w^{-1}{\cdot}(U{\cdot}\Sigma^*)$
4. If $\overline{pre}(U)_W \neq \emptyset$, then $\bigcap_{w \in W} w^{-1}{\cdot}(U) = \emptyset$
5. If $\overline{pre}(U)_W = \emptyset$, then $\bigcap_{w \in W} w^{-1}{\cdot}(U) = \bigcap_{w \in pre(U)_W} w^{-1}{\cdot}(U)$

Proof.
▶ 1. Consider an arbitrary $w \in post(U)_W$. We prove that $w^{-1}{\cdot}(U{\cdot}\Sigma^*) = \Sigma^*$. By definition of the left quotient, there exists a word $u \in U$ and a word $v \in \Sigma^*$, with $w = uv \subseteq U{\cdot}\Sigma^*$. Consequently, any continuation of w lies in the same set: $ws \in U{\cdot}\Sigma^*$ for every word $s \in \Sigma^*$. This is equivalent to $w^{-1}{\cdot}(U{\cdot}\Sigma^*) = \Sigma^*$, completing the argument.

▶ 2. It is sufficient to show that there exists a word $w \in W$ such that $w^{-1}{\cdot}(U{\cdot}\Sigma^*)$ is empty. According to the assumption, we may assume a word $w \in neg(U)_W$, which by definition means that $w \in \overline{pre(U) \cup U{\cdot}\Sigma^*} \cap W$. Therefore, $w \in W$, but neither is w an element of $pre(U)$, nor of $U{\cdot}\Sigma^*$. Hence, w is no prefix of a word $u \in U$ and $w \notin U{\cdot}\Sigma^*$. This implies for every word $s \in \Sigma^*$ that $ws \notin U{\cdot}\Sigma^*$. Consequently, the left quotient $w^{-1}{\cdot}(U{\cdot}\Sigma^*)$ is empty.

▶ 3. We have mentioned in Definition 5.6 that W can be expressed as the following union: $pre(U)_W \cup post(U)_W \cup neg(U)_W$. As $neg(U)_W$ is assumed to be empty, we may split up the intersection $\bigcap_{w \in W} w^{-1}{\cdot}(U{\cdot}\Sigma^*)$ into one intersection over all $w \in pre(U)_W$ and another

over all $w \in post(U)_W$. We have seen in (1), that the intersection over all $w \in post(U)_W$ is equal to Σ^*, which implies the assertion.

▶ 4. and 5. The argument for $\overline{pre}(U)_W$ is analogous to the cases (2) and (3) above. It holds that W can be expressed as the union $pre(U)_W \cup \overline{pre}(U)_W$ and for every word $w \in \overline{pre}(U)_W$ it holds that $w^{-1} \cdot (U)$ is empty. □

It is not yet clear why the above assertions yield a desirable modification. This is clarified in the following lemma, when discussing the complexity of the involved languages.

Lemma 5.8 Decidable Problems
Let $U \subseteq \Sigma^*$ be a finite language and let $\mathcal{B} \in NFA(\Sigma)$ be a nondeterministic finite automaton. Denote the accepted language as $L(\mathcal{B}) = W$ Then:

1. $\|pre(U)\| \leq \|U\|^2$
2. $pre(U)_W$ can be computed in polynomial time in $|\mathcal{B}|$ and $\|U\|$
3. $neg(U)_W \stackrel{?}{=} \emptyset$ is decidable in polynomial time in $|\mathcal{B}|$ und $\|U\|$
4. $\overline{pre}(U)_W \stackrel{?}{=} \emptyset$ is decidable in polynomial time in $|\mathcal{B}|$ and $\|U\|$.

Proof.
▶ 1. Every word $u \in U$ has at most $|u|$ different prefixes, all of which are shorter than u. This implies that the size $\|pre(U)\|$ does not exceed $\sum_{u \in U} |u| \cdot |u|$, which is obviously less or equal to $\|U\|^2$. Observe that consequently, $pre(U)_W$ is also quadratic in the size $\|U\|$, since $pre(U)_W$ is a subset of $pre(U)$.

▶ 2. $pre(U)$ can be computed easily from U. For every word $u \in U$, we simply add every prefix of u to the result. This obviously takes only polynomial time. To construct $pre(U)_W$ from $pre(U)$, we now only have to decide the word problem in respect to $\mathcal{B}$. We have seen in Lemma 2.8, that deciding the word problem costs only polynomial time. Due to (1), we know that the word problem only has to be decided for polynomially many words, which completes our argument.

▶ 3. According to Lemma 4.24, it costs only linear time in the size $\|U\|$ to construct a treelike automation $\mathcal{A}_1$, such that $\mathcal{A}_1$ accepts $U \cdot \Sigma^*$ and the size of $\mathcal{A}_1$ is linear in $\|U\|$. We can analogously define an automaton $A_2 \in Treelike(\Sigma)$ for the representation of $pre(U)$.

For treelike automata, the operations union and complement take only linear time and produce a resulting automaton, which in size does not exceed the sum of the sizes of the original automata. Consequently, we can use the operations on treelike automata to define an automaton $\mathcal{A} := \overline{A_1 \cup A_2}$. Obviously, $\mathcal{A}$ accepts the language $\overline{pre(U) \cup U \cdot \Sigma^*}$. The size of $\mathcal{A}$ is quadratic in $\|U\|$.

Next we construct a nondeterministic finite automaton $\mathcal{C}$ as the product automaton of $\mathcal{A}$ and $\mathcal{B}$. Due to the definition of the product automaton, it holds that $\mathcal{C}$ accepts the intersection of $\overline{pre(U) \cup U \cdot \Sigma^*}$ and W, which is equal to $neg(U)_W$. Furthermore, the size of $\mathcal{C}$ is polynomial in the size of $\mathcal{A}$ and $\mathcal{B}$. Finally, the $\emptyset$-problem for $\mathcal{C}$ can be decided in polynomial time, as shown in Lemma 2.8.

▶ 4. The argument for $\overline{pre}(U)_W$ is identical to (3). We merely have to except the automaton $\mathcal{A}_2$ from the scheme proposed in the above case. □

If we can represent the language W by a nondeterministic finite automaton, then $pre(U)_W$ is only of polynomial size. Moreover, the validity of the prerequisites in Lemma 5.7 can be verified in polynomial time. With these preliminaries, we can specify a modified matching algorithm in the next section.

5.4 Automata-theoretic solution

We have seen in the previous section that it takes only polynomial time to transform a matching problem with acyclic side conditions into an equivalent one without side conditions, where the occurring role languages are represented by finite automata. In Chapter 4, algorithms have been proposed to solve ordinary matching problems. These algorithms are now extended to admitting finite automata for the representation of role languages in the input matching problem. At first we discuss an approach for $\mathcal{FL}_\bot$ and then very briefly address the case of $\mathcal{FL}_\neg$. We will not address a strategy for $\mathcal{FL}_0$ explicitly here because for this case, a strategy analogous to that for $\mathcal{FL}_\bot$ can be specified without difficulty.

5.4.1 Result for $\mathcal{FL}_\bot$

In analogy to Lemma 4.26, we again examine testing solvability for $\mathcal{FL}_\bot$ matching problems. Now the role languages of the form V_i and W_j occurring in the input matching problem are assumed to be represented by nondeterministic finite automata. We will find that the general scheme of the solvability test of Lemma 4.26 can still be applied.

Lemma 5.9 Testing solvability in $\mathcal{FL}_\bot$
Let $C \equiv^? D$ be an $\mathcal{FL}_\bot$-matching problem in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. For every $H \in \{\bot\} \cup \mathcal{C}$ and for every $j \in \{1, \ldots, \ell\}$, let $\mathcal{V}_H, \mathcal{W}_j \in NFA(\Sigma)$ be nondeterministic finite automata such that every automaton $\mathcal{V}_H$ accepts the language V_H and every $\mathcal{W}_j$ accepts W_j. Then

1. Equation $(\bot)$ as introduced in Definition 4.1 can be verified in polynomial time in $|\mathcal{C}|$ and the size of all automata $\mathcal{V}_H$ and $\mathcal{W}_j$.
2. The same holds for Equation (A) for every $A \in \mathcal{C}$.

Proof.
▶ 1. Let us first recall the strategy used previously to decide solvability for equation $(\bot)$. In Lemma 4.26, deciding the following conditions proves sufficient: For every $j \in \{1, \ldots, \ell\}$, define $L'_j := W_j \cdot \bigcap_{w \in W_j} w^{-1} \cdot (U_\bot \cdot \Sigma^*)$. Then equation $(\bot)$ is solvable iff

- $V_\bot \subseteq U_\bot \cdot \Sigma^*$
- For all $u \in U_\bot$: $u \in V_\bot \cdot \Sigma^*$ or there exists a j with $u \in L'_j$.

Suppose verifying the above conditions with the former strategy of Lemma 4.26, which problems would occur? The scheme for the first condition still holds, only we already have an *NFA* accepting $V_\bot$ and do not need to construct it. On the contrary, we even positively abandon constructing it anew from $V_\bot$, because the language $V_\bot$ might be exponentially large in $|\mathcal{V}_i|$. Thus, we test $\mathcal{V}_\bot \cap \overline{U_\bot \cdot \Sigma^*} =^? \emptyset$, using the automaton $\mathcal{V}_\bot$ already given for $V_\bot$.

For the second condition, testing $u \in V_\bot \cdot \Sigma^*$ again remains feasible. The automaton $\mathcal{V}_\bot$ can be modified in linar time to accept $V_\bot \cdot \Sigma^*$, we just have to add cycles to every accepting state. This modification does not enlarge the automaton significantly, so that the word problem is still decidable in polynomial time.

The test for $u \in L'_j$, however, must be modified to remain efficient in our new setting. In the former case, we could afford to construct a treelike automaton for the representation of $w^{-1} \cdot (U_\bot \cdot \Sigma^*)$ for every $w \in W_j$. Now the language W_j might be exponentially large, thus ruling out the possibility to consider every word in W_j separately. Lemma 5.7 provides us with a means to avoid this. If $neg(U_\bot)_{W_j}$ is not empty, then L'_j is empty. Moreover, if $neg(U_\bot)_{W_j}$ is empty, then we may restrict the intersection in the definition of L'_j to all words in $pre(U_\bot)_{W_j}$ instead of W_j. We have seen in Lemma 5.8 that emptyness of $neg(U_\bot)_{W_j}$ is decidable in polynomial time. Furthermore, $pre(U_\bot)_{W_j}$ is of polynomial

size in $\|U_\perp\|$ and can be computed in polynomial time in $\|U_\perp\|$ and $|\mathcal{W}_j|$, so that after restricting the intersection to all words in $neg(U_\perp)_{W_j}$, the former strategy for testing $u \in L'_j$ becomes applicable again: for every $u \in U_\perp$, it takes only polynomial time to test if u is an element of $V_\perp \cdot \Sigma^*$ or an element of $W_j \cdot w^{-1} \cdot (U_0 \cdot \Sigma^*)$ for some $j \in \{1, \ldots, \ell\}$ and $w \in pre(U_\perp)_{W_j}$.

▶ 2. The scheme for equation (A) is similar to the previous one. Because of Lemma 4.26, (A) can be decided by the following conditions. Define $L_j := W_j \cdot \bigcap_{w \in W_j} w^{-1} \cdot (U_A)$ and again $L'_j := W_j \cdot \bigcap_{w \in W_j} w^{-1} \cdot (U_\perp \cdot \Sigma^*)$. Then (A) has a solution if and only if:

- $V_A \subseteq U_A \cup U_\perp \cdot \Sigma^*$
- For all $u \in U_A$: $u \in V_A \cup U_\perp \cdot \Sigma^*$ or there exists a j with $u \in L_j$ or $u \in L'_j$.

The strategy proposed for testing the first condition again requires modification only in so far as constructing an automaton for the representation of V_A is not necessary, since $\mathcal{V}_A$ is already given. For the second condition, we can employ the same arguments as proposed in (1). The only issue remaining is the test for $u \in L_j$. According to Lemma 5.7, the intersection over all $w \in W_j$ in the definition of L_j can be restricted to $w \in pre(U_A)_{W_j}$, if $\overline{pre}(U_A)_{W_j}$ is empty. If $\overline{pre}(U_A)_{W_j}$ is not empty, then L_j is empty itself. We know from Lemma 5.8, that deciding emptyness for $\overline{pre}(U_A)_{W_j}$ requires only polynomial time. Furthermore, the language $pre(U_A)_{W_j}$ is of polynomial size in $\|U_A\|$ and can be computed in polynomial time in $\|U_A\|$ and $|\mathcal{W}_j|$. Thus, with these modifications we can decide $u \in L_j$ in the way formerly described in lemma 4.26. □

By the above lemma solvability can be tested for matching problems, whose right-hand side role languages are represented by nondeterministic finite automata. The question of how to compute the actual solutions under these circumstances has not yet been attended to. We can convince ourselves in Definition 4.26 that the only difficulty imposed by the automata representation is the intersection of left quotients over all elements of the—possibly large—role languages W_j. We have seen in the previous lemma how especially this detail can be handled. The scheme employed there similarly can be used to compute the actual solution languages in polynomial time.

5.4.2 Extension to $\mathcal{FL}_\neg$

In Lemma 4.27, we have seen that only little additional effort is necessary to extend the solution strategy for $\mathcal{FL}_\perp$ to matching problems in $\mathcal{FL}_\neg$. We will see that the same holds for the modified matching algorithm proposed in the previous section. When comparing the equations $(\perp')$ and (A') in Lemmata 4.26 and 4.27 we find that exactly the same problems arise due to right-hand side role languages represented by automata. It is therefore sufficient to employ again the strategy proposed previously for $\mathcal{FL}_\perp$. It should be noted that the constructs $Int(A, \neg A)$ occurring in equation $(\perp')$ do not introduce new problems in this context. Our results on simplifying the intersection of left quotients are sufficient to re-use the strategy originally proposed to compute them.

For $\mathcal{ALN}$, the most interesting part of the matching algorithm comprises the computation of the excluding words. In analogy to the situation for $\mathcal{FL}_\neg$, it is fairly simple to see that the rest of the algorithm proposed in Lemma 4.28 can be extended by the same strategy as seen above. To modify the computation of excluding words accordingly, the respective algorithm, which is provided in [Küs98], would have to be considered in detail. We omit this step, because the overall approach proposed here is weaker than the one to be introduced in Chapter 6.

The generalized substitution introduced in Definition 5.1 is bound to acyclic side conditions. It is not yet clear whether a similar approach can be devised for the acyclic case. For cyclic side conditions, however, we will see in the next chapter that a more intuitive solution strategy exists.

CHAPTER 6

Fixed Points and Side Conditions

Judging by its objective, the present chapter might be seen as belonging to the previous one. We present yet another approach to solve matching problems modulo equivalence with non-strict side conditions in polynomial time. This approach, however, aims at providing a satisfactory scheme for both $\mathcal{ALN}$ and its three sublanguages. Contrary to the strategies discussed previously, it is furthermore intended to cope with cyclic side conditions as well as with acyclic ones.

The idea here is to reduce matching problems with side conditions to such without side conditions. We have already seen that this idea does not bear fruit when pursued in a straight-forward fashion. In Chapter 5, it is shown that the approach of merely syntactically including side conditions into the original matching equation may produce exponentially large matching problems—even when employing intuitive strategies to represent the result in a compact way.

Here, we handle the reduction differently. The transformation of the original matching problem with side conditions into an equivalent one without will not be performed in a single step. On the contrary, we will propose an algorithm to compute a solution by iteratively improving an intermediate result. Every step of this algorithm comprises solving a certain matching problem without side conditions. This approach directly relies on the ability to solve matching problems without side conditions, as addressed in Chapter 4.

In order to prove termination we must make sure that equivalent concept descriptions cannot become arbitrarily large. The $\mathcal{FL}_0$-normal form does not meet this requirement for concept descriptions in $\mathcal{FL}_\bot$, $\mathcal{FL}_\neg$, and $\mathcal{ALN}$. In the Section 2, we therefore specify "reduced normal forms" for these logics. In order to do so, we first need to examine the properties of prefix-free languages in Section 1. The actual algorithm is introduced in Section 3. It will be defined uniformly for all four logics. Thanks to this, the proof of correctness and completeness also can be given simultaneously for all four logics in Section 4. Finally, termination of the algorithm is proved in the last section of this chapter. In order to show termination the properties of reduced normal forms are necessary as prerequisites.

Finally, we will find that the algorithm in fact provides us with an efficient method to solve matching problems with non-strict acyclic or cyclic side conditions in $\mathcal{ALN}$ as well as in its sublanguages. Due to that result, the present chapter may be regarded as the heart of our work.

6.1 Prefix free languages

We define prefix free languages as a specialization of formal languages by introducing a unary function to make a given formal language prefix free.

Definition 6.1 Prefix free languages

$$\begin{aligned} pf\colon \mathfrak{P}(\Sigma^*) &\to \mathfrak{P}(\Sigma^*) \\ L &\mapsto L \setminus (L{\cdot}\Sigma^+) \end{aligned}$$

A language $U \subseteq \Sigma^*$ is called *prefix free* if and only if $U = pf(U)$. □

Intuitively, $pf(L)$ for every word $w \in L$ removes all nontrivial continuations of w. The result is that for every word $w \in pf(L)$, all nontrivial prefixes of w are missing in $pf(L)$. To examine the properties of prefix free sets in greater detail, we must first introduce an appropriate order over finite languages. The definition of multiset orders is taken from [BN98b], where their properties are discussed in depth. However, we employ multiset orders over formal languages and do not need to introduce multisets, which generalize the notion of sets by admitting multiple occurrences of elements.

Definition 6.2 Multiset order for finite languages
Define $(\succ)$ as a multiset order with $(>_{pr})$ on Σ^*. Thus, for finite languages $U, V \subseteq \Sigma^*$ it holds that $V \succ U$ if and only if there exist finite languages $X, Y \subseteq \Sigma^*$ such that:

1. $\emptyset \neq X \subseteq V$
2. $U = (V \setminus X) \cup Y$
3. $\forall y \in Y \exists x \in X\colon x <_{pr} y$ □

According to the definition, finite languages U and V are in prefix order, i.e. $U \succ V$, if and only if U can be transformed into V by performing a modification of the following type one or more times: remove a word u from U and replace it by a finite number of words from $\{u\}{\cdot}\Sigma^+$. Thus, u is replaced by a finite number of (nontrivial) continuations of u. Note that in this modification, u may be removed without substituting any words. This is allowed because in the definition above, the language Y may be empty. The following example illustrates this.

Example 6.3 Multiset order
Let $\Sigma := \{a, b, c\}$ Then $\{a, ab, c\} \succ \{ab, ac, caa, cab, ccc\}$. The definition of the multiset order is satisfied by taking $X := \{a, c\}$ and $Y := \{ac, caa, cab, ccc\}$. On the other hand, we also obtain $\{a, ab, c\} \succ \{ca\}$ by taking $X := \{a, ab, c\}$ and $Y := \{ca\}$. Observe that the relation $U \succ V$ does not imply an obvious relation for the cardinality of the languages or for the length of the longest word contained in them. □

The multiset order can be used to simplify comparing the Σ^*-closure of two given languages. This is addressed by the following lemma.

Lemma 6.4 Σ^*-closures and prefix free languages
Let $U, V \subseteq \Sigma^*$ be finite languages over Σ. Then

1. $U \cdot \Sigma^* = pf(U) \cdot \Sigma^*$
2. $U \cdot \Sigma^* \subset V \cdot \Sigma^*$ iff $pf(U) \prec pf(V)$
3. $U \cdot \Sigma^* = V \cdot \Sigma^*$ iff $pf(U) = pf(V)$.

Proof.
For the sake of brevity, denote $pf(U)$ by U' throughout this lemma. Analogously, denote $pf(V)$ by V'.
▶ 1. Since U' is a subset of U and since the sets on both sides of the equation are Σ^*-closed, it is sufficient to show that $U \setminus U'$ is a subset of $U' \cdot \Sigma^*$. Thus, consider $w \in U \setminus U'$. Then, by definition of prefix free sets, $w \in U \cdot \Sigma^+$. This implies, that in U there exists a word $u \in U$ of minimal length and a word $v \in \Sigma^+$ so that $w = uv$. Consequently, $u \notin U \cdot \Sigma^+$, because in this case the length of u would not be minimal. So we have $u \in U'$, implying that $w = uv \in U' \cdot \Sigma^*$.

▶ 2. ("$\Leftarrow$") If $U' \prec V'$ then, by Definition 6.1, there exist finite sets $X, Y \subseteq \Sigma^*$ with:

1. $\emptyset \neq X \subseteq V'$
2. $U' = (V' \setminus X) \cup Y$
3. $\forall y \in Y \exists x \in X : x <_{pr} y$.

We first prove the non-strict version of the claim, i.e. $U \cdot \Sigma^* \subset V \cdot \Sigma^*$, and then show that the inclusion is strict.

▷ Nonstrict inclusion: As U' equals $(V' \setminus X) \cup Y$, it is sufficient to show that $Y \subseteq V' \cdot \Sigma^*$. Thus, consider an arbitrary $y \in Y$. Because of property 3 of multiset orders it holds that there is an $x \in X \subseteq V'$ so that $x <_{pr} y$. Being less in regard to the prefix order implies, that we obtain $y = xw$ for an appropriate $w \in \Sigma^*$. Since $x \in V'$, this yields $y = xw \in V' \cdot \Sigma^*$, completing the proof.

▷ Strictness of the inclusion: Consider an arbitrary $x \in X \subseteq V'$. According to property 1 of multiset orders, such an x in fact exists. x is no element of $(V' \setminus X)$, because V' is prefix free and thus contains no prefix of x. Now, if $x \in Y$ then property 3 demands that there is another word $x' \in X$ so that $x' <_{pr} X$. This would be a contradiction to V' being prefix free, and therefore: $x \notin U' \cdot \Sigma^*$.

▶ 2. ("$\Rightarrow$") Assume $U' \cdot \Sigma^* \subset V' \cdot \Sigma^*$. Taking advantage of (1), this is equivalent to the original proposition. Define finite languages X, Y in the following way: $X := V' \setminus U'$ and $Y := U' \setminus V'$. We will show that these languages match conditions 1, 2, and 3 stated in the definition of multiset orders.

▷ Property 1: Trivial. X is obviously defined as a subset of V'. If X is empty, then $U' \supseteq V'$, which would rule out $U' \cdot \Sigma^* \subset V' \cdot \Sigma^*$, conflicting with the assumption above.

▷ Property 2: Applying the definitions of X and y, we can expand $(V' \setminus X) \cup Y$ to the expression $(V' \setminus (V' \setminus U')) \cup U' \setminus V'$, which simplifies to $(U' \cap V') \cup U' \setminus V'$. This is obviously equivalent to U'.

▷ Property 3: Consider an arbitrary $y \in Y = U' \setminus V'$. From property 2 of the multiset order we know that $Y \subseteq U' \subset V' \cdot \Sigma^*$. Thus, there are words $v \in V'$ and $w \in \Sigma^*$ such that $y = vw$. This implies $w \neq \varepsilon$, because otherwise y, being equal to v, would be an element of V'. If w is not empty, then v and y are in prefix relation: $v <_{pr} y$. Consequently, v is no element of U', because then U' would not be prefix free. This implies $v \in V' \setminus U'$, which by definition is equivalent to $v \in X$.

▶ 3. ("⇐") This is an immediate consequence of (1). If U' equals V', then obviously $U'\cdot\Sigma^* = V'\cdot\Sigma^*$, which implies $U\cdot\Sigma^* = V\cdot\Sigma^*$, as shown in (1). ("⇒") Reversely assume that $U'\cdot\Sigma^* = V'\cdot\Sigma^*$. According to (1), this is equivalent to the original proposition. It is sufficient to prove the inclusion $U' \subseteq V'$, since the reverse inclusion follows by symmetry.

Consider an arbitrary $u \in U'$. According to the above assumption we have $U' \subseteq V'\cdot\Sigma^*$, which implies the existence of words $v \in V'$ and $w \in \Sigma^*$ with $u = vw$. It reversely holds that $V' \subseteq U'\cdot\Sigma^*$, again implying words $u' \in U'$ and $w' \in \Sigma^*$ so that $v = u'w'$. Therefore, we yield $u = vw = u'w'w$. This implies $w = w' = \varepsilon$, because otherwise U' would not be prefix free, containing a prefix of u. With w equal to ε, we finally obtain $u \in V'$, which had to be shown. □

Observe, that the Σ^*-closure of a language L is uniquely defined by the prefix free version of L. We can also use prefix free languages to guarantee a suffix condition when representing the left quotient of the Σ^*-closure of a language:

Lemma 6.5 Left quotients and prefix free languages
Let $U \subseteq \Sigma^*$ be a finite language and let $w \in \Sigma^*$. Then there exists a finite language $L \subset \Sigma^*$ such that

1. $L\cdot\Sigma^* = w^{-1}(U\cdot\Sigma^*)$ and
2. L is prefix free and
3. L contains only suffixes of words in U.

Proof.
According to [BKBM99], there exists a finite language L' with $L'\cdot\Sigma^* = w^{-1}\cdot(U\cdot\Sigma^*)$. Due to Lemma 6.4, we know that this also holds for $L := pf(L')$. We now show that L contains only suffixes of U, which is sufficient for our claim. Assume a word $v \in L$, which is no suffix of any word in U. Observe, that this implies $v \neq \varepsilon$ because otherwise v would be a trivial suffix of any word in U. By definition of L, we know that v is an element of $w^{-1}\cdot(U\cdot\Sigma^*)$. Thus, there exists a word $u \in U$ and a word $x \in \Sigma^+$ such that $wv = ux \in U\cdot\Sigma^*$. We exclude $x = \varepsilon$, because then v would be a suffix of u. Denote by s the last character of v, i.e. take $s \in \Sigma$ and $v' \in \Sigma^*$ such that $v = v's$. Analogously, let $x = x's$ for an appropriate $x' \in \Sigma^*$. Then we can conclude that $v' \in L$, because $wv' = ux'$ is an element of $U\cdot\Sigma^*$. This implies a contradiction to the language L being prefix free. □

6.2 Reduced normal forms

In $\mathcal{FL}_\bot$, $\mathcal{FL}_\neg$, and $\mathcal{ALN}$, equivalent concept descriptions in $\mathcal{FL}_0$-normal form can differ in size to an arbitrary extent. For instance, $\forall\{\varepsilon\}.\bot \sqcap \forall U_A.A$ is equivalent to $\forall\{\varepsilon\}.\bot$ for every role language U_A. For our algorithm to work, we require normal forms which impose stronger limitations on the size of concept descriptions equivalent to or subsuming each other. For this purpose, reduced normal forms for $\mathcal{FL}_\bot$, $\mathcal{FL}_\neg$, and $\mathcal{ALN}$ are introduced. These are not necessary for $\mathcal{FL}_0$, since here the $\mathcal{FL}_0$-normal is already sufficient.

6.2.1 Reduced normal forms for $\mathcal{FL}_\bot$

Let us now define the first reduced normal form. As done for prefix free sets, we define it by specifying an operation to transform a given concept description into its reduced normal form.

Definition 6.6 Reduced normal form
Let C be an $\mathcal{FL}_\bot$-concept description in U-labelled $\mathcal{FL}_0$-normal form. Its corresponding *$U^\downarrow$-labelled reduced normal form* $C^\downarrow$ is defined as follows:

$$C^\downarrow := \forall U_\bot^\downarrow.\bot \sqcap \bigsqcap_{A\in\mathcal{C}} \forall U_A^\downarrow.A$$

where for $A \in \mathcal{C}$:

$$U_\bot^\downarrow := pf(U_\bot)$$
$$U_A^\downarrow := U_A \setminus U_\bot^\downarrow\cdot\Sigma^*$$

A concept description C is called *reduced*, if C is in $\mathcal{FL}_0$-normal form and if it coincides with $C^\downarrow$ in every occurring role language. The notion of reduction can be extended to substitutions. For a substitution σ, the reduced substitution $\sigma^\downarrow$ is established by defining $\sigma^\downarrow(X) := \sigma(X)^\downarrow$ for every variable X in the domain of σ. □

The above definition implies as immediate consequences the following simple properties, which are stated without proof.

Corollary 6.7 Properties
Let C be an $\mathcal{FL}_\bot$-concept descriptions in U-labelled $\mathcal{FL}_0$-normal form. Then
1. $U_\bot^\downarrow$ is prefix free and $U_A^\downarrow \cap U_\bot^\downarrow\cdot\Sigma^*$ is empty for every $A \in \mathcal{C}$
2. The reduced normal form $C^\downarrow$ can be computed in polynomial time in the size of C.

It will be particularly useful that there is no overlap between the role language $U_\bot^\downarrow$ and the Σ^*-closure of $U_A^\downarrow$. The role languages for $C^\downarrow$ can be constructed in polynomial time using treelike automata, for which the complement and the Σ^*-closure can be computed in linear time. It also takes only polynomial time to make a given finite role language prefix free. The ability to compute reduced normal forms in polynomial time will not be required in the remainder of this chapter. Nevertheless, it might be an important property in the context of presenting the output of matching algorithms in a compact way.

Recall that *pf* in Chapter 2 was defined to make the input language prefix free. The purpose of reduced normal forms is to simplify the characterization of subsumption and equivalence. One can see that in the above definition exactly those languages are made prefix free, whose Σ^*-closure appears in the characterization of the subsumption proposed in Lemma 3.8. Furthermore, by subtracting the Σ^*-closure from the other role languages, we make sure that all unions in the characterising conditions are disjoint. In the next lemma we will see that this is sufficient to reduce equivalence to equality.

Lemma 6.8 Properties
Let B, C, D be $\mathcal{FL}_\bot$-concept descriptions. Let B be in W-labelled $\mathcal{FL}_0$-normal form, let C be in U-labelled reduced normal form, and D in V-labelled reduced normal form. Then:
1. $B \equiv B^\downarrow$
2. $C \equiv D$ iff $U_H = V_H$ for all $H \in \{\bot\} \cup \mathcal{C}$
3. $C \sqsubset D$ iff one of the following conditions holds:
   (a) $U_\bot \succ V_\bot$ and $V_A \subseteq U_A \cup U_\bot\cdot\Sigma^*$ for all $A \in \mathcal{A}$
   (b) $U_\bot = V_\bot$ and $U_A \supseteq V_A$ for all $A \in \mathcal{C}$ and there exists an $A \in \mathcal{C}$ with $U_A \supset V_A$.

Proof.
▶ 1. We have seen in Lemma 3.8 that it is sufficient to prove the following two conditions:

- $W_\bot\cdot\Sigma^* = W_\bot^\downarrow\cdot\Sigma^*$
- $W_A \cup W_\bot\cdot\Sigma^* = W_A^\downarrow \cup W_\bot^\downarrow\cdot\Sigma^*$ for all $A \in \mathcal{C}$.

The first condition was shown as a property of prefix free languages in Lemma 6.4. For the second condition, we can therefore conclude for every A that $W_A^{\downarrow} \cup W_\bot^{\downarrow} \cdot \Sigma^*$ is equal to $W_A^{\downarrow} \cup W_\bot \cdot \Sigma^*$. We may add $(W_A \cap W_\bot \cdot \Sigma^*)$, which is a subset of $W_\bot \cdot \Sigma^*$, thus yielding $W_A^{\downarrow} \cup (W_A \cap W_\bot \cdot \Sigma^*) \cup W_\bot \cdot \Sigma^*$. According to the definition of reduced normal forms, W_A equals $W_A^{\downarrow} \cup (W_A \cap W_\bot \cdot \Sigma^*)$. Therefore, $W_A^{\downarrow} \cup (W_A \cap W_\bot \cdot \Sigma^*) \cup W_\bot \cdot \Sigma^*$ equals $W_A \cup W_\bot \cdot \Sigma^*$.

▶ 2. ("⇐") is trivial. ("⇒") Assume $C \equiv D$. Due to Lemma 3.8, this again is equivalent to $U_\bot \cdot \Sigma^* = V_\bot \cdot \Sigma^*$ and $U_A \cup U_\bot \cdot \Sigma^* = V_A \cup V_\bot \cdot \Sigma^*$ for all $A \in \mathcal{C}$. Since C and D are assumed to be reduced, this implies $U_\bot = V_\bot$, according to the properties of prefix free sets. Furthermore, due to the definition of reduced normal forms, U_A and $U_\bot \cdot \Sigma^*$ are disjoint for every A. The same applies to V_A and $V_\bot \cdot \Sigma^*$. Therefore, $U_A \cup U_\bot \cdot \Sigma^* = V_A \cup V_\bot \cdot \Sigma^*$ implies $U_A = V_A$ for all A, which was to be shown.

▶ 3. ("⇒") Assume $C \sqsubset D$. Then we again have $U_\bot \cdot \Sigma^* \supseteq V_\bot \cdot \Sigma^*$. We distinguish two cases depending on whether the inclusion is strict or not.

▷ Strict inclusion: If $U_\bot \cdot \Sigma^* \supset V_\bot \cdot \Sigma^*$, we can infer $U_\bot \succ V_\bot$, as shown in Lemma 6.4. We know from the characterization of the subsumption that $U_A \cup U_\bot \cdot \Sigma^* \supseteq V_A \cup V_\bot \cdot \Sigma^*$ for all $A \in \mathcal{C}$. We may remove $V_\bot \cdot \Sigma^*$ from the right-hand side of the inclusion, yielding the assertion for case (a), $V_A \subseteq U_A \cup U_\bot \cdot \Sigma^*$.

▷ Equality: If $U_\bot \cdot \Sigma^* = V_\bot \cdot \Sigma^*$, we have $U_\bot = V_\bot$, because C and D are reduced and therefore $U_\bot$ and $V_\bot$ are prefix free. The subsumption $C \sqsubset D$ also implies that $U_A \cup U_\bot \cdot \Sigma^* \supseteq V_A \cup V_\bot \cdot \Sigma^*$ for every A. The unions on both sides of the inclusion are disjoint, as stated in Corollary 6.7. Taking advantage of the equality of $U_\bot \cdot \Sigma^*$ and $V_\bot \cdot \Sigma^*$, we obtain $U_A \supseteq V_A$ for every $A \in \mathcal{C}$. There has to be one A with a strict inclusion $U_A \supset V_A$. Otherwise, C and D would agree on all role languages, implying equivalence as shown in (2). Thus, the assertion for case (b) holds.

▶ 3. ("⇐") We have to show that both conditions for the subsumption as stated in Lemma 3.8 are met. Assuming case (b), this can be seen immediately. Consider case (a). If $U_\bot \succ V_\bot$ holds, the first condition for the subsumption is met as a consequence of Lemma 6.4, obtaining $U_\bot \cdot \Sigma^* \supset V_\bot \cdot \Sigma^*$. We have assumed that $V_A \subseteq U_A \cup U_\bot \cdot \Sigma^*$. Adding $V_\bot \cdot \Sigma^*$ on both sides yields $V_A \cup V_\bot \cdot \Sigma^* \subseteq U_A \cup U_\bot \cdot \Sigma^* \cup V_\bot \cdot \Sigma^*$. As $V_\bot \cdot \Sigma^*$ is a subset of $U_\bot \cdot \Sigma^*$, this is equivalent to $V_A \cup V_\bot \cdot \Sigma^* \subseteq U_A \cup U_\bot \cdot \Sigma^*$. Thus, the second condition of the subsumption is met for every $A \in \mathcal{C}$. We yield strict subsumption $C \sqsubset D$, because otherwise $U_\bot = V_\bot$. □

In part (3) of the lemma a complete characterization of strict subsumption is provided for the sake of completeness. For our purposes we do not require the equivalence in full detail. It would have been suficient to prove that if $C \sqsubset D$, then either we have $U_\bot \succ V_\bot$ or condition (b) holds. It might be interesting that condition (a) can be put a little stricter, stating: $U_\bot \succ V_\bot$ and $V_A \subseteq U_A \cup (U_\bot \cdot \Sigma^* \setminus V_\bot \cdot \Sigma^*)$ for all $A \in \mathcal{A}$. For the remainder of this chapter, however, this will not be required.

6.2.2 Reduced normal forms for $\mathcal{FL}_\neg$

For $\mathcal{FL}_\neg$, we follow the same pattern as seen in the previous section. Firstly, the reduction operation is expanded in such a way that it works with negated atomic concepts as well.

Definition 6.9 Reduced normal form
Let C be an $\mathcal{FL}_\neg$-concept description in U-labelled $\mathcal{FL}_0$-normal form. Like in Definition 6.6, define its corresponding reduced normal form $C^\downarrow$ by modifying the role languages:

$$C^\downarrow := \forall U_\bot^\downarrow.\bot \sqcap \bigsqcap_{A\in\mathcal{C}} \forall U_A^\downarrow.A \sqcap \bigsqcap_{A\in\mathcal{C}} \forall U_{\neg A}^\downarrow.\neg A$$

where for $A \in \mathcal{C}$:

$$U_\bot^\downarrow := pf(U_\bot \cup \bigcup_{A\in\mathcal{C}} U_A \cap U_{\neg A})$$
$$U_A^\downarrow := U_A \setminus U_\bot^\downarrow\cdot\Sigma^*$$

Again, if C is reduced, then its role languages are identical to those of $C^\downarrow$. We extend the notion of reduction to substitutions as in Definition 6.6. □

Observe that in this definition the role language $U_\bot$ referring to the bottom concept may increase in size when normalized. Contrary to $\mathcal{FL}_0$, it is possible to have inconsistencies without involving the bottom concept. The reduced normal form for $\mathcal{FL}_\neg$ aims at making all implicit inconsistencies explicit, i.e. whenever an expression like $\forall w.(A \sqcap \neg A)$ occurs, w is removed from the role languages referring to A and $\neg A$ and is included in the language for the bottom concept. The definition of excluding words again implies some inportant properties, which are stated below without proof.

Corollary 6.10 Properties
Let C be an $\mathcal{FL}_\neg$-concept descriptions in U-labelled $\mathcal{FL}_0$-normal form. Then:
1. $U_\bot^\downarrow$ is prefix free and $U_\bot^\downarrow = (U_\bot^\downarrow)^\frown$.
2. $U_H^\downarrow \cap (U_\bot^\downarrow)^\frown\cdot\Sigma^*$ is empty for every $H \in \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\}$.
3. $U_A^\downarrow \cap U_{\neg A}^\downarrow$ is empty for every $A \in \mathcal{C}$.
4. The reduced normal form $C^\downarrow$ can be computed in polynomial time in the size of C.

Since $(U_\bot^\downarrow)^\frown$ is defined as $U_\bot^\downarrow \cup \bigcup_{A\in\mathcal{C}}(U_A^\downarrow \cap U_{\neg A}^\downarrow)$, the above assertions are readily obtained from the definition of reduced normal forms. Computing the reduced normal form in polynomial time can again be accomplished by employing treelike automata. By virtue of these properties, we again achieve the desired simplification for the characterization of the subsumption. In the next lemma it is shown that the results obtained for $\mathcal{FL}_\neg$ resemble those for $\mathcal{FL}_\bot$ seen in the last section.

Lemma 6.11 Properties
Let B, C, D be $\mathcal{FL}_\neg$-concept descriptions. Let B be in W-labelled $\mathcal{FL}_0$-normal form, let C be in U-labelled reduced normal form, and D in V-labelled reduced normal form. Let $\mathcal{H} := \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\}$. Then
1. $B \equiv B^\downarrow$
2. $C \equiv D$ iff $U_H = V_H$ for all $H \in \{\bot\} \cup \mathcal{H}$
3. $C \sqsubset D$ iff one of the following conditions holds:
   (a) $U_\bot \succ V_\bot$ and $V_H \subseteq U_H \cup U_\bot\cdot\Sigma^*$ for all $H \in \mathcal{H}$
   (b) $U_\bot = V_\bot$ and $U_H \supseteq V_H$ for all $H \in \mathcal{H}$ and there exists an $H \in \mathcal{H}$ with $U_A \supset V_A$.

Proof.
▶ 1. Due to Lemma 3.9, it is sufficient to prove that the following conditions hold:

- $\widehat{W}_\bot\cdot\Sigma^* = (W_\bot^\downarrow)^\frown\cdot\Sigma^*$
- $W_H \cup \widehat{W}_\bot\cdot\Sigma^* = W_H^\downarrow \cup (W_\bot^\downarrow)^\frown\cdot\Sigma^*$ for all $H \in \mathcal{H}$.

▷ First condition: By definition, $\widehat{W}_\bot \cdot \Sigma^*$ equals $(W_\bot \cup \bigcup_{A \in \mathcal{C}} W_A \cap W_{\neg A}) \cdot \Sigma^*$, which is equivalent to the prefix free version $pf(W_\bot \cup \bigcup_{A \in \mathcal{C}} W_A \cap W_{\neg A}) \cdot \Sigma^*$, as we have seen in Lemma 6.4. Applying the definition of reduced normal forms, this is equivalent to $W_\bot^\downarrow \cdot \Sigma^*$. The intersection of $W_A^\downarrow$ and $W_{\neg A}^\downarrow$ is empty for every $A \in \mathcal{C}$, as stated in Corrolary 6.10. We may therefore add $(\bigcup_{A \in \mathcal{C}} W_A^\downarrow \cap W_{\neg A}^\downarrow)$ to the expression, so that we end up with $(W_\bot^\downarrow \cup \bigcup_{A \in \mathcal{C}} W_A^\downarrow \cap W_{\neg A}^\downarrow) \cdot \Sigma^*$. This equals $(W_\bot^\downarrow)^\frown \cdot \Sigma^*$, as can be verified from the definition.

▷ Second condition: Taking advantage of (1), we can see that $W_H^\downarrow \cup (W_\bot^\downarrow)^\frown \cdot \Sigma^*$ is equal to $W_H^\downarrow \cup \widehat{W}_\bot \cdot \Sigma^*$ for every $H \in \mathcal{H}$. We may add a subset of the second term, yielding the expression $W_H^\downarrow \cup (W_H \cap \widehat{W}_\bot \cdot \Sigma^*) \cup \widehat{W}_\bot \cdot \Sigma^*$. The language $W_H^\downarrow$ is defined as $W_H \setminus W_\bot^\downarrow \cdot \Sigma^*$. As stated in Corollary 6.10, this equals $W_H \setminus (W_\bot^\downarrow)^\frown \cdot \Sigma^*$, which in (1) is shown equal to $W_H \setminus \widehat{W}_\bot \cdot \Sigma^*$. The expression $W_H^\downarrow \cup (W_H \cap \widehat{W}_\bot \cdot \Sigma^*) \cup \widehat{W}_\bot \cdot \Sigma^*$ can therefore be simplified to $W_H \cup \widehat{W}_\bot \cdot \Sigma^*$, yielding the desired result.

▶ 2. ("⇐") Trivial. ("⇒") According to Corollary 6.10, we have $\widehat{U}_\bot = U_\bot$ and $\widehat{V}_\bot = V_\bot$. When replacing these role languages, the proposition and the characterization of the subsumption are analogous to those for $\mathcal{FL}_\bot$. Consequently, the proof is identical to (2) in the previous Lemma 6.8.

▶ 3. Again, taking into account that $\widehat{U}_\bot = U_\bot$ and $\widehat{V}_\bot = V_\bot$, we can prove the proposition in the same way as seen in (3) in the previous lemma. □

One can see that the additional complexity of concept descriptions in $\mathcal{FL}_\neg$ is hidden completely by the reduced normal form. It should be noted that, same as for $\mathcal{FL}_\bot$, we will not require the full characterization of the strict subsumption for our reasoning. It is therefore sufficient to keep in mind that $C \sqsubset D$ implies that either $U_\bot \succ V_\bot$ holds or condition (b) applies. However, the result enables us to discover that the size of the role languages V_A and $V_{\neg A}$ occurring in D is limited.

6.2.3 Reduced normal forms for $\mathcal{ALN}$

When introducing reduced normal forms for $\mathcal{ALN}$-concept descriptions, we have to face two additional problems. Firstly, the set of all inconsistencies explicitly occurring or implicitly included in a concept description cannot be obtained in such a straightforward way as in the previous two logics. Secondly, we also have to cope with number restrictions. In the following definition, we utilize the notion of excluding words, which have been introduced in the context of $\mathcal{ALN}$-concept descriptions in Definition 3.3.

Definition 6.12 Reduced normal form
Let C be an $\mathcal{ALN}$-concept description in U-labelled $\mathcal{FL}_0$-normal form. Define the reduced normal form of C by modifying its role languages. It has been stated in [BKBM99] that there exists a finite language U_{E_C} with $E_C = U_{E_C} \cdot \Sigma^*$. Using this language, define $C^\downarrow$ as:

$$C^\downarrow := \forall U_\bot^\downarrow.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall U_A^\downarrow.A \sqcap \bigsqcap_{A \in \mathcal{C}} \forall U_{\neg A}^\downarrow.\neg A$$
$$\sqcap \bigsqcap_{(\geq nR) \in \mathcal{N}_\geq} \forall U_{(\geq nR)}^\downarrow.(\geq nR) \sqcap \bigsqcap_{(\leq nR) \in \mathcal{N}_\leq} \forall U_{(\leq nR)}^\downarrow.(\leq nR)$$

where for $A \in \mathcal{C}$, $(\leq nR) \in \mathcal{N}_{\leq}$, and $(\geq nR) \in \mathcal{N}_{\geq}$:

$$\begin{aligned} U_{\bot}^{\downarrow} &:= pf(U_{E_C}) \\ U_{A}^{\downarrow} &:= U_A \setminus E_C \\ U_{\neg A}^{\downarrow} &:= U_{\neg A} \setminus E_C \\ U_{(\geq nR)}^{\downarrow} &:= \bigcup_{m \geq n} U_{(\geq mR)} \setminus E_C \\ U_{(\leq nR)}^{\downarrow} &:= \bigcup_{m \leq n} U_{(\leq mR)} \setminus E_C \cdot R^{-1} \end{aligned}$$

Analogous to the previous cases, the notion of reduction is extended to substitutions. □

In spite of the formally more complex definition, the objective of the above normal form is equal to those seen before. Inconsistencies are made explicit by augmenting the role language of the bottom concept and the other role languages are minimized as much as possible. Observe that the reduced role language $U_{\bot}^{\downarrow}$ in fact is well-defined, because for languages of the form $L \cdot \Sigma^*$ the set $pf(L)$ is unique. The definition of reduced normal forms again implies some basic properties, which are presented in the corollary below.

Corollary 6.13 Properties
Let C be an $\mathcal{ALN}$-concept descriptions in U-labelled $\mathcal{FL}_0$-normal form. Then:

1. $U_{\bot}^{\downarrow}$ is prefix free
2. $U_{H}^{\downarrow} \cap E_{C^{\downarrow}}$ is empty for every $\mathcal{H} := \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\} \cup \mathcal{N}_{\geq}$.
   Furthermore, $U_{(\leq nR)}^{\downarrow} \cap E_{C^{\downarrow}} \cdot R^{-1}$ is empty for every $(\leq nR) \in \mathcal{N}_{\leq}$
3. $\bigcup_{m \geq n} U_{(\geq mR)}^{\downarrow} = U_{(\geq nR)}^{\downarrow}$ for all $(\geq nR) \in \mathcal{N}_{\geq}$ and analogously for all $(\leq nR) \in \mathcal{N}_{\leq}$
4. The reduced normal form $C^{\downarrow}$ can be computed in polynomial time in the size of C.

As stated in [BKBM99], a role language U_{E_C} with $E_C = U_{E_C} \cdot \Sigma^*$ can be computed in polynomial time. With the aid of treelike automata, it therefore takes only polynomial time to compute the reduced normal form of C. In order to examine the properties of our normal form closer, we have to procure a better characterization for the set of excluding words from [Küs98]. The following definition is necessary as a preparation.

Definition 6.14 Required words
Let C be an $\mathcal{ALN}$-concept description in U-labelled $\mathcal{FL}_0$-normal form. Let v and v' be words over Σ. Let $|v| =: m$ and $|vv'| =: n$ and $v' =: R_{m+1} \ldots R_n$. Then vv' is required by C starting from v iff for all $i \in \{m, \ldots, n-1\}$ there exist positive integers $k_{i+1} \geq 1$ such that $vR_{m+1} \ldots R_i \in U_{(\geq k_{i+1} R_{i+1})}$. □

Intuitively, the continuation vv' is required by a concept description C starting from v, iff there is a sequence of $(\geq)$-number restrictions for every prefix of vv' between v and vv' demanding the presence of the respective following prefix. We give a small example to clarify this.

Example 6.15 Required words
Assume $\Sigma := \{R, S\}$ and let $C := A \sqcap \forall\{RS, RSR\}.(\geq 1R) \sqcap \forall\{RSR\}.(\geq 2S)$. Then the words $RSRR$ and $RSRS$ are required by C starting from RS. □

With the notion of required words we can characterize excluding words for $\mathcal{ALN}$-concept descriptions by the following lemma.

Lemma 6.16 Characterization of excluding words
Let C be an $\mathcal{ALN}$-concept description in U-labelled $\mathcal{FL}_0$-normal form. Let w be a word over Σ. Then $w \in E_C$ iff

1. there exists a prefix $v \in \Sigma^*$ of w and a word $v' \in \Sigma^*$ such that vv' is required by C starting from v and
   (a) $vv' \in U_\bot$, or
   (b) there is an atomic concept $A \in \mathcal{C}$ with $vv' \in U_A \cap U_{\neg A}$, or
   (c) there are number resttrictions $(\geq lR) \in \mathcal{N}_\geq$ and $(\leq rR) \in \mathcal{N}_\geq$ such that $l > r$ and $v \in U_{(\geq lR)} \cap U_{(\leq rR)}$; or
2. there exists a prefix vR of w (with $v \in \Sigma^*, R \in \Sigma$) such that $v \in U_{(\leq 0R)}$.

Now we are set to examine reduced normal forms in detail. Before addressing the standard questions of correctness, equivalence, and subsumption, however, we first introduce one auxiliary result regarding the notion of excluding words, which will be required in Lemma 6.19. In the next lemma, it is shown that transforming a concept description into reduced normal forms does not change its properties in respect to required words.

Lemma 6.17 Required words and reduced normal forms
Let C be an $\mathcal{ALN}$-concept description in U-labelled $\mathcal{FL}_0$-normal form and let v, v' be words over Σ. Then, if vv' is required by $C^\downarrow$ starting from v then vv' is required by C starting from v.

Proof.
To simplify notation, denote $|v| =: s$, $|vv'| =: t$, and $vv' =: R_1 R_2 \ldots R_t$. If vv' is required by $C^\downarrow$ starting from v, then by definition it holds for all $i \in \{s, \ldots, t-1\}$ that there exists a positive integer $k \geq 1$, so that $R_1 \ldots R_i \in U^\downarrow_{(\geq kR_{i+1})}$. By definition of reduced normal forms, this implies that $R_1 \ldots R_i \in \bigcup_{n \geq k} U_{(\geq nR_{i+1})} \setminus E_C$. No n under the union is smaller than k. Consequently, there exists an integer $k' \geq k$ so that $R_1 \ldots R_i$ is an element of $U_{(\geq k'R_{i+1})} \setminus E_C$. Obviously, we can include all the words subtracted by E_C, thus obtaining that $R_1 \ldots R_i \in U_{(\geq k'R_{i+1})}$. This is equivalent to vv' being required by C starting from v, which was to be shown □

A simplified characterization for the set of excluding words is now proposed for concept descriptions in reduced normal form. It is shown by the next lemma that only case (1a) of the characterization given in Lemma 6.16 is relevant for the reduced normal form of concept descriptions.

Lemma 6.18 Excluding words and reduced normal forms
Let C be an $\mathcal{ALN}$-concept description in U-labelled $\mathcal{FL}_0$-normal form. Let w be a word over Σ. Then, $w \in E_{C^\downarrow}$ iff there exists a prefix $v \in \Sigma^*$ of w and a word $v' \in \Sigma^*$ with: vv' is required by $C^\downarrow$ starting from v and $vv' \in U_\bot$.

Proof.
Consider a word $w \in E_{C^\downarrow}$. It is sufficient to prove that the cases (1b), (1c), or (2) specified in the characterization of $E_{C^\downarrow}$ do not apply.

▷ Case (1b): Then there exists a prefix $v \in \Sigma^*$ of w, a word $v' \in \Sigma^*$, and an atomic concept $A \in \mathcal{C}$, so that vv' is required by $C^\downarrow$ starting from v and $vv' \in (U^\downarrow_A \cap U^\downarrow_{\neg A})$. Applying the definition of reduced normal forms, this implies that vv' is an element of $U_A \cap U_{\neg A}$, but no element of E_C. By Definition of the semantics of $\mathcal{ALN}$-concept descriptions, this implies $C \sqsubseteq \forall vv'.\bot$. As a consequence of Definition 3.5, this implies $vv' \in E_C$, in contradiction to the above finding that $vv' \notin E_C$.

▷ Case (1c): Then we have an analogous word vv' and nonnegative numbers $l > r$ with $vv' \in U_{(\geq lR)} \cap U_{(\leq rR)}$. Again by definition of reduced normal forms, we conclude that vv' is

an element of the intersection $\bigcup_{l' \geq l} U_{(\geq l' R)} \cap \bigcup_{r' \leq r} U_{(\leq r' R)}$, but it is not in E_C. Therefore, we can find integers $l' \geq l$ and $r' \leq r$ such that $vv' \in U_{(\geq l' R)} \cap U_{(\leq r' R)}$. Analogous to case (1b), the semantics of $\mathcal{ALN}$ then implies $C \sqsubseteq \forall vv'.\bot$. Due to Definition 3.5, this entails $vv' \in E_C$, contradicting the above statement.

▷ Case (2): We prove that in the reduced normal form $C^{\downarrow}$ the role language $U^{\downarrow}_{(\leq 0R)}$ is empty for every atomic role $R \in \Sigma$. As 0 is the least nonnegative integer, for every atomic role $R \in \Sigma$ the definition of $U^{\downarrow}_{(\leq 0R)}$ can be simplified to $U_{(\leq 0R)} \setminus E_C{\cdot}R^{-1}$, omitting the union. Therefore, if $U^{\downarrow}_{(\leq 0R)}$ is not empty, it contains an element of $U_{(\leq 0R)}$. Thus, assume $w \in U_{(\leq 0R)}$ for a word w. According to the definition of number restrictions, this implies that w has no successors in regard to R. Consequently, $wR \in E_C$. Obviously, we can infer $w \in E_C{\cdot}R^{-1}$. In the definition of $U^{\downarrow}_{(\leq 0R)}$, the set $E_C{\cdot}R^{-1}$ is subtracted from the rest, implying $w \notin U^{\downarrow}{}_{(\leq 0R)}$. Case (2) does therefore not apply to $C^{\downarrow}$. □

The above result suggests a simpler proof of the correctness of the normal form. The standard questions, correctness and modified characterizations for equivalence and subsumption, are addressed in the next lemma.

Lemma 6.19 Properties
Let B, C, D be $\mathcal{ALN}$-concept descriptions. Let B be in W-labelled $\mathcal{FL}_0$-normal form, let C be in U-labelled reduced normal form, and D in V-labelled reduced normal form. Let $\mathcal{H} := \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\} \cup \mathcal{N}_{\leq} \cup \mathcal{N}_{\geq}$. Then

1. $B \equiv B^{\downarrow}$
2. $C \equiv D$ iff $U_H = V_H$ for all $H \in \{\bot\} \cup \mathcal{H}$
3. $C \sqsubset D$ iff one of the following conditions holds:
   (a) $U_\bot \succ V_\bot$ and $V_H \subseteq U_H \cup U_\bot{\cdot}\Sigma^*$ for all $H \in \mathcal{H} \setminus \mathcal{N}_{\leq}$ and
   $V_H \subseteq U_H \cup U_\bot{\cdot}\Sigma^* \cup U_\bot{\cdot}R^{-1}$ for all $(\leq nR) := H \in \mathcal{N}_{\leq}$
   (b) $U_\bot = V_\bot$ and $U_H \supseteq V_H$ for all $H \in \mathcal{H}$ and there exists an $H \in \mathcal{H}$ with $U_A \supset V_A$.

Proof.
▶ 1. In Lemma 3.6, equivalence of $\mathcal{ALN}$-concept decriptions was is characterized by the following conditions. For $A \in \mathcal{C}$, $(\leq mR) \in \mathcal{N}_{\leq}$, and $(\geq mR) \in \mathcal{N}_{\geq}$:

1. $E_{B^{\downarrow}} = E_B$
2. $W^{\downarrow}_A \cup E_{B^{\downarrow}} = W_A \cup E_B$
3. $W^{\downarrow}_{\neg A} \cup E_{B^{\downarrow}} = W_{\neg A} \cup E_B$
4. $\bigcup_{m \geq n} W^{\downarrow}_{(\geq mR)} \cup E_{B^{\downarrow}} = \bigcup_{m \geq n} W_{(\geq mR)} \cup E_B$
5. $\bigcup_{m \leq n} W^{\downarrow}_{(\leq mR)} \cup E_{B^{\downarrow}}{\cdot}R^{-1} = \bigcup_{m \leq n} W_{(\leq mR)} \cup E_B{\cdot}R^{-1}$

▷ Condition 1: Prove $E_{B^{\downarrow}} \subseteq E_B$. Consider an arbitrary $w \in E_{B^{\downarrow}}$. Due to the simplified characterization of exclusion for reduced normal forms, this implies that there exists a prefix $v \in \Sigma^*$ of w and a word $v' \in \Sigma^*$ such that vv' is required by B starting from v and $vv' \in W^{\downarrow}_{\bot}$. According to Definition 6.9, this implies that vv' is in $pf(W_{E_B}) \subseteq E_B$ for an appropriate finite language W_{E_B} with $E_B = W_{E_B}{\cdot}\Sigma^*$. Due to Lemma 6.17, we know that vv' is required by B starting from v. Since $vv' \in E_B$, this implies $v \in E_B$. As E_B is Σ^*-closed and as v is a prefix of w, we obtain $w \in E_B$.

Prove $E_B \subseteq E_{B^{\downarrow}}$. If $w \in E_B$ then there exists a prefix w' of w and a word $w'' \in \Sigma^*$, so that $w = w'w''$ and w' is an element of $pf(W_{E_B})$. Applying the definition of reduced normal forms, we have $w' \in W^{\downarrow}_{\bot}$. This implies $B^{\downarrow} \sqsubseteq \forall w'.\bot$, which is subsumed by $\forall w'w''.\bot$, according to the semantics of $\bot$. Due to the definition of E_B, this yields $w'w'' = w \in E_{B^{\downarrow}}$.

Combining the above two results, we obtain $E_{B^{\downarrow}} = E_B$, which was to be shown.

▷ Condition 2 and 3: Taking into account the result of (1), it holds that $W_A^{\downarrow} \cup E_{B\downarrow}$ is equal to $W_A^{\downarrow} \cup E_B$ for every $A \in \mathcal{C}$. Applying the definition of $W_A^{\downarrow}$ yields the expression $(W_A \setminus E_B) \cup E_B$, which is obviously equal to $W_A \cup E_B$. The same argument holds for negated atomic concepts $\neg A$.

▷ Condition 4 and 5: Again, the result of (1) and the definition of $W_{(\geq mR)}^{\downarrow}$ enable us to expand $\bigcup_{m \geq n} W_{(\geq mR)}^{\downarrow} \cup E_{B\downarrow}$ to the expression $\bigcup_{m \geq n} (\bigcup_{p \geq m} W_{(\geq pR)} \setminus E_B) \cup E_B$. By applying distributivity over the union, we obtain $(\bigcup_{m \geq n} \bigcup_{p \geq m} W_{(\geq pR)}) \setminus E_B \cup E_B$, which can be simplified to $(\bigcup_{m \geq n} W_{(\geq mR)}) \setminus E_B \cup E_B$. We can omit subtracting E_B before adding it again, so that we finally have $(\bigcup_{m \geq n} W_{(\geq mR)}) \cup E_B$.

In (1) we have seen that $E_{B\downarrow} = E_B$. This implies $E_{B\downarrow} \cdot R^{-1} = E_B \cdot R^{-1}$ for every atomic role R. Consequently, the above argument applies to condition 5 as well.

▶ 2. ("⇐") Trivial. ("⇒") If $C \equiv D$, then the characterization of the subsumption allows us to conclude the following conditions again:

1. $E_C = E_D$
2. $U_A \cup E_C = V_A \cup E_D$
3. $U_{\neg A} \cup E_C = V_{\neg A} \cup E_D$
4. $\bigcup_{m \geq n} U_{(\geq mR)} \cup E_C = \bigcup_{m \geq n} V_{(\geq mR)} \cup E_D$
5. $\bigcup_{m \leq n} U_{(\leq mR)} \cup E_C \cdot R^{-1} = \bigcup_{m \leq n} V_{(\leq mR)} \cup E_D \cdot R^{-1}$

Taking advantage of Lemma 6.4, we can infer from condition 1 that $pf(U_{E_C}) = pf(V_{E_D})$, which is equivalent to $U_\perp = V_\perp$, since both concept descriptions are assumed to be reduced. Due to reduction, it also holds that $U_A = U_A \setminus E_C$ and analogously $V_A = V_A \setminus E_D$. Therefore, the unions in condition 2 are disjoint. Because of condition 1 we may replace E_D by E_C in condition 2, which yields $U_A = V_A$. The same argument applies to condition 3. Because C and D are reduced, the role languages $U_{(\leq mR)}$ and $U_{(\geq mR)}$ already contain the union over all lesser and the union over all greater numbers respectively, as stated in Corollary 6.13. In condition 4 and 5, we may therefore ommit the unions over m. Moreover, the role languages in condition 4 and 5 are defined as disjoint to E_C and E_D respectively, so that finally the argument for conditions 2 and 3 also applies, yielding $U_{(\leq nR)} = V_{(\leq nR)}$ for every number restriction $(\leq nR) \in \mathcal{N}_{\leq}$ and analogously $U_{(\geq nR)} = V_{(\geq nR)}$ for every $(\geq nR) \in \mathcal{N}_{\geq}$.

▶ 3. ("⇒") If $C \sqsubset D$, then from the characterization of subsumption we know that $E_C \supseteq E_D$. We first consider the case that this inclusion is strict, then the case of equality of the languages.

▷ $E_C \supset E_D$: Then, as stated in [BKBM99], there are finite languages U_{E_C} and V_{E_D} such that $pf(U_{E_C}) \cdot \Sigma^* \supset pf(V_{E_D}) \cdot \Sigma^*$. Due to the definition of reduced normal forms, this is equivalent to the inclusion $U_\perp \cdot \Sigma^* \supset V_\perp \cdot \Sigma^*$. According to Lemma 6.4, we can then infer $U_\perp \succ V_\perp$. Since $C \sqsubset D$, we know from the characterization of subsumption that $U_H \cup E_C \supseteq V_H \cup E_D$ for all $H \in \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\}$. As mentioned above, this inclusion is equivalent to $U_H \cup U_\perp \cdot \Sigma^* \supseteq V_H \cup V_\perp \cdot \Sigma^*$. We may drop the term $V_\perp \cdot \Sigma^*$ on the right-hand side, obtaining the desired result for all $H \in \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\}$.

For $(\geq nR) \in \mathcal{N}_{\geq}$, we similarly yield $\bigcup_{m \geq n} U_{(\leq mR)} \cup U_\perp \cdot \Sigma^* = \bigcup_{m \leq n} V_{(\leq mR)} \cup V_\perp \cdot \Sigma^*$. As mentioned before, the union over all $m \geq n$ can be omitted. Dropping the term $V_\perp \cdot \Sigma^*$ on the right-hand side of the inclusion afterwards analogously produces $V_H \subseteq U_H \cup U_\perp \cdot \Sigma^*$, which was to be shown.

This analogy does not hold for $\leq$-number restrictions, where we need to cope with the right quotient ($\cdot R^{-1}$) in the respective equations: For every $(\leq nR) := H \in \mathcal{N}_{\leq}$, we obtain $U_H \cup U_\perp \cdot \Sigma^* \cdot R^{-1} \supseteq V_H \cup V_\perp \cdot \Sigma^* \cdot R^{-1}$. We may drop the expression $V_\perp \cdot \Sigma^* \cdot R^{-1}$ on the right-hand side of the inclusion. Furthermore, as stated in [BKBM99], $U \cdot \Sigma^* \cdot R^{-1}$ equals $U \cdot \Sigma^* \cup U \cdot R^{-1}$ for every finite language U over Σ and $R \in \Sigma$. Consequently, the inclusion can be simplified to $U_H \cup U_\perp \cdot \Sigma^* \cup U_\perp \cdot R^{-1} \supseteq V_H$, which we wanted to show.

$\triangleright$ $E_C = E_D$: As shown in (2), the reduced normal form of C and D then allows us to infer $U_\perp \cdot \Sigma^* = V_\perp \cdot \Sigma^*$, which yields $U_\perp = V_\perp$, as both languages are prefix free. The characterization of the subsumption furthermore allows us to conclude that $U_H \supset V_H$ for every $H \in \mathcal{H}$. Obviously, C and D cannot agree on all role languages, since this would imply $C \equiv D$, in contradiction to the assumption. Consequently, there is one $H \in \mathcal{H}$ such that $U_H \supset V_H$.

$\blacktriangleright$ 3. ("$\Leftarrow$") In case (b), it is not difficult to verify that the conditions for subsumption stated in Lemma 3.6 are met. Assume case (a). From $U_\perp \succ V_\perp$ we can infer by Lemma 6.4 that $U_\perp \cdot \Sigma^* \supset V_\perp \cdot \Sigma^*$. Since C and D are reduced, this implies $E_C \supset E_D$, matching the first condition for subsumption. As assumed, for every $H \in \mathcal{H} \setminus \mathcal{N}_{\leq}$ it holds that $V_H \subseteq U_H \cup U_\perp \cdot \Sigma^*$. We have already seen in (3) that $U_\perp \cdot \Sigma^*$ equals E_C. Therefore, after adding the language E_D on both sides of the inclusion we have $V_H \cup E_D \subseteq U_H \cup E_C \cup E_D$. Since E_D is a subset of E_C, we obtain $V_H \cup E_D \subseteq U_H \cup E_C$. For $H \in \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\}$, this equals conditions 2 and 3 for the subsumption as stated in Lemma 3.6.

According to Corollary 6.13, for all $(\geq nR) \in \mathcal{N}_{\geq}$ the language $U_{(\geq nR)}$ is equal to the union $\bigcup_{m \geq n} U_{(\geq nR)}$, so that the inclusion $V_H \cup E_D \subseteq U_H \cup E_C$ can be expanded to $\bigcup_{m \geq n} V_{(\geq mR)} \cup E_D \subseteq \bigcup_{m \geq n} U_{(\geq mR)} \cup E_C$, which meets condition 4 for the subsumption.

For $(\leq nR) \in \mathcal{N}_{\leq}$, we have assumed $V_{(\leq nR)} \subseteq U_{(\leq nR)} \cup U_\perp \cdot \Sigma^* \cup U_\perp \cdot R^{-1}$. As mentioned above for the reverse direction of (3), we can replace $U_\perp \cdot \Sigma^* \cup U_\perp \cdot R^{-1}$ by $U_\perp \cdot \Sigma^* \cdot R^{-1}$, which is equal to $E_C \cdot R^{-1}$. Following a similar line as for the $\geq$-number restrictions, $E_D \cdot R^{-1}$ is added on both sides of the inclusion, yielding $V_{(\leq nR)} \cup E_D \cdot R^{-1} \subseteq U_{(\leq nR)} \cup E_C \cdot R^{-1} \cup E_D \cdot R^{-1}$. As E_C is a superset of E_D and as also both languages are of the form $L \cdot \Sigma^*$ for some finite language L, it is easy to see that $E_C \cdot R^{-1}$ is a superset of $E_C \cdot R^{-1}$ for every $R \in \Sigma$. The inclusion therefore simplifies to $V_{(\leq nR)} \cup E_D \cdot R^{-1} \subseteq U_{(\leq nR)} \cup E_C \cdot R^{-1}$. Exploiting Corollary 6.13, the languages $U_{(\leq nR)}$ and $V_{(\leq nR)}$ can be replaced by the respective unions over all $m \leq n$, thus matching condition 5 of the subsumption conditions of Lemma 3.6. Consequently, all conditions for subsumption are met. We obtain strict subsumption, because (2) would otherwise imply $U_\perp = V_\perp$, contradicting $U_\perp \succ V_\perp$. $\square$

The characterization of strict subsumption in (3) can be expressed in a slightly stricter form, stating for case (a) that $U_\perp \succ V_\perp$ and $V_H \subseteq U_H \cup (U_\perp \cdot \Sigma^* \setminus V_\perp \cdot \Sigma^*)$ for all $H \in \mathcal{H} \setminus \mathcal{N}_{\leq}$ and $V_H \subseteq U_H \cup (U_\perp \setminus V_\perp) \cdot R^{-1} \cup (U_\perp \cdot \Sigma^* \setminus (V_\perp \cdot \Sigma^* \cup V_\perp \cdot R^{-1}))$ for all $(\leq nR) := H \in \mathcal{N}_{\leq}$. Nevertheless, this will not be required here.

Observe, however, that the characterizations of equivalence and subsumption derived for $\mathcal{FL}_\perp$ and $\mathcal{FL}_\neg$ are of similar structure. The only difference regards $\leq$-number restrictions in the characterization of the subsumption. Therefore, one advantage of the normal forms proposed in this section is the ability to exploit structural similarities between the logics, allowing a uniform handling.

6.3 The algorithm

We are now prepared to introduce the actual algorithm for solving matching problems modulo equivalence with non-strict side conditions. Its idea is to simulate solving matching

problems with side conditions in a single step by solving a series of matching problems without them in several steps. Hence, this approach is based on the ability to solve matching problems without side conditions in a logic $\mathcal{L}$. An appropriate algorithm for this task has been proposed under the name $match_{\mathcal{L}}$ in Lemma 4.29. The new algorithm is specified in an imperative fashion by the following definition.

Definition 6.20 Algorithm
Let $\mathcal{L}$ be a logic in $\{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be a (U, V, W)-labelled $\mathcal{L}$-matching problem modulo equivalence with non-strict side conditions. The algorithm $\mathcal{A}_{\mathcal{L}}(P)$ is defined as follows:

1. $t := 0,\ \sigma_0 := match_{\mathcal{L}}\{C \equiv^? D\}$
2. $\sigma_{t+1} := match_{\mathcal{L}}(\{C \equiv^? D\} \cup \{\sigma_t(X_j) \sqsubseteq^? E_j | 1 \leq j \leq n\})$
3. If σ_{t+1} is undefined: return "no solution".
   If $\sigma_t \equiv \sigma_{t+1}$: return σ_t.
   Otherwise: $t := t + 1$, continue at (2). □

At first glance, we find that upon input P the algorithm starts by merely ignoring the side conditions included in the matching problem and solves it without them. Thus, it yields a first result σ_0, which might be too specific for P. By solving certain matching problems, the algorithm then improves the intermediate solution σ_0 iteratively until a fixed point is reached in respect to equivalence. Taking a closer look, we will see that the fixed point iteration exhibits four underlying properties:

- No possible solution to the input matching problem P is more specific than the initial substitution σ_0.
- The same holds for every subsequent substitution σ_t.
- Every substitution σ_{t+1} is more general than its respective predecessor σ_t.
- If two consecutive substitutions σ_t and σ_{t+1} are equivalent, then they are valid solutions of P.

Before dealing with the question in terms of a formal proof, let us discuss intuitively why the above properties hold. The substitution σ_0 lies below every possible solution to the input matching problem, since $match_{\mathcal{L}}$ by definition always returns the least solution. Thus, for every solution σ_L of P and for every $j \in \{1, \ldots, \ell\}$ it holds that $\sigma_0(X_j) \sqsubseteq \sigma_L(X_j)$, which in turn implies that $\sigma_0(X_j) \sqsubseteq \sigma_L(E_j)$. Consequently, every substitution σ_t produced in step 2 of the iteration also lies below every solution to P, i.e. $\sigma_t \sqsubseteq \sigma_L$. Since additional constraints are included, it is easy to see that σ_1 is never more specific than σ_0, i.e. $\sigma_0 \sqsubseteq \sigma_1$. Consequently, we find by induction that $\sigma_t \sqsubseteq \sigma_{t+1}$ for every t, meaning that the substitutions produced by the iteration become more general in every step. In case of equivalence, i.e. $\sigma_t \equiv \sigma_{t+1}$, it holds that $C \equiv \sigma_t(D)$ as well as $\sigma_t(X_j) \sqsubseteq \sigma_t(E_j)$ for every j, implying that σ_t solves P.
In the next section we will find that proving the algorithm to be correct and complete is particularly simple when following the above lines. Nevertheless, it is not yet clear whether the iteration always reaches a solution in a finite number of steps provided there exists one. Analogously, we have to ascertain that the algorithm returns "no solution", whenever there exists no solution to the input matching problem. This issue, i.e. the question of termination, is addressed in Section 6.5.

6.4 Correctness and completeness

Here two properties have to be shown. Firstly, if our algorithm terminates with a certain solution, then this solution solves the input matching problem. Secondly, if it terminates without finding a solution, then in fact no solution exists. To show this, we will begin by formally proving the properties discussed at the end of the previous section.

Lemma 6.21 Correctness and completeness
Let $\mathcal{L}$ be a logic in $\{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be a (U, V, W)-labelled $\mathcal{L}$-matching problem modulo equivalence with non-strict side conditions. Then:

1. For every solution σ_L to the input matching problem P and for every substitution σ_t occuring during the execution of the algorithm $\mathcal{A}_\mathcal{L}(P)$, it holds that $\sigma_t \sqsubseteq \sigma_L$.
2. For all substitutions σ_t and σ_{t+1} occuring during the execution of the algorithm $\mathcal{A}_\mathcal{L}(P)$ it holds that: $\sigma_t \sqsubseteq \sigma_{t+1}$.
3. If $\mathcal{A}_\mathcal{L}(P)$ returns a substitution σ, then σ solves the input matching problem P.
4. If $\mathcal{A}_\mathcal{L}(P)$ returns "no solution", then the input matching problem P has no solution.

Proof.
▶ 1. Proof by induction over the number of iterations t the algorithm takes.

▷ $t = 0$: We know from the properties of $match_\mathcal{L}$ that then σ_t is the least solution for the matching problem $C \equiv^? D$. Every solution σ_L to P especially solves $C \equiv^? D$ with respect to $(\sqsubseteq)$. Therefore, we always obtain $\sigma_0 \sqsubseteq \sigma_L$.

▷ $t > 0$: Assume that σ_{t+1} exists. By the indiction hypothesis we know that $\sigma_t \sqsubseteq \sigma_L$. Thus, for all $j \in \{1, \ldots, \ell\}$ we have $\sigma_t(X_j) \sqsubseteq \sigma_L(X_j)$. We now show that this implies

$$match_\mathcal{L}(\{C \equiv^? D\} \cup \{\sigma_t(X_j) \sqsubseteq^? E_j | 1 \leq j \leq n\}) \\ \sqsubseteq match_\mathcal{L}(\{C \equiv^? D\} \cup \{\sigma_L(X_j) \sqsubseteq^? E_j | 1 \leq j \leq n\}) \ .$$

Every solution to the—more general—matching problem of the right-hand side is especially a solution to the matching problem of the left-hand side of the equation. As $match_\mathcal{L}$ always computes minimal solutions for both sides, the above conclusion is valid. Applying the definition of $\mathcal{A}_\mathcal{L}$, the above yields $\sigma_{t+1} \sqsubseteq \sigma_L$, which was to be shown.

▶ 2. Analogous to (1). Due to the properties of $match_\mathcal{L}$, we similarly have $\sigma_0 \sqsubseteq \sigma_1$. The induction is identical to (1): If $\sigma_{t-1} \sqsubseteq \sigma_t$ then we can infer by the same scheme as above that $\sigma_t \sqsubseteq \sigma_{t+1}$.

▶ 3. By definition of the algorithm, $C \equiv \sigma_t(D)$ for every substitution σ_t. Hence, σ_t is a valid solution to the matching problem without side conditions. It remains to be shown that the final solution meets the side conditions as well. Thus, assume that $\mathcal{A}_\mathcal{L}(P) = \sigma_t$ for some nonnegative integer t. Then by definition, $\sigma_t \equiv \sigma_{t+1}$, which implies that for every $j \in \{1, \ldots, \ell\}$ we have $\sigma_t(X_j) \sqsubseteq \sigma_{t+1}(E_j) \equiv \sigma_t(E_j)$. The subsumption holds by definition of σ_{t+1}, the equivalence by the assumption above. Thus, the side conditions are met.

▶ 4. If $match_\mathcal{L}$ fails in step 1 of the algorithm then no solution exists for the matching problem without side conditions. This obviously implies that there is no solution for the matching problem with side conditions as well.

If the algorithm fails in step 2 for some positive integer t, then the matching problem $C \equiv^? D$ is solvable, but there is a $j \in \{1, \ldots, \ell\}$ such that $\sigma_t(X_j) \sqsubseteq^? E_j$ does not have a solution. This implies that $\sigma_t(X_j)$ is an assignment too general for the i-th side condition. Taking into account the results of (1), this consequently applies to any possible solution σ_L to the original matching problem with side conditions. □

As an immediate consequence of the above lemma, we can derive an important property of the algorithm $\mathcal{A}_\mathcal{L}$, which is stated in the following corollary: The solutions returned by $\mathcal{A}_\mathcal{L}$ are minimal.

Corollary 6.22 Minimal solutions
Let $\mathcal{L} \in \{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let P be an $\mathcal{L}$-matching problem modulo equivalence with non-strict side conditions. Then, if $\mathcal{A}_\mathcal{L}(P)$ returns a solution σ, this solution is the least solution to P in respect to subsumption.

We have seen in the previous lemma that no substitution σ_t is more general than any solution σ_L to P. Provided the fixed-point iteration terminates it is therefore obvious that the obtained solution is minimal. Observe that the main argument in the above proof of correctness and completeness is the property of the matching algorithm $match_\mathcal{L}$ to produce minimal solutions in respect to subsumption. Reduced normal forms and their characteristics have not been required so far.

6.5 Termination

In this section, we will show that the algorithm $\mathcal{A}_\mathcal{L}$ terminates in polynomial time in the size of any input matching problem. The objective is to avoid giving analogous proofs of termination iteratively for every logic. Therefore, we identify three conditions as sufficient to permit a general proof of termination. We then only have to ensure that these conditions hold in all four logics.

Two steps are necessary in preparation. Firstly, a uniform notation is needed to denote all role languages produced by the algorithm during the fixed point iteration. Secondly, we will assume that every concept description occuring in the algorithm is in reduced normal form. Naturally, it has to be clarified beforehand why such an assumption can be made without loss of generality. But let us first take care of the notation problem. It was shown in Lemma 4.29, that $match_\mathcal{L}$ does not introduce new atomic concepts or number restrictions for its solution. Moreover, the solutions are presented in $\mathcal{FL}_0$-normal form. For a uniform notation, we therefore only need to specify three things. Firstly, the set of all indices t occuring during the computation of $\mathcal{A}_\mathcal{L}(P)$; secondly; the set of all concept names and role restrictions occuring in the input problem; and thirdly, an appropriate notation for the $\mathcal{FL}_0$-normal forms of all substitutions:

Definition 6.23 Notation
Let $\mathcal{L}$ be a logic in $\{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be a (U, V, W)-labelled $\mathcal{L}$-matching problem modulo equivalence with non-strict side conditions. Upon Input P, the algorithm $\mathcal{A}_\mathcal{L}$ generates substitutions σ_t, where $t \in \mathbb{N}$.

- Denote by $T(\mathcal{A}_\mathcal{L}, P)$ the set of indices t occurring during the fixed point iteration for $\mathcal{A}_\mathcal{L}(P)$. Formally, $T(\mathcal{A}_\mathcal{L}, P) := \mathbb{N}$ iff $\mathcal{A}_\mathcal{L}$ upon input P does not terminate, $T(\mathcal{A}_\mathcal{L}, P) := \{0, \ldots, t+1\}$ iff it terminates in step 3 with σ_t as solution and $T(\mathcal{A}_\mathcal{L}, P) := \{0, \ldots, t\}$ iff σ_{t+1} is undefined.
- Define $\mathcal{H}$ as the set of all concept names and number restricions occurring in the input matching problem, i.e. for $\mathcal{ALN}$, define $\mathcal{H} := \{\bot\} \cup \mathcal{C} \cup \{\neg A | A \in \mathcal{C}\} \cup \mathcal{N}_\leq \cup \mathcal{N}_\geq$. For $\mathcal{FL}_\neg$, we require $\mathcal{N}_\leq$ and $\mathcal{N}_\geq$ to be empty. For $\mathcal{FL}_\bot$, negated atomic concepts are omitted as well. For $\mathcal{FL}_0$, we simply have $\mathcal{H} = \mathcal{C}$.
- For every occurring index $t \in T(\mathcal{A}_\mathcal{L}, P)$ and for every $j \in \{1, \ldots, \ell\}$, denote $\sigma_t(X_j)$ in U_t-labelled $\mathcal{FL}_0$-normal form. □

With the above notation, every concept description $\sigma_t(X_j)$ occurring in the the fixed point iteration can be represented by a set of role languages $\{U_{t,j,H}|H \in \mathcal{H}\}$.

We want to identify certain conditions in order to simplify the proof of termination of algorithm $\mathcal{A}_\mathcal{L}$ for all four cases of $\mathcal{L}$. For these conditions to hold, it is necessary that every concept description occurring during the execution of the $\mathcal{A}_\mathcal{L}$ is in reduced normal form, i.e. for every input matching problem $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j|1 \leq j \leq \ell\})$, for every $t \in T(\mathcal{A}_\mathcal{L}, P)$ and for every $j \in \{1, \ldots, \ell\}$, C as well as $\sigma_t(X_j)$ are in reduced normal form. The definition of $\mathcal{A}_\mathcal{L}$, however, does not include this. Why can such an assumption be made without loss of generality? We know from Lemmata 6.8, 6.11, and 6.19 that transforming a concept description into reduced normal form conserves equivalence. Thus, the results computed by $match_\mathcal{L}$ in steps 1 and 2 of the algorithm are not altered in respect to equivalence by assuming reduced normal forms. The termination criterion in step 3 of the algorithm also refers only to equivalence and not to equality. Consequently, the reduced version of the algorithm terminates if and only if the non-reduced version does. Hence, we may safely assume reduced normal forms.

We are now prepared to introduce the termination conditions.

Definition 6.24 Termination conditions
Let $\mathcal{L} \in \{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let P be an $\mathcal{L}$-matching problem modulo equivalence with non-strict side conditions as introduced in Definition 3.17. Define the following conditions for the matching algorithm $\mathcal{A}_\mathcal{L}$.

1. *Representation condition*
   $\mathcal{A}_\mathcal{L}$ operates on a fixed set of role languages for the representation of the substitutions $\sigma_t(X_j)$ occuring during the execution of $\mathcal{A}_\mathcal{L}(P)$. Therefore, no new (negated) atomic concepts or role restrictions are introduced.

2. *Suffix condition*
   For every $H \in \mathcal{H}$ there exists a role language M_H of polynomial size in the size of P such that for all $t \in T$ and $j \in \{1, \ldots, \ell\}$ the role languages $U_{t,j,H}$ contain only suffixes of words from M_H.

3. *Deletion condition*
   If a word occuring in a role language $U_{t,j,H}$ assigned by σ_t is missing in the role language $U_{t+1,j,H}$ assigned by σ_{t+1}, then this word is missing in every role language $U_{t',j,H}$ assigned by further substitutions $\sigma_{t'}$ with $t' > t$. □

Observe that the representation condition in our case is only relevant for $\mathcal{ALN}$. In the sublanguages the problem does not arise. For $\mathcal{ALN}$, however, it is stated in Lemma 4.29 that actually already $match_\mathcal{ALN}$ respects the representation condition. This condition is nevertheless mentioned explicitly here, because we want to emphasize that it is an essential prerequisite for the proof of correctess. The next subsection provides a general proof of termination, presupposing the validity of the termination conditions. The last four subsections are devoted to verifying these conditions in our four logics.

6.5.1 General result

We have to show that by virtue of the termination conditions we can prove termination of the algorithm $\mathcal{A}_\mathcal{L}$ simultaneously for all four cases of $\mathcal{L}$. The following lemma performs this step.

Lemma 6.25 Termination
Let $\mathcal{L}$ be a logic in $\{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be a (U, V, W)-labelled $\mathcal{L}$-matching problem modulo equivalence with non-strict side conditions. Then $\mathcal{A}_\mathcal{L}(P)$ terminates in polynomial time in the size of P.

Proof.
It is sufficient to show that two properties hold for $\mathcal{A}_\mathcal{L}$. Firstly, the number t of steps the execution of $\mathcal{A}_\mathcal{L}(P)$ takes is polynomially limited in the size of P; secondly, the time required for a single step is polynomially bounded in the size of P as well.

▷ Limit for t: It is shown in Lemma 6.21 that $\sigma_t \sqsubseteq \sigma_{t+1}$ for every $t \in T(\mathcal{A}_\mathcal{L}, P)$. Since the fixed point iteration in $\mathcal{A}_\mathcal{L}$ terminates in case $\sigma_t \equiv \sigma_{t+1}$, we have $\sigma_t \sqsubset \sigma_{t+1}$ for every t as long as the iteration does not terminate. The strict subsumption of the substitutions implies that for every t there is an $j \in \{1, \ldots, \ell\}$ such that $\sigma_t(X_j) \sqsubset \sigma_{t+1}(X_j)$.

Due to the characterization of strict subsumption for reduced normal forms (Lemmata 6.8, 6.11, and 6.19), this implies that there is an $H \in \mathcal{H}$, such that at least one word in the role language $U_{t,j,H}$ occurring in $\sigma_t(X_j)$ is deleted at the transition to $U_{t+1,j,H}$ occurring in $\sigma_{t+1}(X_j)$. Now on the one hand, the deletion condition guarantees that no word can reappear once it has been deleted at such a transition. On the other hand, the suffix condition ensures for every $H \in \mathcal{H}$ that every word deleted from a role language $U_{t,j,H}$ is a suffix of some word in the polynomially large language M_H.

We now have obtained three facts sufficient to prove the existence of a polynomial upper bound for t. Firstly, at the transition from t to $t+1$ some word has to be deleted from some role language; secondly, once deleted, no word will reappear later on; and thirdly, the choice of words to delete is polynomially limited and independent of t. Consequently, t cannot exceed the sum of the number of all suffixes of the words of all role languages M_H. $\mathcal{H}$ is immediately limited by the input matching problem. Furthermore, M_H is required by the suffix condition to be polynomial in the size of the input problem. Finally, the number of suffixes of a word is quadratic in the length of the word. We therefore end up with a polynomial upper bound for t.

▷ Limit for a single step: The reduced normal form of concept descriptions can be computed in polynomial time. According to the properties of $match^\downarrow_\mathcal{L}$, we can solve $\mathcal{L}$-matching problems in polynomial time in the size of the input. The matching problem solved in every single step in $\mathcal{A}_\mathcal{L}(P)$ is always of polynomial size in the size of P. This fact is guaranteed by the representation condition and the suffix condition which we have shown valid for every logic $\mathcal{L}$ considered here. Therefore, a single step costs only polynomial time. □

It should be stressed that, as a consequence of the above result, the algorithm $\mathcal{A}_\mathcal{L}$ itself does terminate correctly even without the assumption of reduced normal forms. Only the proof of termination is simplified significantly by this additional requirement. As an immediate consequence of the termination of the algorithm $\mathcal{A}_\mathcal{L}$ we obtain the following corollary.

Corollary 6.26 Minimal solutions
Solvable matching problems modulo equivalence with non-strict side conditions in $\mathcal{FL}_0$, $\mathcal{FL}_\bot$, $\mathcal{FL}_\neg$, and $\mathcal{ALN}$ have a minimal solution in respect to subsumption.

This claim holds, because the algorithm terminates successfully if and only if the input matching problem has a solution and the solution returned is minimal in respect to subsumption. Let us now verify the validity of the termination conditions in our four logics.

6.5.2 Termination conditions in $\mathcal{FL}_0$

For $\mathcal{FL}_0$, proving the conditions is particularly easy because of the simple characterization of the subsumption obtained by Lemma 3.7. The characterization immediately implies the desired results, as shown in the next lemma.

Lemma 6.27 Termination in $\mathcal{FL}_0$
Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be an $\mathcal{FL}_0$-matching problem modulo equivalence with non-strict side conditions in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Then the termination conditions introduced in Definition 6.24 hold for $\mathcal{A}_{\mathcal{FL}_0}(P)$.

Proof.
▷ Representation: We have seen in Lemma 4.29 that $match_{\mathcal{L}}$ already meets the representation condition. As a consequence, it also holds for $\mathcal{A}_{\mathcal{FL}_0}$.

▷ Suffix: Due to Lemma 6.21, for every $t \in T(\mathcal{A}_{\mathcal{FL}_0}, P)$ occuring during the execution of $\mathcal{A}_{\mathcal{FL}_0}(P)$ it holds that: $\sigma_t \sqsubseteq \sigma_{t+1}$. In $\mathcal{FL}_0$, this implies $U_{t,j,H} \supseteq U_{t+1,j,H}$ for every $j \in \{1, \ldots, \ell\}$ and for every $H \in \mathcal{H}$. Thus, we can infer $U_{t,j,H} \subseteq U_{0,j,H}$ for every j and H.
The role language $U_{0,j,H}$ occurs in step 1 of the algorithm in the soluion to the matching problem $C \equiv^? D$. In [BKBM99], it is shown that for the $\mathcal{FL}_0$-normal form of the solution to $C \equiv^? D$ it holds that $U_{0,j,H}$ equals $\bigcup_{w \in W_j} w^{-1}(U_H)$ for every j and H. Therefore, every role language $U_{t,j,H}$ contains only suffixes of U_H. Since U_H is part of the input matching problem P, the set of its suffixes serve as an appropriate upper bound. Thus, the suffix condition is met by chosing M_H as the set of all suffixes of words in U_H.

▷ Deletion: We have seen above that $\sigma_t \sqsubseteq \sigma_{t+1}$ for every t, j, and H implies a superset relation $U_{t,j,H} \supseteq U_{t+1,j,H}$. This relation entails that words cannot reappear after they have been deleted at the transition from $U_{t,j,H}$ to $U_{t+1,j,H}$. □

6.5.3 Termination conditions in $\mathcal{FL}_\bot$

Reduced normal forms were not necessary for the termination conditions to hold in $\mathcal{FL}_0$. We will see for $\mathcal{FL}_\bot$, $\mathcal{FL}_\neg$, and $\mathcal{ALN}$ that they are essential for the proof of the suffix- and deletion property. We shall also see that the bottom-concept contributes most to the greater effort necessary to prove the termination conditions. At first, the validity of the suffix condition is shown. The idea is to use the solution languages introduced in Definition 4.2 to derive a recursive relationship with respect to t between the role languages occuring in consecutive substitutions σ_t. We can then infer the desired properties from σ_0 upwards by induction.

Lemma 6.28 Suffix condition in $\mathcal{FL}_\bot$
Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be an $\mathcal{FL}_\bot$-matching problem with non-strict side conditions in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Denote the role languages occuring during the execution of $\mathcal{A}_{\mathcal{FL}_\bot}(P)$ as specified in Definition 6.23. Then for all $t \in T(\mathcal{A}_{\mathcal{FL}_\bot}, P)$ and for all $j \in \{1, \ldots, \ell\}$ it holds that

1. $U_{t,j,\bot}$ contains only suffixes of $U_\bot$.
2. $U_{t,j,H}$ contains only suffixes of U_H for all $H \in \mathcal{H}$.

Proof.
▶ 1. When performing step t of the algorithm $\mathcal{A}_{\mathcal{FL}_\bot}(P)$, the following system of matching problems must be solved.

$$\forall U_\bot.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall U_A.A \equiv^? \forall V_\bot.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall V_A.A \sqcap \bigsqcap_j \forall W_j.X_j$$

$$\forall U_{t,j,\bot}.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall U_{t,j,A}.A \sqsubseteq^? \forall V_{j,0}.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall V_{j,i}.A \sqcap \bigsqcap_{j'} \forall W_{j,j'}.X_{j'}$$

where the second line represents one equation for every $j \in \{1, \ldots, \ell\}$. As stated in Lemma 3.15, this system can be combined into a single matching problem modulo equivalence with little difficulty. The exact strategy is omitted here, but its idea has been illustrated in Example 3.16. For the resulting matching problem, setting up the solvability equations proposed in Definition 4.1 and applying Lemma 4.2, we yield the following solution language for the bottom-concept.

$$U_{t+1,j,\perp}\cdot\Sigma^* = \bigcap_{w\in W_j} w^{-1}(U_\perp\cdot\Sigma^*) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,\perp}\cdot\Sigma^*) \qquad (*)$$

Due to the notation introduced for the solutions σ_t, here $U_{t+1,j,\perp}\cdot\Sigma^*$ takes the place of $\widehat{L}_{j,\perp}$ used in Lemma 4.2 to denote the solution language for the $\perp$-concept. We have to show that the $U_{t+1,j,\perp}$ contains only suffixes of $U_\perp$.

According to Lemma 6.5, for every finite language U and for every word w there exists a finite prefix free language L such that firstly, $L\cdot\Sigma^* = w^{-1}(U\cdot\Sigma^*)$; and secondly, L contains only suffixes of U. Using this result we now show the proposition for $U_{t,j,\perp}$ by induction over the number of steps t the algorithm $\mathcal{A}_{\mathcal{FL}_\perp}(P)$ takes.

$\triangleright$ $(t = 0)$: According to equation $(*)$, it holds that

$$U_{0,j,\perp}\cdot\Sigma^* = \bigcap_{w\in W_j} w^{-1}(U_\perp\cdot\Sigma^*). \qquad (*')$$

At first, we show that the suffix condition does not get lost when intersecting languages of the form $L\cdot\Sigma^*$ having that property. It is shown in [BKBM99] that for finite languages L and L' the intersection $L\cdot\Sigma^* \cap L'\cdot\Sigma^*$ is equal to $((L \cap L'\cdot\Sigma^*) \cup (L' \cap L\cdot\Sigma^*))\cdot\Sigma^*$.

Obviously, $(L \cap L'\cdot\Sigma^*) \cup (L' \cap L\cdot\Sigma^*)$ is a subset of the union $L \cup L'$. This implies that the intersection $L\cdot\Sigma^* \cap L'\cdot\Sigma^*$ can be represented as $L''\cdot\Sigma^*$ such that every element of L'' comes from L or from L'.

Because of Lemma 6.5, it holds for every $j \in \{1, \ldots, \ell\}$ and for every $w \in W_j$ that the language $w^{-1}(U_\perp\cdot\Sigma^*)$ can be represented as $L\cdot\Sigma^*$, where L contains only suffixes of $U_\perp$. We have just seen that the suffix condition is respected by the intersection. Thus, the entire right-hand side of equation $(*')$ is of the form $L\cdot\Sigma^*$, where L contains only suffixes of $U_\perp$. $pf(L)$ is a subset of L and therefore contains only suffixes as well. $pf(L)\cdot\Sigma^*$ also represents the right-hand side of $(*')$, as we know from Lemma 6.4. From the definition of reduced normal forms in $\mathcal{FL}_\perp$ we also know that $U_{0,j,0}$ is prefix free. Lemma 6.4 now implies that $U_{0,j,0}$ is equal to $pf(L)$, completing our argument.

$\triangleright$ $(t > 0)$: Due to induction, we may assume that all role languages on the right-hand side of equation $(*)$ contain only suffixes of $U_\perp$. Analogous to the argument for the case $t = 0$, the suffix property is valid for $U_{t+1,j,\perp}$ as well.

$\blacktriangleright$ 2. Consider $U_{t,j,H}$ for an arbitrary $H \in \mathcal{H}$. Starting again with the system of matching equations proposed in (1) and taking into account the definition of the solution languages

in Lemma 4.29, we obtain the following result for $U_{t,j,H}$.

$$\begin{aligned}
U_{t+1,j,H} &= \bigcap_{w\in W_j} w^{-1}(U_h \cup U_\perp\cdot\Sigma^*) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,H} \cup U_{t,j,\perp}\cdot\Sigma^*) \\
&\quad \setminus U_{t+1,j,\perp}\cdot\Sigma^* \\
&= \underbrace{\bigcap_{w\in W_j} w^{-1}(U_h \cup U_\perp\cdot\Sigma^*) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,H} \cup U_{t,j,\perp}\cdot\Sigma^*)}_{M_1} \\
&\quad \setminus \underbrace{\bigcap_{w\in W_j} w^{-1}(U_\perp\cdot\Sigma^*) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,\perp}\cdot\Sigma^*)}_{M_2} \\
&\overset{!}{\subseteq} \bigcup_{w\in W_j} w^{-1}(U_h) \cup \bigcup_{j'} \bigcup_{w\in W_{j,j'}} w^{-1}(U_{t,j,H})
\end{aligned}$$

The equality to $M_1 \setminus M_2$ is obtained by replacing $U_{t+1,j,\perp}\cdot\Sigma^*$ with the right-hand side of equation $(*)$. The last step in the above sequence remains to be shown. Consider an arbitrary word v in $U_{t+1,j,H} = M_1 \setminus M_2$. Since v is not an element of M_2, there exists a word $w \in W_j$ or a word $w' \in W_{j,j'}$ such that v is no element of $w^{-1}(U_\perp\cdot\Sigma^*)$ or no element of $w^{-1}(U_{t,j,\perp}\cdot\Sigma^*)$. Assume the first case, i.e. $v \notin w^{-1}(U_\perp\cdot\Sigma^*)$. As v is an element of M_1, obviously $v \in w^{-1}(U_h \cup U_\perp\cdot\Sigma^*)$, which implies $v \in w^{-1}(U_h)$. Thus, v is a suffix of a word in U_h. The second case is analogous, yielding that v is a prefix of a word in $U_{t,j,H}$. Thus, the inclusion claimed above holds.

Since U_H and all $U_{t,j,H}$ are finite languages, it is not difficult to see that the left quotients $w^{-1}(U_H)$ and $w^{-1}(U_{t,j,H})$ for every word w only contain suffixes of U_H and $U_{t,j,H}$ respectively. We still have to ensure that the suffix condition is respected by the union. This can be shown inductively similar to the proof seen in (1) for the intersection. In case of the union, however, the induction argument is by far simpler, since for finite languages L, L' the union $L\cdot\Sigma^* \cup L'\cdot\Sigma^*$ is equal to $(L \cup L')\cdot\Sigma^*$. □

For the proof of the deletion condition, the characterization of the subsumption for reduced normal forms can be utilized to rule out words reappearing after being deleted. A subsumption argument, of course, can only be used since we know from the proof of correctess, that the solutions σ_t in fact are subsumed by its respective successors σ_{t+1}.

Lemma 6.29 Deletion condition in $\mathcal{FL}_\perp$
Include P and $\mathcal{A}_{\mathcal{FL}_\perp}(P)$ from above. Again, we refer to the notation introduced in Definition 6.23 for the execution of the algorithm. Then $\mathcal{A}_{\mathcal{FL}_\perp}(P)$ meets the deletion condition specified in Definition 6.24 for all occurring role languages.

Proof.
We first prove the deletion condition for role languages referring to the $\perp$-concept and then consider those referring to atomic concepts $A \in \mathcal{C}$.
▷ $\perp$-concept: Assume that contrary to our claim a word w can reappear for greater values of t after being deleted from a role language at a certain point during the execution of the algorithm. Thus, assume for $w \in \Sigma^*$ that $w \in U_{t,j,\perp}$ and $w \notin U_{t',j,\perp}$ but finally $w \in U_{t'+1,j,\perp}$ for some $j \in \{1,\ldots,\ell\}$ and for nonnegative integers $t < t' \in T$.

We know from Lemma 6.21 that $\sigma_t \sqsubseteq \sigma_{t'} \sqsubseteq \sigma_{t'+1}$. As all substitutions are reduced we further know due to our assumption, that $\sigma_t(X_j) \not\equiv \sigma_{t'}(X_j) \not\equiv \sigma_{t'+1}(X_j)$. From this we can infer by means of Lemma 6.8 that $U_{t,j,\perp} \succ U_{t',j,\perp} \succ U_{t'+1,j,\perp}$.

We have assumed that $w \in U_{t'+1,j,\bot}$. The above relation then for $U_{t',j,\bot}$ demands that $U_{t',j,\bot}$ contains a prefix w' of w. As w is no element of $U_{t',j,\bot}$, this is a nontrivial prefix. Similarly we find that $U_{t,j,\bot}$ contains a prefix of w' or w' itself. The language $U_{t,j,\bot}$, however, initially was assumed to contain w as well, yielding a contradiction to $U_{t,j,\bot}$ being prefix free.

$\triangleright$ A-concept: Assume similarly for a word $w \in \Sigma^*$ that $w \in U_{t,j,H}$ and $w \notin U_{t',j,H}$ but finally $w \in U_{t'+1,j,H}$ for some $j \in \{1,\ldots,\ell\}$, for $H \in \mathcal{H}$, and for nonnegative integers $t < t' \in T$. Since $\sigma_t \sqsubseteq \sigma_{t'} \sqsubseteq \sigma_{t'+1}$ and as also all substitutions are reduced we obtain as a consequence of lemma 6.8:

$$U_{t,j,H} \,\dot\cup\, U_{t,j,\bot}\cdot\Sigma^* \;\supseteq\; U_{t',j,H} \,\dot\cup\, U_{t',j,\bot}\cdot\Sigma^* \;\supseteq\; U_{t'+1,j,H} \,\dot\cup\, U_{t'+1,j,\bot}\cdot\Sigma^* \;.$$

We have assumed that $w \in U_{t'+1,j,k}$. Since w is no element of $U_{t',j,k}$, the subset relation implies that $w \in U_{t',j,\bot}\cdot\Sigma^*$. From the characterization of the subsumption we know that $U_{t,j,\bot}\cdot\Sigma^* \supseteq U_{t',j,\bot}\cdot\Sigma^*$, which in our case implies $w \in U_{t,j,\bot}\cdot\Sigma^*$. This contradicts the disjointedness of the union with $U_{t,j,H}$, which was shown in Lemma 6.8. $\square$

6.5.4 Termination conditions in $\mathcal{FL}_\neg$

For $\mathcal{FL}_\neg$, a separate proof of termination is omitted, because we can exploit the analogy to $\mathcal{FL}_\bot$. Verifying the termination conditions again yields a positive result, which is stated below without proof.

Lemma 6.30 Termination conditions in $\mathcal{FL}_\neg$
Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \le j \le \ell\})$ be an $\mathcal{FL}_\neg$-matching problem modulo equivalence with non-strict side conditions. Then $\mathcal{A}_{\mathcal{FL}_\neg}(P)$ meets the termination conditions introduced in Definition 6.24.

Let us discuss briefly why we can expect to gain the same result for $\mathcal{FL}_\neg$ in exactly the same way as seen for $\mathcal{FL}_\bot$. The idea is to show that due to the reduced normal form of all substitutions σ_t occurring during the execution of $\mathcal{A}_{\mathcal{FL}_\neg}(P)$, the validity of the termination conditions can be shown analogous to the proof for $\mathcal{FL}_\bot$. Recall that the prerequisites for the existence of a solution in $\mathcal{FL}_\neg$ are stronger than in $\mathcal{FL}_\bot$. Nevertheless, once the matching problem is solvable, the solution assigned by σ_t is syntactically similar to $\mathcal{FL}_\bot$—the only difference being the construct $\widehat{U}$ instead of U. This can be found when comparing Lemma 4.2 and Lemma 4.4, where the solution languages are introduced. In the presence of reduced normal forms the difference between languages of the form $\widehat{U}$ and U disappears, as stated in Corollary 6.10. Furthermore, a comparison of Lemma 6.8 and Lemma 6.11 yields the same characerization of equivalence and subsumption for reduced normal forms in $\mathcal{FL}_\bot$ and $\mathcal{FL}_\neg$. Hence the results obtained for $\mathcal{FL}_\neg$ are analogous to those for $\mathcal{FL}_\bot$.

6.5.5 Termination conditions in $\mathcal{ALN}$

The overall task of solving matching problems in $\mathcal{ALN}$ is significantly more complex than in the preceding logics. However, most of the additional complexity is hidden in the notion of excluding words, which has been studied in depth in [Küs98]. Once we know that sets of excluding words are of the form $L\cdot\Sigma^*$ for some finite language L, we do not need to introduce new ideas to prove the termination conditions. By virtue of the reduced normal forms we again find a situation analogous to $\mathcal{FL}_\bot$, though consisting of considerably larger equations.

Lemma 6.31 Suffix condition in $\mathcal{ALN}$
Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be an $\mathcal{ALN}$-matching problem with non-strict side conditions in (U, V, W)-labelled $\mathcal{FL}_0$-normal form. Denote the role languages occuring during the execution of $\mathcal{A}_{\mathcal{ALN}}(P)$ as specified in Definition 6.23. Then for all $t \in T$ and for all $j \in \{1, \ldots, \ell\}$ it holds that:

1. $U_{t,j,\bot}$ contains only suffixes of $U_\bot$.
2. $U_{t,j,A}$ contains only suffixes of U_A for every $A \in \mathcal{C}$ and $U_{t,j,\neg A}$ contains only suffixes of $U_{\neg A}$ for every $A \in \mathcal{C}$.
3. $U_{t,j,\geq nR}$ contains only suffixes of $U_{(\geq nR)}$ for every $(\geq nR) \in \mathcal{N}_\geq$.
4. $U_{t,j,\leq nR}$ contains only suffixes of $U_{(\leq nR)} \cup U_\bot \cdot R^{-1}$ for every $(\leq nR) \in \mathcal{N}_\leq$.

Proof.
▶ 1. At step t of the algorithm $\mathcal{A}_{\mathcal{ALN}}(P)$, the following system of matching problems has to be solved:

$$\begin{gathered}
\forall U_\bot.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall U_A.A \sqcap \bigsqcap_{A \in \mathcal{C}} \forall U_{\neg A}.\neg A \\
\sqcap \bigsqcap_{(\geq nR) \in \mathcal{N}_\geq} \forall U_{(\geq nR)}.(\geq nR) \sqcap \bigsqcap_{(\leq nR) \in \mathcal{N}_\leq} \forall U_{(\leq nR)}.(\leq nR) \\
\equiv^? \\
\forall V_\bot.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall V_A.A \sqcap \bigsqcap_{A \in \mathcal{C}} \forall V_{\neg A}.\neg A \\
\sqcap \bigsqcap_{(\geq nR) \in \mathcal{N}_\geq} \forall V_{(\geq nR)}.(\geq nR) \sqcap \bigsqcap_{(\leq nR) \in \mathcal{N}_\leq} \forall V_{(\leq nR)}.(\leq nR) \\
\sqcap \bigsqcap_{j=1}^{n} \forall W_j.X_j
\end{gathered}$$

and for every $j \in \{1, \ldots, n\}$:

$$\begin{gathered}
\forall U_{t,j,\bot}.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall U_{t,j,A}.A \sqcap \bigsqcap_{A \in \mathcal{C}} \forall U_{t,j,\neg A}.\neg A \\
\sqcap \bigsqcap_{(\geq nR) \in \mathcal{N}_\geq} \forall U_{t,j,(\geq nR)}.(\geq nR) \sqcap \bigsqcap_{(\leq nR) \in \mathcal{N}_\leq} \forall U_{t,j,(\leq nR)}.(\leq nR) \\
\sqsubseteq^? \\
\forall V_{j,\bot}.\bot \sqcap \bigsqcap_{A \in \mathcal{C}} \forall V_{j,A}.A \sqcap \bigsqcap_{A \in \mathcal{C}} \forall V_{j,\neg A}.\neg A \\
\sqcap \bigsqcap_{(\geq nR) \in \mathcal{N}_\geq} \forall V_{j,(\geq nR)}.(\geq nR) \sqcap \bigsqcap_{(\leq nR) \in \mathcal{N}_\leq} \forall V_{j,(\leq nR)}.(\leq nR) \\
\sqcap \bigsqcap_{j'=1}^{n} \forall W_{j,j'}.X_{j'}
\end{gathered}$$

This system can be combined into a single matching problem modulo equivalence. For the solution to this problem, Lemma 4.7 provides us with appropriate solution languages. Regarding the $\bot$-concept, we obtain the following result for the solution language $U_{t+1,j,\bot}$ assigned by $\sigma_{t+1}(X_j)$:

$$U_{t+1,j,\bot} \cdot \Sigma^* = \bigcap_{w \in W_j} w^{-1}(E_C) \cap \bigcap_{j'} \bigcap_{w \in W_{j,j'}} w^{-1}(E_{t,j,C}) \qquad (*)$$

Again, due to our notation $U_{t+1,j,\bot} \cdot \Sigma^*$ takes the place of $\widehat{L}_{j,\bot}$ as used in Lemma 4.7. Furthermore, E_C denotes the set of C-excluding words and analogously $E_{t,j,C}$ the set of excluding words for the j-th matching problem in the above system of matching problems.

We may assume C to be in reduced normal form. Consequently, it holds that $U_\bot \cdot \Sigma^* = E_C$, as seen in Definition 6.12. As σ_t is also in reduced normal form, we furthermore obtain

that $U_{t,j,\bot}\cdot\Sigma^* = E_{t,j,C}$ for every $t \in T$. In equation $(*)$, we may therefore replace E_C by $U_\bot\cdot\Sigma^*$ and $E_{t,j,C}$ by $U_{t,j,\bot}\cdot\Sigma^*$. This reveals the inductive relation of the role languages:

$$U_{t+1,j,\bot}\cdot\Sigma^* = \bigcap_{w\in W_j} w^{-1}(U_\bot\cdot\Sigma^*) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,\bot}\cdot\Sigma^*) \qquad (*')$$

It is to prove that $U_{t+1,j,\bot}$ contains only suffixes of $U_\bot$. Equation $(*')$ is only a syntactic variant of equation $(*)$ established in Lemma 6.28. As $U_{t+1,j,\bot}$ is prefix free, we can prove the claim exactly following the same pattern as seen for $\mathcal{FL}_\bot$ in Lemma 6.28.

▶ 2. From the system of matching problems introduced in (1), we now derive solutions for role languages of the form $U_{t+1,j,A}$ referring to the atomic concept A in $\sigma_{t+1}(X_j)$. By virtue of Lemma 4.7 we obtain:

$$U_{t+1,j,A} = \bigcap_{w\in W_j} w^{-1}(U_A \cup E_C) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,A} \cup E_{t,j,C}) \setminus U_{t+1,j,\bot}\cdot\Sigma^*$$

Taking into account that $U_\bot\cdot\Sigma^* = E_C$ and that $U_{t,j,\bot}\cdot\Sigma^* = E_{t,j,C}$, we can apply the argument of Lemma 6.28 and replace the expression $U_{t+1,j,\bot}\cdot\Sigma^*$ with the right-hand side of equation $(*')$. Again, we can obtain an upper bound for the resulting expression, yielding that:

$$U_{t+1,j,A} \subseteq \bigcup_{w\in W_j} w^{-1}(U_A) \cup \bigcup_{j'} \bigcup_{w\in W_{j,j'}} w^{-1}(U_{t,j,A}) \ .$$

Because U_A and every $U_{t+1,j,A}$ is finite, it is not difficult to prove that $w^{-1}(U_A)$ and every $w^{-1}(U_{t+1,j,A})$ contain only suffixes of U_A. We know from Lemma 6.28, that this property is respected by the union, thus completing the proof. For role languages $U_{t,j,\neg A}$ referring to negated atomic concepts $\neg A$, exactly the same argument holds.

▶ 3. We already know that σ_t is in reduced normal form for every $t \in T$. Thus, we have for every number restriction $(\geq nR) \in \mathcal{N}_\geq$ that $\bigcup_{m\geq n} U_{t,j,(\geq mR)}$ is equal to $U_{t,j,(\geq nR)}$, i.e. the union can be omitted. The same holds for C, which is in reduced normal form as well. Therefore, the expression $\bigcup_{m\geq n} U_{(\geq mR)}$ similarly can be replaced by $U_{(\geq nR)}$. This observation enables us to simplify the solution language derived from the system of matching problems proposed in (1). By means of Lemma 4.7, we can infer for $U_{t+1,j,(\geq nR)}$ that:

$$\begin{aligned} U_{t+1,j,(\geq nR)} &= \bigcap_{w\in W_j} w^{-1}\Big(\bigcup_{m\geq n} U_{(\geq mR)} \cup E_C\Big) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}\Big(\bigcup_{m\geq n} U_{t,j,(\geq mR)} \cup E_{t,j,C}\Big) \\ &\quad \setminus U_{t+1,j,\bot}\cdot\Sigma^* \\ &= \bigcap_{w\in W_j} w^{-1}(U_{(\geq nR)} \cup E_C) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,(\geq nR)} \cup E_{t,j,C}) \\ &\quad \setminus U_{t+1,j,\bot}\cdot\Sigma^* \end{aligned}$$

We can see that after removing the unions for the number restrictions, the above equation is syntactically identical to the one derived for $A \in \mathcal{C}$ in (2). The rest of the argument therefore is identical to what has been proposed there.

▶ 4. For $(\leq)$-number restrictions, we can again remove the union-operator in the same fashion as done in (3). However, we obtain slightly different results for the solution languages derived from the system of matching problems introduced in (1). For $U_{t+1,j,(\leq nR)}$

we can infer that:

$$\begin{aligned}
U_{t+1,j,(\leq nR)} = & \bigcap_{w\in W_j} w^{-1}\Big(\bigcup_{m\geq n} U_{(\leq mR)} \cup E_C\cdot R^{-1}\Big) \\
& \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}\Big(\bigcup_{m\geq n} U_{t,j,(\leq mR)} \cup E_{t,j,C}\cdot R^{-1}\Big) \\
& \setminus U_{t+1,j,\bot}\cdot\Sigma^* \\
= & \bigcap_{w\in W_j} w^{-1}(U_{(\leq nR)} \cup (U_\bot\cdot\Sigma^*)\cdot R^{-1}) \\
& \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,(\leq nR)} \cup (U_{t,j,\bot}\cdot\Sigma^*)\cdot R^{-1}) \\
& \setminus \underbrace{\left(\bigcap_{w\in W_j} w^{-1}(U_\bot\cdot\Sigma^*) \cap \bigcap_{j'} \bigcap_{w\in W_{j,j'}} w^{-1}(U_{t,j,\bot}\cdot\Sigma^*)\right)}_{=:M_2}
\end{aligned}$$

Observe, that in the second step we could replace E_C by $U_\bot\cdot\Sigma^*$ and $E_{t,j,C}$ by $U_{t,j,\bot}\cdot\Sigma^*$. This replacement is valid because C and σ_t are in reduced normal form. However, the result deviates from the pattern seen in the previous cases of this proof—the right-quotients of $U_\bot\cdot\Sigma^*$ and $U_{t,j,\bot}\cdot\Sigma^*$ occur instead of the original languages. Nevertheless, we can simplify the right quotient thanks to the finiteness of $U_\bot$ and $U_{t,j,\bot}$: $(U_\bot\cdot\Sigma^*)\cdot R^{-1}$ equals $U_\bot\cdot R^{-1}\cup U_\bot\cdot\Sigma^*$ and similarly $(U_{t,j,\bot}\cdot\Sigma^*)\cdot R^{-1}$ can be simplified to $U_{t,j,\bot}\cdot R^{-1}\cup U_{t,j,\bot}\cdot\Sigma^*$ for all t and j. Since after this transformation all right quotients refer to finite languages, we can subtract M_2 and follow the argument familiar from Lemma 6.28. Consequently, we obtain:

$$\begin{aligned}
U_{t+1,j,(\leq nR)} \subseteq & \bigcup_{w\in W_j} w^{-1}(U_{(\leq nR)} \cup U_\bot\cdot R^{-1}) \\
& \cup \bigcup_{j'} \bigcup_{w\in W_{j,j'}} w^{-1}(U_{t,j,(\leq nR)} \cup U_{t,j,\bot}\cdot R^{-1})
\end{aligned}$$

Finally, we can again employ an induction argument to prove that every $U_{t+1,j,(\leq nR)}$ contains only suffixes of $U_{(\leq nR)} \cup U_\bot\cdot R^{-1}$. □

After eliminating the union over number restrictions and the right-quotient for $(\leq)$-number restrictions in the above equations, the resulting situation appeared very similar to the analogous problems for $\mathcal{FL}_\bot$. Recalling the characterizations of equivalence and subsumption for reduced normal forms in $\mathcal{FL}_\bot$ and $\mathcal{ALN}$, this is not surprising. By comparing Lemma 6.8 and 6.19, we find almost the same conditions for subsumption. Observe, that we again assumed C to be in reduced normal form. This is legitimate, since in the definition of $\mathcal{A}_{\mathcal{ALN}}$, C is only referred to in reduced normal form.

Lemma 6.32 Deletion condition in $\mathcal{ALN}$
Let $P := (C \equiv^? D, \{X_j \sqsubseteq^? E_j | 1 \leq j \leq \ell\})$ be an $\mathcal{ALN}$-matching problem with non-strict side conditions in (U,V,W)-labelled $\mathcal{FL}_0$-normal form. Denote the role languages occuring during the execution of $\mathcal{A}_{\mathcal{ALN}}(P)$ as specified in Definition 6.23 Then $\mathcal{A}_{\mathcal{ALN}}(P)$ meets the deletion condition specified in Definition 6.24 for all occurring role languages.

Proof.
At first, the assertion is proved for role languages referring to the $\bot$-concept and then for the remaining cases.

▷ $\bot$-concept: Assume that a word w can appear in a role language for greater t after having been deleted, i.e. there exists a word $w \in \Sigma^*$, and indices $t < t' \in T$ and a $j \in \{1, \ldots, \ell\}$ such that $w \in U_{t,j,\bot}$ and $w \notin U_{t',j,\bot}$ but $w \in U_{t'+1,j,\bot}$. We can now infer a contradiction to $U_{t,j,\bot}$ being prefix free, as already done for $\mathcal{FL}_\bot$ in Lemma 6.29.

As the substitutions σ_t, $\sigma_{t'}$ and $\sigma_{t'+1}$ are reduced, we can infer from the assumptions by virtue of the properties of reduced normal forms in $\mathcal{ALN}$ that $U_{t,j,\bot} \succ U_{t',j,\bot} \succ U_{t'+1,j,\bot}$. The rest of the argument is analogous to Lemma 6.29. We apply the definition of the multiset order ($\succ$) and infer that $U_{t,j,\bot}$ must contain a nontrivial prefix of w as well as w itself.

▷ Other cases: Assume similarly for a word $w \in \Sigma^*$ that $w \in U_{t,j,A}$ and $w \notin U_{t',j,A}$, but $w \in U_{t'+1,j,A}$ for an atomic concept $A \in \mathcal{C}$, for some $j \in \{1, \ldots, \ell\}$, and for nonnegative integers $t < t' \in T$. Since again $\sigma_t \sqsubseteq \sigma_{t'} \sqsubseteq \sigma_{t'+1}$ and since all substitutions are reduced, we yield by Lemma 6.19:

$$U_{t,j,A} \,\dot\cup\, U_{t,j,\bot} \cdot \Sigma^* \;\supseteq\; U_{t',j,A} \,\dot\cup\, U_{t',j,\bot} \cdot \Sigma^* \;\supseteq\; U_{t'+1,j,A} \,\dot\cup\, U_{t'+1,j,\bot} \cdot \Sigma^*$$

Now we can follow the argument employed in Lemma 6.29 to infer a contradiction to the disjointness of the unions. It is shown in Lemma 6.19 that the argument of disjoint unions also applies for negated atomic concept and number restrictions. □

This section completes our discussion on matching problems modulo equivalence with non-strict side conditions. It should be noted that our proofs strongly rely on the assumption of reduced normal forms. Nevertheless, we have pointed out in Section 6.5 that the algorithm $\mathcal{A}_\mathcal{L}$ has the same behaviour without this requirement. The results obtained here therefore apply to the algorithm as introduced in Definition 6.20. It might be worth emphasizing that the algorithm proposed here is applicable to both acyclic and cyclic side conditions.

CHAPTER 7

Conclusion

7.1 Summary

In this work the computational complexity of matching algorithms has been discussed for four common description logics—$\mathcal{ALN}$ and three of its sublanguages. Three different problems have been considered in this context: Matching modulo equivalence without side conditions, the approach of eliminating side conditions and the use of fixed point algorithms for solving matching problems with side conditions.

The article [BKBM99] by Baader, Küsters, Borgida, and McGuinness formed the basis for our work, providing a characterization of subsumption for our logics and algorithms for matching problems without side conditions. Regarding the computational complexity of these algorithms, only one minor gap had to be closed. In order to formally verify that the algorithms are efficient, the properties of treelike automata had to be examined. This gap is closed by Chapter 4, where treelike automata have been discussed in depth. In consequence, the results of [BKBM99] have been confirmed—matching problems without side conditions can be solved in polynomial time. This has been our first main topic.

It was shown in [BKBM99] that matching problems with non-strict side conditions cannot be solved efficiently by straightforwardly eliminating side conditions. It has been our second main topic in Chapter 5 to discuss how the approach of eliminating side conditions can be carried out successfully. Eventually, we have seen that structure sharing is required for compact representations of role languages in order to gain an efficient solution. However, the result proposed here is limited to acyclic side conditions. It is not clear whether this idea can be extended to the cyclic case.

Finding a solution applicable to matching problems with acyclic as well as cyclic side conditions has been the third main issue in this work (covered in Chapter 6). Here a fixed point algorithm has been developed solving this problem. In this context, reduced normal forms have been introduced in order to simplify the proof of termination for that algorithm. Nevertheless, reduced normal forms might also be interesting in $\mathcal{FL}_\bot$, $\mathcal{FL}_\neg$, and $\mathcal{ALN}$, because they reduce equivalence to equality in these logics. $\mathcal{FL}_0$-normal forms do not suit this purpose.

7.2 Future goals

In addition to the non-strict case, in [BKBM99] also matching problems with strict side conditions are examined. It was shown by a reduction to 3SAT ([GJ79]) that matching under strict side conditions is NP-hard even in $\mathcal{FL}_0$. Nevertheless, an appropriate matching

algorithm has not yet been proposed. It seems worthwhile utilize the efficient matching algorithm for non-strict side conditions proposed here in the context of a matching algorithm for strict side conditions.

For matching under non-strict side conditions, the fixed-point algorithm proposed in Chapter 6 comprises a strategy built on top of a matching algorithm already existing for $\mathcal{ALN}$ or its sublanguages studied here. It might be promising to apply a similar approach to other description logics, where standard matching algorithms have already been found. One such example could be the logic $\mathcal{ALE}$, which allows for existential role restrictions instead of number restrictions. Matching in $\mathcal{ALE}$ has already been studied in [BK00a].

Bibliography

[BK00a] F. Baader and R. Küsters. Matching in description logics with existential restrictions. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, pages 261–272, San Francisco, CA, 2000. Morgan Kaufmann Publishers. → p. 82

[BK00b] A. Borgida and R. Küsters. What's not in a name: Some properties of a purely structural approach to integrating large dl knowledge bases. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, number 33 in CEUR-WS, Aachen, Germany, 2000. → p. 3

[BKBM99] F. Baader, R. Küsters, A. Borgida, and D. McGuinness. Matching in description logics. *Journal of Logic and Computation*, 9(3):411–447, 1999. → pp. 2, 3, 4, 9, 10, 11, 12, 14, 17, 20, 40, 41, 42, 45, 58, 62, 63, 66, 67, 73, 74, 81

[BMS+91] Ronald Brachman, Deborah McGuinness, Peter Patel Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In John F. Sowa, editor, *Principles of Semantic Networks — Explorations in the Representation of Knowledge*, pages 401–456. Morgan Kaufmann, 1991. → p. 3

[BN98a] F. Baader and P. Narendran. Unification of concept terms in description logics. In H. Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 331–335. John Wiley & Sons Ltd, 1998. → pp. 1, 2, 10

[BN98b] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, 1998. → p. 56

[BS96] F. Baader and U. Sattler. Knowledge representation in process engineering. In *Proceedings of the International Workshop on Description Logics*, Cambridge (Boston), MA, U.S.A., 1996. AAAI Press/The MIT Press. → p. 3

[DLNN91] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 151–162, San Mateo, CA, USA, April 1991. Morgan Kaufmann Publishers. → p. 2

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979. → p. 81

[HB91] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning, KR-91*, pages 335–346, Boston (USA), 1991. → p. 2

[HNSS90] Bernhard Hollunder, Werner Nutt, and Manfred Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proceedings of ECAI-90, 9th European Conference on Artificial Intelligence*, pages 348–353, Stockholm, Sweden, 1990. → p. 2

[HU80] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, N. Reading, MA, 1980. → pp. 5, 6

[Küs98] R. Küsters. Characterizing the Semantics of Terminological Cycles in $\mathcal{ALN}$ using Finite Automata. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 499–510. Morgan Kaufmann, 1998. → pp. 11, 17, 40, 52, 63, 76

[MPS98] Deborah L. McGuinness and Peter F. Patel-Schneider. Usability issues in knowledge representation systems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 608–614, Menlo Park, July 26–30 1998. AAAI Press. → p. 2

[YZ91] S. Yu and Q. Zhuang. On the state complexity of intersection of regular languages. *ACM SIGACT News*, 22(3):52–54, 1991. → p. 20