FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Practical Automata-Based Technique For Reasoning In Expressive Description Logics

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Master of Science (Computational Logic) (MSc)

im Rahmen des Studiums

Computational Logic (Erasmus-Mundus)

eingereicht von

Domenico Carbotta
Matrikelnummer 0727901

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung

Betreuer: Prof. Dr. Thomas Eiter

Mitwirkung: Prof. Dr. Diego Calvanese
Dr. Magdalena Ortiz

Wien, 14.10.2010

(Unterschrift Verfasser)

(Unterschrift Betreuer)

Domenico Carbotta
Via S. d'Acquisto, 26
I-70055 Minervino Murge
Italy

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit — einschließlich Tabellen, Karten und Abbildungen —, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 14.10.2010

_____

Probably I am very naïve, but I also think
I prefer to remain so, at least for the time being
and perhaps for the rest of my life.

E. W. Dijkstra

Abstract

Description Logics are a well-established family of languages designed for knowledge representation and reasoning. Thanks to the balance they provide between expressivity and computational efficiency, they have gained acceptance in several areas, such as information integration, bioinformatics, software engineering. The reasoning requirements arising in these different scenarios have stimulated many paths of research.

In this work we will take advantage of specific properties of a wide range of Description Logics, the *tree model property*. This property makes it possible to reduce DL reasoning problems to verifying certain properties of an automaton over infinite trees. Many worst-case optimal algorithms have been obtained using this technique; however, their practicality is questioned because they usually involve costly computations on very large objects.

The goal of this thesis is to help overcome these hurdles, designing and implementing a new automata-based procedure for reasoning over expressive Description Logics that is worst-case optimal and lends itself to an efficient implementation. In order to show the feasibility of the approach, we have realized a working prototype of a reasoner based upon these techniques. An experimental evaluation of this prototype shows encouraging results.

Kurzfassung

Description Logics sind eine Familie von Logiken, die speziell für die Wissensrepräsentation eingeführt wurden und in den letzten Jahren ausgiebig erforscht worden sind. Aufgrund der Ausgewogenheit zwischen ihrer Ausdrucksstärke und der Effizienz im automatischen Schließen, haben Description Logics in verschiedenen Gebieten Anwendung gefunden, zum Beispiel in der Integration von Informationen, in der Bioinformatik und im Software Engineering. Die Bedürfnisse bezüglich Inferenzkapazität in all diesen Bereichen haben die Forschung in verschiedenen Richtungen vorangetrieben.

In dieser Arbeit nützen wir eine Eigenschaft, die die meisten Description Logics besitzen, nämlich die Baummodell Eigenschaft. Dank dieser Eigenschaft ist es möglich, automatisches Schließen in Description Logics auf das Problem zu reduzieren, bestimmte Eigenschaften von endlichen Automaten auf unendlichen Bäumen zu überprüfen. Viele Algorithmen wurden entwickelt, die diese Idee nützen und worst-case optimal sind; allerdings lassen sich diese Algorithmen in der Praxis nicht gut nützen, da sie meistens aufwändige Berechnungen über sehr großen Objekten erfordern.

Das Ziel dieser Diplomarbeit ist es, diese Schwierigkeiten überwinden zu helfen, und einen neuen automatenbasierten Algorithmus für das Schließen in ausdrucks-

starken Description Logics zu entwickeln, der worst-case optimal ist und auch für eine effiziente Implementierung geeignet ist. Um zu zeigen, dass der Ansatz auch in der Praxis realisierbar ist, haben wir auch einen Prototypen eines Reasoners entwickelt, der den neuen Algorithmus für automatisches Schließen implementiert. Eine experimentelle Evaluierung des Prototypen zeigt ermutigende Resultate.

Contents

Chapter 1

Introduction

Knowledge representation has emerged in the last forty years as one of the most active branches of research in the realm of artificial intelligence. After all, every "intelligent application" must base its inference processes on a machine-readable description of its domain.

Research in knowledge representation is focused on the identification and description of modeling formalisms. One of the fundamental goals is reaching a balance between expressive power – the ability of modeling more complex scenarios – and reasoning efficiency – the possibility of drawing inferences in a timely fashion.

The Description Logic family [BCM+07] has emerged in the latest years as one of the most interesting results of this research area. Different applications call for different levels of expressiveness and different reasoning services: the DL family makes it possible to combine language features in order to obtain the most convenient trade-off.

1.1 Background

All Description Logic languages model the domain of interest by means of *concepts* (classes of objects) and *roles* (binary relations between objects). The final outcome of the modeling process is a *Knowledge Base*, a set containing axioms that with intensional knowledge and assertions that specify the participation of objects to concepts and roles. In this work we will be focusing on the intensional component of a Knowledge Base, the so-called *TBox*.

As we have mentioned, a wide range of Description Logic languages can be obtained by choosing different combinations of constructors for concepts and roles. In this work we will focus on *expressive* DLs, i.e. languages that support a rich set of constructors; *lightweight* DLs, on the other hand, shift the trade-off towards a lower computational complexity (see [CDGL$^+$05]).

Various reasoning tasks emerge in this context. The first is concerned with the *satisfiability of a concept*, i.e. checking whether a concept description can be satisfied by an object. For instance, the concept

$$\forall \mathsf{supervises}.\mathsf{Student} \sqcap \exists \mathsf{supervises}.(\neg \mathsf{Student})$$

can be shown to be unsatisfiable. Intuitively, it describes the class of all objects that *(i)* only supervise students, and *(ii)* supervise at least one non-student — the two requirements are in contradiction.

Analogously, another important reasoning task is concerned with the *satisfiability of a TBox*. A third reasoning task emerges from the interplay of these two problems: checking the *consistency of a concept in a TBox*, i.e. making sure that there exists a model of a TBox where at least one object participates

in the given concept. For instance, consider a TBox $\mathcal{T}$ containing the following assertions:

$$\mathsf{NoviceTeacher} \sqsubseteq \forall \mathsf{teaches}.\mathsf{IntroductoryCourse}$$

$$\mathsf{ExperiencedTeacher} \sqsubseteq \exists \mathsf{teaches}.\mathsf{AdvancedCourse}$$

$$\exists \mathsf{teaches}.\mathsf{AdvancedCourse} \sqsubseteq \mathsf{ExperiencedTeacher}$$

$$\mathsf{IntroductoryCourse} \sqcap \mathsf{AdvancedCourse} \sqsubseteq \bot$$

$$\mathsf{Seminary} \sqsubseteq \mathsf{AdvancedCourse}$$

Intuitively, they describe the following constraints:

- a novice teacher only teaches introductory courses;
- an experienced teacher teaches at least one advanced course;
- everyone who teaches at least one advanced course is an experienced teacher;
- no object is at the same time an introductory course and an advanced course (namely, the intersection between the two concepts is the empty concept $\bot$);
- every seminary is an advanced course.

The concept $\mathsf{NoviceTeacher} \sqcap \mathsf{ExperiencedTeacher}$, identifying those objects that are both novice and experienced teachers, can be shown to be inconsistent in $\mathcal{T}$. Every course taught by a novice teacher will be an introductory course, and since nothing can be both an introductory and an advanced course at the same time, no novice teacher can teach an advanced course — and, therefore, no novice teacher satisfies the requirements to be an experienced teacher.

The last reasoning problem we address in this work is the *implication of an assertion by a TBox*. For instance, consider the following assertion:

$$\exists \text{teaches.Seminary} \sqsubseteq \text{ExperiencedTeacher}$$

Intuitively, it states that whoever teaches a seminary is an experienced teacher. The TBox $\mathcal{T}$ given above implies this assertion: every seminary is an advanced course, and everyone who teaches advanced courses is an experienced teacher.

1.2 Motivation

Automata-based decision procedures for various logic languages have been a very active line of research for the last fifty years, since the work of J. Richard Büchi [Büc60] and Michael O. Rabin [Rab69] on different fragments of monadic second order logic.

Automata-based techniques for Description Logics have led to many interesting results, starting with the complexity bounds for $\mathcal{SHIQ}$ in [Tob01] and for $\mathcal{ALCQIb}_{reg}$ in [CDGL02a]. For a thorough review of the field, the reader can refer to [Ort10, Section 3.5]. Attempts at a successful implementation, though, have found serious difficulties.

Constructing a tree automaton that recognizes a suitable encoding of the same models as a given DL concept or knowledge base is a relatively easy task; manipulating the resulting automaton, though, has proven to be rather problematic. The main issue lies with the necessity of operating on intermediate representations of the automaton, whose space state is exponentially bigger than the size of the input formulas.

Reasoning on many DL languages has shown to be provably intractable (i.e., EXPTIME-hard), so this behaviour is to be expected on some "tough" problem instances. Reducing the number of these difficult instances, though, remains an open problem.

1.3 Contributions

In this work we attempt to attack the problem from a different angle. We will focus on $\mathcal{ALC}^{\mathrm{f}}$, a variant of $\mathcal{ALC}$ which also allows global functionality assertions: by choosing an expressive yet "small" DL language, we will try to keep the algorithm as simple as possible. This will allow us to obtain a procedure that does not have to operate on big intermediate objects, yielding an encouraging performance level even with a relatively naïve implementation.

The results obtained in this simple scenario can successively be extended to cover richer languages. A possible way to handle inverse roles will be described in this work, although it is not (yet) part of the prototype. Several preprocessing steps described in the literature can be used to $\mathcal{SHIQ}$; the effectiveness of the resulting reasoning algorithms will have to be investigated.

1.4 Structure of the Work

In Chapter 2 we will introduce the main formalisms used throughout this work. In particular, we will give a detailed account of the properties of DL languages that make the automata-based approach possible.

Chapter 3 contains an exposition of various reductions. Starting from a TBox and a concept, we will show how to build a tree automaton that recognizes (a

suitable representation of) the models of the given TBox in which at least an individual satisfies the specified concept.

In Chapter 4 we will move our focus to the tree automata. We will show a chain of procedures that make it possible to decide the emptiness of a given automaton.

In Chapter 5 we will finally combine the pieces of the puzzle. We will describe a decision procedure that, starting from a TBox and a concept, implicitly builds the corresponding automaton (as described in Chapter 3) and uses a modified version of the procedures developed in Chapter 4 to decide its emptiness.

This procedure has been implemented in a working prototype: in Chapter 6, we will describe the main components of its architecture. We will also present the results of a few synthetic benchmarks which show the scalability of the approach.

Chapter 2

Preliminaries

In this section we will introduce a few key formalisms and fix the notation used throughout the rest of the work.

We will start with a quick overview of some basic concepts (words, trees, boolean formulae). In the following section we will give a formal definition of automata on infinite trees, and describe the way in which they recognize a given tree. We will close the chapter with an introduction to the family of Description Logics and to the reasoning problems we will address in this work.

2.1 Generalities

This section contains basic definitions and theorems regarding the foundations of this work. This overview is not intended to be exhaustive in any way: its main purpose is to formulate a theoretical background and fix the notation.

2.1.1 Words

Let X be a non-empty finite set: a *word* over X is an ordered sequence of symbols chosen from X. The *length* of a word is the number of symbols that appear in the sequence; the *empty word* ε is the sequence with length 0. The set of words that can be built from X is denoted by X^*. For any two words $v, v' \in X^*$, their *concatenation* is a word containing the sequence of characters from v followed by the sequence of characters from v', and is denoted by $v \cdot v'$. Given a word $v = c_1 \dots c_n$ of length $n \geqslant 1$, we define the following functions

$$\textbf{init}(v) = c_1 \dots c_{n-1}$$
$$\textbf{last}(v) = c_n$$

2.1.2 Trees

Let k be a natural number: a k-*ary tree* N is a subset of $[1 : k]^*$ (the set of words built from an alphabet including all natural numbers from 1 to k) that enjoys the following properties:

1. N contains the empty word, i.e. $\varepsilon \in N$;
2. N is prefix-closed: if $v \in N$, then $\textbf{init}(v) \in N$.

The elements of N are called *nodes*. The *parent* of a node $v = c_1 \dots c_n$ is the node $\textbf{init}(v) = c_1 \dots c_{n-1}$; the node v is a *child* of $\textbf{init}(v)$. A *leaf node* is a node $v \in N$ such that, for any $i \in [1 : k]$, $v \cdot i \notin N$ (i.e., a leaf has no child nodes). A *complete* k-ary tree is a tree that has all the elements of $[1 : k]^*$ as nodes.

A *path* $\pi = w_1, w_2, \dots$ is a sequence of nodes from N such that every word is the parent of the word that follows it. Note that a path does not have to be

finite. A *branch* is a maximal path, i.e. a path that starts from the root and either ends in a leaf node or is infinite.

Given a set Σ, a Σ-*labeled* k-*ary tree* is a tuple $T = (N, \tau)$, where N is a k-ary tree and $\tau : N \to \Sigma$ is the *labeling function*. Σ is the *labeling alphabet*, $\tau(v)$ is the *label* of the node v.

2.1.3 Positive Boolean Formulas

Given a set X, the set $\mathbf{PBF}(X)$ contains all the *positive boolean formulas* that can be built according to the following grammar:

$$\begin{aligned} \phi \quad &\to \quad \mathbf{true} \\ &\mid \quad \mathbf{false} \\ &\mid \quad (\phi) \\ &\mid \quad x \in X \\ &\mid \quad \phi \vee \phi \\ &\mid \quad \phi \wedge \phi \end{aligned}$$

We recursively define the function $\mathbf{sat} : \mathbf{PBF}(X) \times 2^X \to \{\mathbf{true}, \mathbf{false}\}$ as follows:

$$\begin{aligned} \mathbf{sat}(\mathbf{true}, Y) &= \mathbf{true} \\ \mathbf{sat}(\mathbf{false}, Y) &= \mathbf{false} \\ \mathbf{sat}((\phi), Y) &= \mathbf{sat}(\phi, Y) \\ \mathbf{sat}(x, Y) &= \begin{cases} \mathbf{true} & \text{if } x \in Y \\ \mathbf{false} & \text{otherwise} \end{cases} \end{aligned}$$

$$\mathbf{sat}(\phi \vee \phi', Y) = \begin{cases} \mathbf{true} & \text{if } \mathbf{sat}(\phi, Y) \text{ or } \mathbf{sat}(\phi', Y) \\ \mathbf{false} & \text{otherwise} \end{cases}$$

$$\mathbf{sat}(\phi \wedge \phi', Y) = \begin{cases} \mathbf{true} & \text{if } \mathbf{sat}(\phi, Y) \text{ and } \mathbf{sat}(\phi', Y) \\ \mathbf{false} & \text{otherwise} \end{cases}$$

A set $Y \subseteq X$ *satisfies* a formula $\phi \in \mathbf{PBF}(X)$ if $\mathbf{sat}(\phi, Y) = \mathbf{true}$; in this case, we will say that Y is a *model* of ϕ, and write $Y \models \phi$. Two formulas are *equivalent* if they are satisfied by the same sets; we will denote equivalence using the $\equiv$ operator (e.g., $\phi \equiv \phi'$).

A *substitution* for formulas in $\mathbf{PBF}(X)$ is a partial function $\sigma : X \to \mathbf{PBF}(X)$. The result of applying a substitution σ to a formula ϕ is denoted as $\phi\sigma$, and is defined as follows:

$$x\sigma = \begin{cases} \sigma(x) & \text{if } \sigma(x) \text{ is defined} \\ x & \text{otherwise} \end{cases}$$

$$(\mathbf{true})\sigma = \mathbf{true}$$

$$(\mathbf{false})\sigma = \mathbf{false}$$

$$(\phi \vee \phi')\sigma = \phi\sigma \vee \phi'\sigma$$

$$(\phi \wedge \phi')\sigma = \phi\sigma \wedge \phi'\sigma$$

A substitution $\{(x_1, \phi_1), \ldots, (x_n, \phi_n)\}$ is often denoted as $\left[x_1, \ldots, x_n \,/\, \phi_1, \ldots, \phi_n\right]$.

A formula is said to be in *disjunctive normal form (DNF)* if it is expressed as the disjunction of *clauses* (conjunctions of atoms), i.e. it is in the following form:

$$(c_{1,1} \wedge \cdots \wedge c_{1,n_1}) \vee \cdots \vee (c_{n,1} \wedge \cdots \wedge c_{n,n_n})$$

A formula is said to be in *conjunctive normal form (CNF)* if it is expressed as the conjunction of *(dual) clauses* (disjunctions of atoms), i.e. it is in the following form:

$$(\beta_{1,1} \vee \cdots \vee \beta_{1,n_1}) \wedge \cdots \wedge (\beta_{n,1} \vee \cdots \vee \beta_{n,n_n})$$

A *DNF expansion* of a formula ϕ is a formula $\phi_{\textbf{dnf}}$ in disjunctive normal form which is equivalent to ϕ; analogously, a *CNF expansion* of a formula ϕ is a formula $\phi_{\textbf{cnf}}$ in conjunctive normal form which is equivalent to ϕ. It is possible to obtain a DNF and CNF expansion of a formula ϕ by repeatedly applying the distribution rules of $\wedge$ over $\vee$ and vice versa:

$$\phi \wedge (\psi \vee \psi') \equiv (\phi \wedge \psi) \vee (\phi \wedge \psi')$$
$$\phi \vee (\psi \wedge \psi') \equiv (\phi \vee \psi) \wedge (\phi \vee \psi')$$

2.2 Tree Automata

A *tree automaton* is a tuple $A = (\Sigma, Q, q_0, k, \delta, \mathcal{F})$, whose components are defined as follows:

- Σ is the *input alphabet*, a finite set of *input symbols*;
- Q is the (finite) set of *states*;
- $q_0 \in Q$ is the *initial state*;
- $k \in \mathbb{N}$ is the *branching degree*;
- $\mathcal{F}$ is the *acceptance condition*;
- $\delta : Q \times \Sigma \to \textbf{PBF}([-1 : k] \times Q)$ is the *transition function*.

We will introduce different kinds of acceptance conditions once we have defined the run of a tree automata.

Given an automaton $A = (\Sigma, Q, q_0, k, \delta, \mathcal{F})$ and a Σ-labeled k-ary tree $T = (N, \tau)$, the *run* of A over T is a $N \times Q$-labeled tree $R = (N_R, \rho)$ that satisfies the following constraints:

1. $\rho(\varepsilon) = (\varepsilon, q_0)$;
2. for every $r \in N_R$, if $\rho(r) = (v, q)$, then there exist a set of index-state pairs $\{ (i_1, q_1), \ldots, (i_n, q_n) \}$ that satisfies the positive boolean formula $\delta(q, \tau(v))$ and such that, for every $j \in [1 : n]$, $r{\cdot}j \in N_R$ and $\rho(r \cdot j) = (v \cdot i_j, q_j)$.

Note that, with a slight abuse of notation, we extend the concatenation operator so that, for every word $v \in [1 : k]^*$ and every symbol $c \in [1 : k]$, $v \cdot 0 = v \cdot c \cdot -1 = v$.

For every node $r \in N_R$, with $\rho(r) = (v, q)$, we define $\mathbf{lab}_R(r)$ as follows:

$$\begin{aligned}\mathbf{lab}_R(r) = &\left\{ (i, q') \;:\; \exists j.\ \rho(r \cdot j) = (v \cdot i, q') \right\} \cup \\ &\cup \left\{ (0, q') \;:\; \exists j.\ \rho(r \cdot j) = (v, q') \right\} \cup \\ &\cup \left\{ (-1, q') \;:\; \exists j.\ \rho(r \cdot j) = (\mathbf{init}(v), q') \right\}\end{aligned}$$

We can rephrase the second condition in the definition of run as follows: for every $r \in N_R$, $\mathbf{lab}_R(r) \models \delta(q, \tau(v))$.

Informally, every node of the run tree captures a snapshot of the automaton moving over the original tree. The first component of the label $\rho(\cdot)$ points at a location in the original tree, and the second component records the current state of the automaton. The first condition above starts the automaton on the

root of the original tree, in the initial state q_0. The second condition describes a step in the run of the automaton, which is carried out as follows:

- The automaton reads the current symbol c.
- The automaton nondeterministically guesses a set of index and state pairs $\{(i_1, q_1), \ldots, (i_n, q_n)\}$ that satisfies $\delta(q, c)$.
- The automaton spawns n copies of itself. For each $j \in [1, n]$, the j-th copy will switch to state q_j and move according to the value of i_j:
  - If $i_j > 0$, the j-th copy moves to the i_j-th child of the current node;
  - If $i_j = 0$, the j-th copy stays in the current node (a *zero-transition*);
  - If $i_j = -1$, the j-th copy moves to the parent of the current node.

Each copy of the automaton will then repeat the process.

An automaton $A = (\Sigma, Q, q_0, k, \delta, \mathcal{F})$ *accepts* a Σ-labeled tree T iff there exists a run R of A over T that satisfies the acceptance condition $\mathcal{F}$ (an *accepting run*). We distinguish between different families of acceptance conditions:

- *Looping condition:* every run is considered to be accepting. In this case, we will drop the $\mathcal{F}$ component from the tuple that specifies the automaton.
- *Büchi condition:* the acceptance condition is a set $Q_{\mathcal{F}} \subseteq Q$. A run is said to be accepting iff, for every infinite branch of the run tree, there exists (at least) a state in $Q_{\mathcal{F}}$ that occurs infinitely often in its labeling.
- *Co-Büchi condition:* the acceptance condition is a set $Q_{\mathcal{F}} \subseteq Q$. A run is said to be accepting iff no state in $Q_{\mathcal{F}}$ occurs infinitely often in the labeling of any infinite branch of the run tree.

- *Parity condition:* the acceptance condition is a function $\alpha : Q \to \mathbb{N}$. A run is said to be accepting iff, for every infinite branch in the run tree, the smallest index $\alpha(q)$ associated to a state that occurs infinitely often in the labeling of that branch is even.

Given an automaton $A = (\Sigma, Q, q_0, k, \delta, \mathcal{F})$, the *language accepted by* A (denoted by $\mathcal{L}(A)$) is the set of all the Σ-labeled k-ary trees for which there exists an accepting run of A.

Orthogonally, automata can be classified according to several constraints imposed over their transition function. In this work, we will use the following classes of automata:

- *Two-way alternating automata:* no restriction is imposed over the transition function.
- *(One-way) alternating automata:* the transition function does not include any atom whose first component is -1. Informally, the automaton will never move "upward" (i.e., towards the root of the tree).
- *Nondeterministic automata:* the transition function does not include any atom whose first component is -1 or 0; moreover, for every pair $(q, \sigma) \in Q \times \Sigma$, no clause in the disjunctive normal form of $\delta(q, \sigma)$ can contain two atoms that share the same first component.
- We will also introduce a non-standard family of automata: *zero-free (one-way) alternating automata*, the class of all automata whose transition function does not include any atom whose first component is -1 or 0.

Moreover, we will say that an ALT $A_{\mathbf{a}} = (\Sigma, Q, q_0, k, \delta)$ is *zero-layered* if there exists a bijection $\mathbf{depth} : Q \to [1 : |Q|]$ such that, for any two states q, q' and

any symbol c, if $\mathbf{depth}(q) \leqslant \mathbf{depth}(q')$ then the atom $(0, q')$ does not appear in the PBF $\delta(q, c)$.

From now on we will use the acronyms *2-ALT* to indicate two-way alternating looping automata, *ALT* for alternating looping automata, *ZLT* for zero-free alternating looping automata, and *NLT* for nondeterministic looping automata.

2.2.1 Minimal Run Trees

A *minimal run tree* for an automaton A over a tree $T = (N_T, \tau)$ is a run tree $R = (N_R, \rho)$ such that, for every set $N'_R \subset N_R$, the tree $R' = (N'_R, \rho)$ is not a run tree for A over T. It is clearly possible to obtain a minimal run tree starting from any run tree and successively removing nodes.

Because of the trivial acceptance condition, any looping automata that accepts a given tree has an accepting run tree over it that is also minimal. The same property can be shown to hold for Büchi, co-Büchi and parity automata by a simple argument: by removing nodes from a given run tree, the set of infinite branches cannot increase. As a consequence, if all the infinite branches in the starting run tree satisfy the acceptance condition, so will the infinite branches of the run tree resulting from the removal.

In the following sections we may therefore choose, without loss of generality, to only work with minimal run trees.

2.2.2 Partial Run of an Automaton

Given an automaton $A = (\Sigma, Q, q_0, k, \delta, \mathcal{F})$, a tree $T = (N, \tau)$, a node $v \in N$ and a state $q \in Q$, a *partial run* of A over T starting from (v, q) is a $N \times Q$-

labeled tree $\mathrm{PR}_q^\nu = (\mathrm{N}_q^\nu, \rho_q^\nu)$ that satisfies the following constraints:

- $\rho_q^\nu(\varepsilon) = (\nu, q)$
- if $\rho_q^\nu(r) = (\nu', q')$, then $\mathbf{lab}_{\mathrm{PR}_q^\nu}(r) \models \delta(q', \tau(\nu'))$.

Informally, a partial run makes it possible to start an automaton from an arbitrary configuration. This will enable us to build "full" runs of an automaton piece-wise, by combining smaller partial runs.

2.3 Description Logics

Description Logics [BCM+07] are a well-established family of languages designed for knowledge representation and reasoning. Thanks to the balance they provide between expressivity and computational efficiency, they have gained acceptance in fields such as information integration [CDGL02b], software engineering [BCM+07, Ch. 11] and bioinformatics [BCM+07, Ch. 13]. In particular, the OWL and OWL 2 standardization efforts for the Semantic Web [W3C04, W3C09] have been heavily influenced by research in the field of Description Logics.

2.3.1 Syntax and Semantics

Description Logics focus on representing the domain of interest by identifying groups of individuals (*concepts*) and relationships between individuals (*roles*). The *signature* of a specific DL language is a pair $(\mathcal{C}, \mathcal{R})$, where $\mathcal{C}$ and $\mathcal{R}$ are two disjoint sets containing *atomic concepts* and *atomic roles*. These symbols can be

used to form *concept* and *role expressions* according to a given set of syntactic rules.

Fix a signature $(\mathcal{C}, \mathcal{R})$, with $\mathcal{C} = \{ A_i \}$ and $\mathcal{R} = \{ P_i \}$. The following grammar generates the DL language called $\mathcal{ALC}$, which represents the starting point for most languages in the DL family, and for all the languages used in this work. The productions R and C identify respectively role and concept expressions.

R	$\rightarrow$	$P_1 \mid P_2 \mid \ldots$	atomic roles from $\mathcal{R}$
C	$\rightarrow$	$\top$	*top* concept
	$\mid$	$\bot$	*bottom* concept
	$\mid$	$A_1 \mid A_2 \mid \ldots$	atomic concepts from $\mathcal{C}$
	$\mid$	$\neg C$	complement
	$\mid$	$C \sqcup C$	concept union
	$\mid$	$C \sqcap C$	concept intersection
	$\mid$	$\exists R.C$	existential quantification
	$\mid$	$\forall R.C$	universal quantification

The grammar can be extended with other constructs, giving place to more expressive (and, possibly, computationally more expensive) languages. The extensions that will be mentioned throughout this work are:

- *Functional restrictions:* concept expressions of the form $(\leqslant 1\, R.\, \top)$ and $(\geqslant 2\, R.\, \top)$, where R is an arbitrary role expression.
- *Unqualified number restrictions:* concept expressions of the form $(\leqslant n\, R.\, \top)$ and $(\geqslant n\, R.\, \top)$, where n is a positive integer and R is an arbitrary role expression.

- *Qualified number restrictions:* concept expressions of the form $(\leqslant n\, R.\, C)$ and $(\geqslant n\, R.\, C)$, where n is a positive integer, R is an arbitrary role expression and C is an arbitrary concept expression;
- *Inverse roles:* role expressions of the form P^{-}, where P is an atomic role.

It is customary to identify the presence of an extension by appending a letter to the language name: $\mathcal{F}$ for functional restrictions, $\mathcal{N}$ for unqualified number restrictions, $\mathcal{Q}$ for qualified number restrictions and $\mathcal{I}$ for inverse roles. The logic $\mathcal{ALCIF}$, for instance, will include all the basic constructs of $\mathcal{ALC}$ plus functional restrictions and inverse roles.

Given a first order interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ over a DL signature $(\mathcal{C}, \mathcal{R})$, where atomic constructs are considered unary predicates and atomic roles are considered binary predicates, the semantics of an arbitrary DL expression can be defined by extending $\cdot^{\mathbf{I}}$ as follows:

$$
\begin{aligned}
(P^{-})^{\mathbf{I}} &= \left\{ (b, a) \in \Delta_{\mathbf{I}}^{2} \;:\; (a, b) \in P^{\mathbf{I}} \right\} \\
\top^{\mathbf{I}} &= \Delta_{\mathbf{I}} \\
\bot^{\mathbf{I}} &= \emptyset \\
(\neg C)^{\mathbf{I}} &= \Delta_{\mathbf{I}} \setminus C^{\mathbf{I}} \\
(C \sqcup D)^{\mathbf{I}} &= C^{\mathbf{I}} \cup D^{\mathbf{I}} \\
(C \sqcap D)^{\mathbf{I}} &= C^{\mathbf{I}} \cap D^{\mathbf{I}} \\
(\exists R.C)^{\mathbf{I}} &= \left\{ a \in \Delta_{\mathbf{I}} \;:\; \exists x \in \Delta_{\mathbf{I}}.\, (a, x) \in R^{\mathbf{I}} \wedge x \in C^{\mathbf{I}} \right\} \\
(\forall R.C)^{\mathbf{I}} &= \left\{ a \in \Delta_{\mathbf{I}} \;:\; \forall x \in \Delta_{\mathbf{I}}.\, (a, x) \in R^{\mathbf{I}} \rightarrow x \in C^{\mathbf{I}} \right\} \\
(\leqslant n\, R.\, C)^{\mathbf{I}} &= \left\{ a \in \Delta_{\mathbf{I}} \;:\; \left| \left\{ b \in \Delta_{\mathbf{I}} \;:\; (a, b) \in R^{\mathbf{I}} \wedge b \in C^{\mathbf{I}} \right\} \right| \leqslant n \right\} \\
(\geqslant n\, R.\, C)^{\mathbf{I}} &= \left\{ a \in \Delta_{\mathbf{I}} \;:\; \left| \left\{ b \in \Delta_{\mathbf{I}} \;:\; (a, b) \in R^{\mathbf{I}} \wedge b \in C^{\mathbf{I}} \right\} \right| \geqslant n \right\}
\end{aligned}
$$

A concept expression C is said to be *satisfiable* if it does not necessarily denote the empty concept, i.e. if there exists an interpretation $\mathbf{I}$ such that the set $C^{\mathbf{I}}$ is not empty.

An arbitrary concept is said to be in *negation normal form* if negation only occurs immediately above atomic concepts. It is possible to rewrite any $\mathcal{ALCIQ}$ concept in negation normal form. We define the mutually recursive functions $\mathbf{nnf}(\cdot)$ and $\mathbf{nnf}'(\cdot)$ as follows:

$$\begin{aligned}
\mathbf{nnf}(\top) &= \top \\
\mathbf{nnf}(\bot) &= \bot \\
\mathbf{nnf}(A) &= A \\
\mathbf{nnf}(D \sqcap D') &= \mathbf{nnf}(D) \sqcap \mathbf{nnf}(D') \\
\mathbf{nnf}(D \sqcup D') &= \mathbf{nnf}(D) \sqcup \mathbf{nnf}(D') \\
\mathbf{nnf}(\exists R.D) &= \exists R.\mathbf{nnf}(D) \\
\mathbf{nnf}(\forall R.D) &= \forall R.\mathbf{nnf}(D) \\
\mathbf{nnf}((\leqslant n\, R.\, D)) &= (\leqslant n\, R.\, \mathbf{nnf}(D)) \\
\mathbf{nnf}((\geqslant n\, R.\, D)) &= (\geqslant n\, R.\, \mathbf{nnf}(D)) \\
\mathbf{nnf}(\neg C) &= \mathbf{nnf}'(C)
\end{aligned}$$

$$\begin{aligned}
\mathbf{nnf}'(\top) &= \bot \\
\mathbf{nnf}'(\bot) &= \top \\
\mathbf{nnf}'(A) &= \neg A \\
\mathbf{nnf}'(D \sqcap D') &= \mathbf{nnf}'(D) \sqcup \mathbf{nnf}'(D') \\
\mathbf{nnf}'(D \sqcup D') &= \mathbf{nnf}'(D) \sqcap \mathbf{nnf}'(D') \\
\mathbf{nnf}'(\exists R.D) &= \forall R.\mathbf{nnf}'(D)
\end{aligned}$$

$$\begin{aligned}
\mathbf{nnf}'(\forall R.D) &= \exists R.\mathbf{nnf}'(D) \\
\mathbf{nnf}'((\leqslant n\, R.\, D)) &= (\geqslant (n+1)\, R.\, \mathbf{nnf}'(D)) \\
\mathbf{nnf}'((\geqslant n\, R.\, D)) &= \begin{cases} \bot & \text{if } n = 0 \\ (\leqslant (n-1)\, R.\, \mathbf{nnf}'(D)) & \text{otherwise} \end{cases} \\
\mathbf{nnf}'(\neg C) &= \mathbf{nnf}(C)
\end{aligned}$$

It is easy to prove that, for every concept C and every interpretation $\mathbf{I}$ over the same signature, $C^{\mathbf{I}} = \mathbf{nnf}(C)^{\mathbf{I}}$.

An *inclusion assertion* is an expression of the form $C \sqsubseteq D$, where C and D are concept expressions; we say that the interpretation $\mathbf{I}$ is a *model* of the assertion (or just "models the assertion") iff $C^{\mathbf{I}} \subseteq D^{\mathbf{I}}$. A *TBox* is a set of inclusion assertions; an interpretation $\mathbf{I}$ is a *model* of a given TBox $\mathcal{T}$ (we will write $\mathbf{I} \models \mathcal{T}$) if it is a model of every assertions contained in the TBox.

A TBox is said to be *satisfiable* if it has a model; a concept C is said to be *consistent* in a TBox $\mathcal{T}$ if there exists a model $\mathbf{I} \models \mathcal{T}$ such that $C^{\mathbf{I}} \neq \emptyset$. Clearly, a TBox $\mathcal{T}$ is satisfiable iff the concept $\top$ is consistent in $\mathcal{T}$.

A TBox $\mathcal{T}$ *logically implies* an assertion $C \sqsubseteq D$ (we will write $\mathcal{T} \models C \sqsubseteq D$) iff every model of $\mathcal{T}$ is also a model of $\mathcal{T} \cup \{ C \sqsubseteq D \}$ or, equivalently, the concept $C \sqcap \neg D$ is not consistent in $\mathcal{T}$.

2.3.2 The Tree Model Property

A *tree-shaped interpretation* $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ over a given DL signature $(\mathcal{C}, \mathcal{R})$ is an interpretation that satisfies the following constraints:

1. $\Delta_{\mathbf{I}}$ is a k-tree, for some $k \in \mathbb{N}$;

2. every individual is connected to *all* and *only* its child nodes, i.e.:

   - for every role $\mathsf{P} \in \mathcal{R}$ and every pair $(v, v') \in \mathsf{P}^{\mathbf{I}}$, either $v' = \mathbf{init}(v)$ or $v = \mathbf{init}(v')$;
   - for every individual $v \in \Delta_{\mathbf{I}}$, there exists a *single* role $\mathsf{P} \in \mathcal{R}$ such that either $(\mathbf{init}(v), v) \in \mathsf{P}^{\mathbf{I}}$ or $(v, \mathbf{init}(v)) \in \mathsf{P}^{\mathbf{I}}$.

For any concept expression C over the signature $(\mathcal{C}, \mathcal{R})$ (which we suppose without loss of generality to be in NNF), let $\mathbf{sub}(\mathsf{C})$ be the set of subconcepts of C — i.e. all the concept expressions that occur inside C, including C itself:

$$
\begin{aligned}
\mathbf{sub}(\mathsf{A}) &= \{\, \mathsf{A} \,\} \\
\mathbf{sub}(\neg\mathsf{A}) &= \{\, \neg\mathsf{A} \,\} \\
\mathbf{sub}(\top) &= \{\, \top \,\} \\
\mathbf{sub}(\bot) &= \{\, \bot \,\} \\
\mathbf{sub}(\mathsf{C} \sqcap \mathsf{C}') &= \left\{\, \mathsf{C} \sqcup \mathsf{C}' \,\right\} \cup \mathbf{sub}(\mathsf{C}) \cup \mathbf{sub}(\mathsf{C}') \\
\mathbf{sub}(\mathsf{C} \sqcup \mathsf{C}') &= \left\{\, \mathsf{C} \sqcap \mathsf{C}' \,\right\} \cup \mathbf{sub}(\mathsf{C}) \cup \mathbf{sub}(\mathsf{C}') \\
\mathbf{sub}(\exists \mathsf{R}.\mathsf{C}) &= \{\, \exists \mathsf{R}.\mathsf{C} \,\} \cup \mathbf{sub}(\mathsf{C}) \\
\mathbf{sub}(\forall \mathsf{R}.\mathsf{C}) &= \{\, \forall \mathsf{R}.\mathsf{C} \,\} \cup \mathbf{sub}(\mathsf{C})
\end{aligned}
$$

Let $\mathbf{exs}(\hat{\mathsf{C}})$ be the set of existential subconcepts of C — i.e., subconcepts of C of the form $\exists \mathsf{R}.\mathsf{C}'$. We define $\mathbf{idx}_{\mathsf{C}}$ as an arbitrary but fixed bijection from $\mathbf{exs}(\mathsf{C})$ to $[1 : |\mathbf{exs}(\mathsf{C})|]$.

A tree interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ is *subconcept ordered* with respect to C iff these two properties hold:

- the branching degree of $\Delta_{\mathbf{I}}$ is greater or equal to $|\mathbf{exs}(\mathsf{C})|$;

- for any individual v and any existential subconcept $\exists R.D$ from $\mathbf{exs}(C)$, if $v \in (\exists R.D)^{\mathbf{I}}$ then $(v, v \cdot \mathbf{idx}_C(\exists R.D)) \in R^{\mathbf{I}}$ and $v \cdot \mathbf{idx}_C(\exists R.D) \in C^{\mathbf{I}}$.

Intuitively, we can use $\mathbf{idx}_C$ as a "lookup table": given an existential concept $\exists R.D \in \mathbf{exs}(C)$, let $i = \mathbf{idx}_C(\exists R.D)$; in order to check whether an individual satisfies $\exists R.D$ we only need to check its i-th child.

Theorem 2.1. (Tree Model Property) *For any satisfiable $\mathcal{ALCI}$ concept C, there exists a tree-shaped interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ such that $\varepsilon \in C^{\mathbf{I}}$.*

Proof. Let $\hat{C}$ be an $\mathcal{ALCI}$ concept in negation normal form over the signature $(\mathcal{C}, \mathcal{R})$, and let $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ be an interpretation such that there exists an individual $a \in \hat{C}^{\mathbf{I}}$. We will show a procedure to build a tree-shaped interpretation $\mathbf{J} = (\Delta_{\mathbf{J}}, \cdot^{\mathbf{J}})$ such that $\varepsilon \in \hat{C}^{\mathbf{J}}$. This specific kind of model manipulation is usually referred to as *unraveling*.

Let $k = |\mathbf{exs}(C)|$; let μ be a partial mapping from $[1 : k]^*$ to $\Delta_{\mathbf{I}}$ built as follows:

1. let $\mu(\varepsilon) = a$;
2. for every v in the domain of μ and every existential subconcept $\exists R.C \in \mathbf{exs}(\hat{C})$, if $\mu(v) \in (\exists R.C)^{\mathbf{I}}$ let $\mu(v \cdot \mathbf{idx}_{\hat{C}}(\exists R.C))$ be an individual ζ such that $(\mu(v), \zeta) \in R^{\mathbf{I}}$ and $\zeta \in C^{\mathbf{I}}$.

We will build the tree model $\mathbf{J} = (\Delta_{\mathbf{J}}, \cdot^{\mathbf{J}})$ as follows:

- $\Delta_{\mathbf{J}} = \{ v \in [1 : k]^* \ : \ \mu(v) \text{ is defined} \}$;
- for every atomic concept $A \in \mathcal{C}$, $A^{\mathbf{J}} = \left\{ v \in \Delta_{\mathbf{J}} \ : \ \mu(v) \in A^{\mathbf{I}} \right\}$;
- for every atomic role $P \in \mathcal{R}$, $P^{\mathbf{J}} = \left\{ (v, v') \ : \ (\mu(v), \mu(v')) \in P^{\mathbf{I}}, \mathbf{rel}(v, v') \right\}$, where $\mathbf{rel}(v, v')$ is true if either $v = \mathbf{init}(v')$ or $v' = \mathbf{init}(v)$.

It is easy to verify that this interpretation is subconcept-ordered with respect to $\mathbf{idx}_{\hat{C}}$.

We claim that, for every $v \in \Delta_J$ and every concept $C \in \mathbf{sub}(\hat{C})$, if $\mu(v) \in C^I$ then $v \in C^J$. We will prove it by induction over the structure of C.

- C is an atomic concept. This base case holds by construction.
- $C = \neg A$ By definition, $v \in C^J$ iff $v \notin A^J$; by inductive hypothesis, this holds iff $\mu(v) \notin A^I$ or, equivalently, $\mu(v) \in C^I$.
- $C = D \sqcup E$ By definition, $v \in C^J$ iff $v \in D^J \cup E^J$; by inductive hypothesis, this holds iff $\mu(v) \in D^I \cup E^I = C^I$.
- $C = D \sqcap E$ By definition, $v \in C^J$ iff $v \in D^J \cap E^J$; by inductive hypothesis, this holds iff $\mu(v) \in D^I \cap E^I = C^I$.
- $C = \exists R.D$ Let $i = \mathbf{idx}(C)$; by construction, $\mu(v) \in C^I$ iff $(v, v \cdot i) \in R^J$ and $v \cdot i \in D^J$. By definition, this is equivalent to saying that $v \in C^J$.
- $C = \forall R.D$ By construction, all the R-successors of v are counter-images of R-successors of $\mu(v)$. By definition, if $\mu(v) \in C^I$ then all of its R-successors are in D^I, and by inductive hypothesis this means that all of their images are in D^J. As a consequence, $v \in C^J$.

By construction $\mu(\varepsilon) \in \hat{C}^I$, and therefore $\varepsilon \in \hat{C}^J$. □

2.3.3 TBox Internalization

We will define a custom DL extension, denoted by the subscript **prop**, which adds to the underlying language *propagated concepts* of the form $\forall u^*.C$. We

impose a further restriction on this construct, specifying that propagated concepts can never appear under the scope of an odd number of complementation operators. Given a signature $(\mathcal{C}, \mathcal{R})$ and an interpretation $\mathbf{I} = (\Delta_\mathbf{I}, \cdot^\mathbf{I})$ over the specified signature, the semantics of this construct are defined by extending $\cdot^\mathbf{I}$ as follows:

$$(u^*)^\mathbf{I} = \left(\bigcup_{P \in \mathcal{R}} P^\mathbf{I} \cup (P^-)^\mathbf{I} \right)^*$$

where X^* represents the transitive-reflexive closure of X over $\Delta_\mathbf{I}$, the smallest set such that:

1. $X \subseteq X^*$;
2. $\forall x, y, z.\ (x, y) \in X^* \wedge (y, z) \in X^* \rightarrow (x, z) \in X^*$;
3. $\forall x \in \Delta_\mathbf{I}.\ (x, x) \in X^*$.

It follows that the role u^* connects an individual to every other individual that is reachable via any combination of atomic and inverse roles.

We will say that an interpretation is *connected* iff any individual in its domain can be reached from any other individual via a concatenation of direct and inverse roles of arbitrary length. Formally, the condition can be stated as follows: for every two individuals $a, b \in \Delta_\mathbf{I}$, there exists a finite sequence of objects $c_1, \ldots, c_n \in \Delta_\mathbf{I}$ such that $c_1 = a$, $c_n = b$ and the following condition holds.

$$\forall i \in [1, n-1].\ \bigvee_{P \in \mathcal{R}} \left((a_i, a_{i+1}) \in P^\mathbf{I} \vee (a_{i+1}, a_i) \in P^\mathbf{I} \right)$$

By definition, any tree interpretation is connected. The following result, shown in the context of Modal Logics [Kri67] and Propositional Dynamic Logics

[Str82], clearly holds for a wide family of DL languages.

Lemma 2.2. *For every satisfiable $\mathcal{ALCI}_{\mathbf{prop}}$ concept C, there exists a connected model $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ such that $C^{\mathbf{I}} \neq \emptyset$.*

Intuitively, the importance of concept propagation lies in the fact that it allows us to encode global constraints on connected models in a concept expression: we will clarify this claim in the following lemma.

Lemma 2.3. *Given an $\mathcal{ALC}_{\mathbf{prop}}$ concept $\forall u^*.C$ and an interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ defined over the same signature such that $(\forall u^*.C)^{\mathbf{I}} \neq \emptyset$, if $\mathbf{I}$ is connected then $C^{\mathbf{I}} = \Delta_{\mathbf{I}}$.*

Proof. It is immediate to verify that, if $\mathbf{I}$ is connected, $(u^*)^{\mathbf{I}} = (\Delta_{\mathbf{I}} \times \Delta_{\mathbf{I}})$. The statement is therefore a direct consequence of the semantics of u^*. □

This property makes it possible to "embed" a description of a TBox in a single concept, as shown in the following theorem. Intuitively, we will use the propagation construct to check that each individual in the domain of the interpretation satisfies every inclusion assertion in the TBox, therefore asserting that the interpretation satisfies the TBox.

First of all, we need to show that the addition of propagated concepts to the language does not invalidate our previous result on tree-shaped models.

Theorem 2.4. *For any satisfiable $\mathcal{ALCI}_{\mathbf{prop}}$ concept C, there exists a tree-shaped interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ such that $\varepsilon \in C^{\mathbf{I}}$.*

Proof. We have to extend the structural induction argument contained in the previous proof with a further case.

When $C = \forall u^*.D$, let M be the domain of μ. By construction, every individual in M is reachable from $\mu(v)$ and therefore, by definition, if $v \in C^{\mathbf{I}}$ then $M \subseteq D^{\mathbf{I}}$; by inductive hypothesis, $N = D^{\mathbf{J}}$ and therefore $v \in (\forall u^*.D)^{\mathbf{I}}$. □

We can now show that propagated concepts give us enough expressive power to apply a construction called *TBox internalization*, first described in [Baa91].

Theorem 2.5. (TBox Internalization) *Given a TBox $\mathcal{T}$ expressed in a sublanguage of $\mathcal{ALCI}$, let $C_{\mathcal{T}}$ be the concept defined as follows:*

$$C_{\mathcal{T}} = \bigsqcap_{C_i \sqsubseteq D_i \text{ in } \mathcal{T}} \neg C_i \sqcup D_i$$

The following results hold:

- TBox satisfiability: *$\mathcal{T}$ is satisfiable iff the concept $\forall u^*.C_{\mathcal{T}}$ is satisfiable;*
- Concept consistency: *a concept $\hat{C}$ is consistent in $\mathcal{T}$ iff the concept $\hat{C} \sqcap \forall u^*.C_{\mathcal{T}}$ is satisfiable.*

Proof (sketch). By the previous lemma, we know that $a \in \forall u^*.C_{\mathcal{T}}$ iff every individual in $\Delta_{\mathbf{I}}$ models $C_{\mathcal{T}}$. By the definition of $C_{\mathcal{T}}$, this means that every individual $a' \in \Delta_{\mathbf{I}}$ models every subconcept $\neg C_i \sqcup D_i$, and therefore models all the inclusion assertions in $\mathcal{T}$: therefore, by definition, $\mathbf{I}$ models $\mathcal{T}$ itself. □

When combined with Theorem 2.4, these results allow us to restrict our attention to tree-like models and, therefore, make it possible to use automata-based techniques to determine satisfiability and consistency.

Corollary 2.6. *Given a TBox $\mathcal{T}$ expressed in a sublanguage of $\mathcal{ALCI}$, let $C_{\mathcal{T}}$ be the concept previously defined. The following results hold:*

- TBox satisfiability: $\mathcal{T}$ *is satisfiable iff there exists a tree-like interpretation* $\mathbf{I} = (\Delta_\mathbf{I}, \cdot^\mathbf{I})$ *such that* $(C_\mathcal{T})^\mathbf{I} = \Delta_\mathbf{I}$ *;*
- Concept consistency: *a concept* C *is consistent in* $\mathcal{T}$ *iff there exists a tree-like interpretation* $\mathbf{I} = (\Delta_\mathbf{I}, \cdot^\mathbf{I})$ *such that* $(C_\mathcal{T})^\mathbf{I} = \Delta_\mathbf{I}$ *and* $\varepsilon \in C^\mathbf{I}$.

Proof. According to Theorem 2.4, if the concept $C \sqcap \forall u^*.C_\mathcal{T}$ is satisfiable then there exists a tree-like interpretation $\mathbf{I} = (\Delta_\mathbf{I}, \cdot^\mathbf{I})$ such that $\varepsilon \in (C \sqcap \forall u^*.C_\mathcal{T})^\mathbf{I}$. Since a tree-like interpretation is by definition connected (i.e., there exists a path between any two individuals in $\Delta_\mathbf{I}$), it follows that $\varepsilon \in (\forall u^*.C_\mathcal{T})^\mathbf{I}$ iff every individual is in $(C_\mathcal{T})^\mathbf{I}$. □

Chapter 3

Automata for Description Logics

As first shown in [CDGL99], automata-based techniques similar to those used in other fields of logic (see [VW07]) can be applied to a broad family of Description Logics thanks to the tree model property.

In this chapter, we will present a strategy for deciding various reasoning tasks over DL TBoxes. We will show how to build an automaton that recognizes an encoding of tree-shaped models; as a consequence, the reasoning problem at hand can be reduced to checking whether the language recognized by the automaton is empty.

3.1 Automata for $\mathcal{ALC}$ Concepts

Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox and $\hat{C}$ be a $\mathcal{ALC}$ concept defined over the same DL signature $(\mathcal{C}, \mathcal{R})$. Let $C_{\mathcal{T}}$ be the concept defined in Theorem 2.5:

$$C_{\mathcal{T}} = \mathbf{nnf}\left(\bigsqcap_{C \sqsubseteq D \in \mathcal{T}} \neg C \sqcup D\right)$$

Let $\Sigma = \{ \sigma_{\mathbf{ne}} \} \cup 2^{\mathcal{C}}$, where $\sigma_{\mathbf{ne}}$ is a fresh symbol.

Let **sub**, **exs**, and $\mathbf{idx}_{\hat{C} \sqcap \forall u^*.C_{\mathcal{T}}}$ (from now on referred to as **idx**) be the functions defined in Section 2.3; let $k = |\mathbf{exs}(\hat{C} \sqcap \forall u^*.C_{\mathcal{T}})|$. Given a tree-shaped interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ which is subconcept ordered with respect to the the function **idx**, we define $\mathbf{treeEnc}(\mathbf{I}) = ([1:k]^*, \tau)$ as follows:

- for every $v \in \Delta_{\mathbf{I}}$, let $\tau(v) = \left\{ A \in \mathcal{C} \ : \ v \in A^{\mathbf{I}} \right\}$;
- for every $v \in [1:k]^* \setminus \Delta_{\mathbf{I}}$, let $\tau(v) = \sigma_{\mathbf{ne}}$.

We define the function $\mathbf{all} : \mathcal{R} \to 2^{[1:k]}$ as follows:

$$\mathbf{all}(P) = \left\{ i \ : \ \mathbf{idx}^{-1}(i) \text{ has the form } \exists P.C \right\}$$

i.e., $\mathbf{all}(P)$ is a set containing the indexes of all the existential subconcepts that "talk about" the role P.

Let $\mathbf{Aut}(\mathcal{T}, \hat{C}) = (\Sigma, Q, \hat{C}, k, \delta)$ be an alternating looping automaton, where Σ is the labeling alphabet previously defined, $Q = \{ q_0, q_{\mathbf{tbox}}, q_{\mathbf{ne}} \} \cup \mathbf{sub}(\hat{C}) \cup \mathbf{sub}(C_{\mathcal{T}})$ (where $\mathbf{sub}(C)$ is the set containing all the subconcepts of C, including C itself).

The transition function δ is defined by structural induction over its first argument:

$$\begin{aligned}
\delta(q_0, c) &= (0, \hat{C}) \wedge (0, q_{\mathbf{tbox}}) \\
\delta(q_{\mathbf{tbox}}, c) &= (0, C_{\mathcal{T}}) \ \wedge \bigwedge_{1 \leqslant i \leqslant k} (i, q_{\mathbf{tbox}}) \vee (i, q_{\mathbf{ne}}) \\
\delta(q_{\mathbf{ne}}, \sigma_{\mathbf{ne}}) &= \bigwedge_{1 \leqslant i \leqslant k} (i, q_{\mathbf{ne}})
\end{aligned}$$

$$
\begin{aligned}
\delta(\top, c) &= \textbf{true} \\
\delta(\bot, c) &= \textbf{false} \\
\delta(A, c) &= \textbf{true} \quad \text{when } A \in c \\
\delta(\neg A, c) &= \textbf{true} \quad \text{when } A \notin c \\
\delta(C \sqcap C', c) &= (0, C) \wedge (0, C') \\
\delta(C \sqcup C', c) &= (0, C) \vee (0, C') \\
\delta(\exists P.C, c) &= (\textbf{idx}(\exists P.C), C) \\
\delta(\forall P.C, c) &= \bigwedge_{i \in \textbf{all}(P)} (i, q_{\textbf{ne}}) \vee (i, C)
\end{aligned}
$$

where c is a symbol from $\Sigma \setminus \{\sigma_{\textbf{ne}}\}$, A is an atomic concept from $\mathcal{C}$, P is a role from $\mathcal{R}$, C and C' are subconcepts of $\hat{C}$ and every nonspecified entry is assumed to be equal to **false**.

Intuitively, a run of this automaton is composed of two threads of execution. The first thread, starting from the state $\hat{C}$, checks whether the root of the tree is a model of the given concept; the second thread, starting from the state $q_{\textbf{tbox}}$, checks whether the current node is a model of $C_{\mathcal{T}}$ and propagates itself to all the "real" child nodes (those not denoted by the $\sigma_{\textbf{ne}}$ symbol).

We claim the following results:

Theorem 3.1. *Given an $\mathcal{ALC}$ TBox $\mathcal{T}$, an $\mathcal{ALC}$ concept $\hat{C}$ defined over the same signature and a tree-like interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ subconcept ordered wrt* **idx***, let $T = (N, \tau) =$ **treeEnc**$(\mathbf{I})$ be the tree encoding of $\mathbf{I}$. Given a node $v \in N$ and a concept $C \in$ **sub**$(\hat{C}) \cup$ **sub**$(C_{\mathcal{T}})$, the automaton* **Aut**$(\mathcal{T}, \hat{C}) = (\Sigma, Q, \hat{C}, k, \delta)$ *admits a partial run over T starting from (v, C) iff $v \in C^{\mathbf{I}}$.*

Proof. We will use structural induction to show that an automaton starting in

the configuration (v, C) admits a partial run iff $v \in C^{\mathbf{I}}$. In the rest of this proof, we will use the shorthand PR_C^v to denote a partial run of A over T starting in the configuration (v, C), where v is a word from N and C is a state from Q.

The base cases can easily be shown to hold: if C is an atomic concept, $\delta(C, \tau(v))$ is satisfiable iff $C \in \tau(v)$, which by construction only holds iff $v \in C^{\mathbf{I}}$; if $C = \neg A$, where A is an atomic concept, $\delta(C, \tau(v))$ is satisfiable iff $A \notin \tau(v)$, which by construction only holds iff $v \notin A^{\mathbf{I}}$ or, equivalently, $v \in C^{\mathbf{I}}$.

The various iteration steps can be shown to hold as follows:

- $C = D \sqcap E$ By inductive hypothesis, $v \in C^{\mathbf{I}}$ iff $\mathbf{Aut}(\mathcal{T}, \hat{C})$ admits a partial run $PR_D^v = (N_D^v, \rho_D^v)$ and a partial run $PR_E^v = (N_E^v, \rho_E^v)$. If $v \in C^{\mathbf{I}}$, a partial run $PR_C^v = (N_C^v, \rho_C^v)$ can therefore be built as follows:

$$N_C^v = \{ 0 \cdot v \,:\, v \in N_D^v \} \cup \{ 1 \cdot v \,:\, v \in N_E^v \}$$

$$\rho_C^v(\varepsilon) = (v, C)$$

$$\rho_C^v(c \cdot v') = \begin{cases} \rho_D^v(v') & \text{if } c = 0 \\ \rho_E^v(v') & \text{if } c = 1 \end{cases}$$

  On the other hand, if $v \notin C^{\mathbf{I}}$ then $\mathbf{Aut}(\mathcal{T}, \hat{C})$ does not admit a partial run starting from (v, C), since such a run would necessarily contain the prefixed version of two partial runs for (v, D) and (v, E). By inductive hypothesis, this would imply that $v \in D^{\mathbf{I}} \cap E^{\mathbf{I}} = C^{\mathbf{I}}$, causing a contradiction.

- $C = D \sqcup E$ By inductive hypothesis, $v \in C^{\mathbf{I}}$ iff $\mathbf{Aut}(\mathcal{T}, \hat{C})$ admits a partial run PR_D^v for (v, D), a partial run PR_E^v for (v, E) or both. If $v \in C^{\mathbf{I}}$,

a partial run $\mathrm{PR}^{\nu}_{C} = (\mathrm{N}^{\nu}_{C}, \rho^{\nu}_{C})$ can therefore be built as follows:

$$(\mathrm{N}^{\nu}_{X}, \tau^{\nu}_{X}) = \begin{cases} \mathrm{PR}^{\nu}_{D} & \text{if } \nu \in D^{\mathbf{I}} \\ \mathrm{PR}^{\nu}_{E} & \text{otherwise} \end{cases}$$

$$\mathrm{N}^{\nu}_{C} = \{\, \varepsilon \,\} \cup \{\, 0 \cdot \nu \,:\, \nu \in \mathrm{N}^{\nu}_{X} \,\}$$

$$\rho^{\nu}_{C}(\varepsilon) = (\nu, C)$$

$$\rho^{\nu}_{C}(0 \cdot \nu') = \rho^{\nu}_{X}(\nu')$$

On the other hand, if $\nu \notin C^{\mathbf{I}}$ then $\mathbf{Aut}(\mathcal{T}, \hat{C})$ does not admit a partial run starting from (ν, C), since such a run would necessarily contain the prefixed version of a partial run from (ν, D) or (ν, E). By inductive hypothesis, this would imply that $\nu \in D^{\mathbf{I}} \cup E^{\mathbf{I}} = C^{\mathbf{I}}$, causing a contradiction.

- $C = \exists R.D$ Let $i = \mathbf{idx}(C)$; by construction, $\nu \in C^{\mathbf{I}}$ iff $\nu \cdot i \in D^{\mathbf{I}}$; by inductive hypothesis, $\nu \cdot i \in D^{\mathbf{I}}$ iff $\mathbf{Aut}(\mathcal{T}, \hat{C})$ admits a partial run $\mathrm{PR}^{\nu \cdot i}_{D}$ for $(\nu \cdot i, D)$. A partial run $\mathrm{PR}^{\nu}_{D} = (\mathrm{N}^{\nu}_{D}, \rho^{\nu}_{D})$ can therefore be built as follows:

$$\mathrm{N}^{\nu}_{D} = \{\, \varepsilon \,\} \cup \left\{\, 0 \cdot \nu \,:\, \nu \in \mathrm{N}^{\nu \cdot i}_{D} \,\right\}$$

$$\rho^{\nu}_{D}(\varepsilon) = (\nu, C)$$

$$\rho^{\nu}_{D}(0 \cdot \nu') = \rho^{\nu \cdot i}_{D}(\nu')$$

On the other hand, if $\nu \notin C^{\mathbf{I}}$ then $\mathbf{Aut}(\mathcal{T}, \hat{C})$ does not admit a partial run starting from (ν, C), since such a run would necessarily contain the prefixed version of a partial run from $(\nu \cdot i, D)$. By inductive hypothesis, this would imply that $\nu \cdot i \in D^{\mathbf{I}}$ and therefore $\nu \in C^{\mathbf{I}}$, causing a contradiction.

- $C = \forall R.D$ By construction, $\nu \in C^{\mathbf{I}}$ iff, for every $i \in \mathbf{all}(C)$, $\nu \cdot i$ is either undefined (i.e., $\tau(\nu \cdot i) = \sigma_{\mathbf{ne}}$) or is an individual in $D^{\mathbf{I}}$. By inductive hypothesis, the condition can be restated as follows: $\nu \in C^{\mathbf{I}}$ iff, for every $i \in \mathbf{all}(C)$, $\mathbf{Aut}(\mathcal{T}, \hat{C})$ admits either a partial run $PR_D^{\nu \cdot i}$ or a partial run $PR_{q_{\mathbf{ne}}}^{\nu \cdot i}$. A partial run $PR_D^{\nu} = (N_D^{\nu}, \rho_D^{\nu})$ can therefore be built as follows:

$$(N_{\mathbf{tmp}}, \rho_{\mathbf{tmp}}) = \begin{cases} PR_D^{\nu \cdot i} & \text{if } \nu \cdot i \in D^{\mathbf{I}} \\ PR_{q_{\mathbf{ne}}}^{\nu \cdot i} & \text{otherwise} \end{cases}$$

$$N_D^{\nu} = \{ \varepsilon \} \cup \left\{ i \cdot \nu' \,:\, i \in \mathbf{all}(C), \nu' \in N_{\mathbf{tmp}} \right\}$$

$$\rho_D^{\nu}(\varepsilon) = (\nu, C)$$

$$\rho_D^{\nu}(i \cdot \nu') = \rho_{\mathbf{tmp}}(\nu')$$

On the other hand, if $\nu \notin C^{\mathbf{I}}$ then $\mathbf{Aut}(\mathcal{T}, \hat{C})$ does not admit a partial run starting from (ν, C), since such a run would contain, for each $i \in \mathbf{all}(C)$, a prefixed version of a partial run from either $(\nu \cdot i, D)$ or $(\nu \cdot i, q_{\mathbf{ne}})$. By inductive hypothesis this would imply that, for every $i \in \mathbf{all}(C)$, $\nu \cdot i$ is either undefined or a member of $D^{\mathbf{I}}$, and therefore $\nu \in C^{\mathbf{I}}$, causing a contradiction.

□

Theorem 3.2. *Given an $\mathcal{ALC}$ TBox $\mathcal{T}$, an $\mathcal{ALC}$ concept $\hat{C}$ and a tree interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ over the same signature, let $T = \mathbf{treeEnc}(\mathbf{I})$. The automaton $\mathbf{Aut}(\mathcal{T}, \hat{C})$ accepts T iff $C_{\mathcal{T}} = \Delta_{\mathbf{I}}$ and $\varepsilon \in \hat{C}^{\mathbf{I}}$.*

Proof. The automaton will admit a partial run starting from $(\varepsilon, q_{\mathbf{tbox}})$ iff, for every node ν such that $\tau(\nu) \neq \sigma_{\mathbf{ne}}$, the automaton either admits a partial run

starting from $(\nu, C_{\mathcal{T}})$ or a partial run starting from $(\nu, q_{\mathbf{ne}})$. By the previous theorem, this is equivalent to stating that the automaton admits a partial run starting from $(\varepsilon, q_{\mathbf{tbox}})$ iff the two conditions hold:

$$\forall \nu \in [1:k]^*.\ \tau(\nu) = \sigma_{\mathbf{ne}} \vee \nu \in (C_{\mathcal{T}})^{\mathbf{I}}$$

$$\forall \nu, \nu' \in [1:k]^*.\ \tau(\nu) = \sigma_{\mathbf{ne}} \rightarrow \tau(\nu \cdot \nu') = \sigma_{\mathbf{ne}}$$

If $\mathbf{I}$ satisfies these conditions, it is clearly a model of $\mathcal{T}$; as a consequence, the automaton will allow a partial run starting from $(\varepsilon, q_{\mathbf{tbox}})$ iff $\mathbf{I} \models \mathcal{T}$. By the previous theorem, $\mathbf{Aut}(\mathcal{T}, \hat{C})$ will admit a partial run over T starting from $(\varepsilon, \hat{C})$ iff $\varepsilon \in \hat{C}^{\mathbf{I}}$, and therefore $\mathbf{Aut}(\mathcal{T}, \hat{C})$ will admit a run over T iff $\mathbf{I}$ is a model of $\mathcal{T}$ where $\varepsilon \in \hat{C}^{\mathbf{I}}$. □

This result enables us to claim the following theorem: we can reduce the problem of concept satisfiability to an emptiness check of an alternating looping automaton.

Theorem 3.3. *An $\mathcal{ALC}$ concept $\hat{C}$ is consistent in an $\mathcal{ALC}$ TBox T if and only if the language accepted by the automaton $\mathbf{Aut}(\mathcal{T}, \hat{C})$ is not empty.*

As a consequence, we can claim the following corollaries.

Corollary 3.4. *An $\mathcal{ALC}$ TBox $\mathcal{T}$ is satisfiable if and only if the language accepted by the automaton $\mathbf{Aut}(\mathcal{T}, \top)$ is not empty.*

Corollary 3.5. *An $\mathcal{ALC}$ TBox $\mathcal{T}$ implies an inclusion assertion $C \sqsubseteq D$ if and only if the language accepted by the automaton $\mathbf{Aut}(\mathcal{T}, C \sqcap \neg D)$ is not empty.*

3.2 Automata for $\mathcal{ALCI}$ Concepts

The introduction of inverse roles makes it possible to model complex scenarios in a more "natural" way. We must somehow change our construction to take into account the fact that now we must be able to follow roles in both directions. Intuitively, when talking about tree models, this results in "upwards" transitions: as we have seen in Section 2.2, two-way alternating automata can accomplish this task.

We will therefore reformulate the constructions of Section 3.1 in a way that takes into account the new requirements imposed by the presence of inverse roles.

Let $\mathcal{T}$ be an $\mathcal{ALCI}$ TBox and $\hat{C}$ be a $\mathcal{ALCI}$ concept over the same DL signature $(\mathcal{C}, \mathcal{R})$. Let $C_{\mathcal{T}}$ be the concept defined in Theorem 2.5, and let Σ be a labeling alphabet defined as follows:

$$\Sigma = \{\, \sigma_{\mathbf{ne}} \,\} \cup (\{\, R_\varepsilon \,\} \cup \mathcal{R}^{\pm}) \times 2^{\mathcal{C}}$$

where $\mathcal{R}^{\pm} = \{\, P, P^- \,:\, P \in \mathcal{R} \,\}$, and $\sigma_{\mathbf{ne}}$, R_ε are fresh symbols.

Let **sub**, **exs**, and $\mathbf{idx}_{\hat{C} \sqcap \forall u^*.C_{\mathcal{T}}}$ (from now on referred to as **idx**) be the functions defined in Section 2.3; let $k = |\mathbf{exs}(\hat{C}) \cup \mathbf{exs}(C_{\mathcal{T}})|$.

Given a tree-shaped interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ which is subconcept ordered with respect to the function **idx**, we define $\mathbf{treeEnc}'(\mathbf{I}) = ([1:k]^*, \tau)$ as follows:

- Let $\mathbf{roleAt} : \Delta_{\mathbf{I}} \to \mathcal{R}^{\pm}$ be a function such that $\mathbf{roleAt}(\varepsilon) = R_\varepsilon$ and $\mathbf{roleAt}(v) = R$ iff $\mathbf{idx}^{-1}(\mathbf{last}(v))$ is a concept of the form $\exists R.C$.
- Let $\mathbf{atomsAt}(v) = \left\{\, A \in \mathcal{C} \,:\, v \in A^{\mathbf{I}} \,\right\}$.

- For every $v \in \Delta_{\mathbf{I}}$, let $\tau(v) = (\mathbf{roleAt}(v), \mathbf{atomsAt}(v))$.
- For every $v \in [1:k]^* \setminus \Delta_{\mathbf{I}}$, let $\tau(v) = \sigma_{\mathbf{ne}}$.

This differs from the previous construction in that we now need to keep track of the role with which we reached the current node. We will store it as the first component of the label, using the special symbol R_ε to mark the root node.

Similarly as before, we define the function $\mathbf{all} : \mathcal{R}^{\pm} \to 2^{[1:k]}$ as follows:

$$\mathbf{all}(R) = \{\, i \;:\; \mathbf{idx}^{-1}(i) \text{ has the form } \exists R.C \,\}$$

i.e., $\mathbf{all}(R)$ is a set containing the indexes of all the existential subconcepts that "talk about" the role R.

Let $\mathbf{Aut}'(\mathcal{T}, \hat{C}) = (\Sigma, Q, q_0, k, \delta)$ be a two-way alternating looping automaton, where Σ is the labeling alphabet previously defined, and the set of states Q is defined as follows:

$$Q = \{\, q_0, q_{\mathbf{tbox}}, q_{\mathbf{ne}}, q_{\mathbf{wf}} \,\} \cup \{\, P, P^- \;:\; P \in \mathcal{R} \,\} \cup \mathbf{sub}(\hat{C}) \cup \mathbf{sub}(C_{\mathcal{T}})$$

Intuitively, a run of the automaton will be constructed by three independent "threads" of execution. The partial run starting in the state $q_{\mathbf{wf}}$ will check that the tree labels are "well formed", by forcing every non-$\sigma_{\mathbf{ne}}$ child node to have the correct label according to the function **roleAt**. The partial runs starting in the states $q_{\mathbf{tbox}}$ and $\hat{C}$ serve the same purpose as in the previous construction.

The transition function δ is defined by structural induction over its first argument:

$$\delta(q_0, (R_\varepsilon, c)) = (0, \hat{C}) \wedge (0, q_{\mathbf{tbox}}) \wedge (0, q_{\mathbf{wf}})$$

$$\delta(q_{\mathbf{wf}}, (r, c)) = \bigwedge_{1 \leqslant i \leqslant k} (i, q_{\mathbf{ne}}) \vee \big((i, \mathbf{roleAt}(i)) \wedge (i, q_{\mathbf{wf}})\big)$$

$$\delta(q_{\mathbf{tbox}}, (r, c)) = (0, C_{\mathcal{T}}) \ \wedge \bigwedge_{1 \leqslant i \leqslant k} (i, q_{\mathbf{ne}}) \vee (i, q_{\mathbf{tbox}})$$

$$\delta(q_{\mathbf{ne}}, \sigma_{\mathbf{ne}}) = \bigwedge_{1 \leqslant i \leqslant k} (i, q_{\mathbf{ne}})$$

$$\delta(\top, (r, c)) = \mathbf{true}$$

$$\delta(\bot, (r, c)) = \mathbf{false}$$

$$\delta(A, (r, c)) = \mathbf{true} \quad \text{when } A \in c$$

$$\delta(\neg A, (r, c)) = \mathbf{true} \quad \text{when } A \notin c$$

$$\delta(R, (R, c)) = \mathbf{true}$$

$$\delta(C \sqcap C', (r, c)) = (0, C) \wedge (0, C')$$

$$\delta(C \sqcup C', (r, c)) = (0, C) \vee (0, C')$$

$$\delta(\exists R.C, (r, c)) = (\mathbf{idx}(\exists R.C), C)$$

$$\delta(\forall R.C, (R^{-}, c)) = (-1, C) \wedge \bigwedge_{i \in \mathbf{all}(R)} (i, q_{\mathbf{ne}}) \vee (i, C)$$

$$\delta(\forall R.C, (r, c)) = \bigwedge_{i \in \mathbf{all}(R)} (i, q_{\mathbf{ne}}) \vee (i, C) \qquad \text{with } r \neq R^{-}$$

where (r, c) is a symbol from $\Sigma \setminus \{\sigma_{\mathbf{ne}}\}$, A is an atomic concept from $\mathcal{C}$, R is a role from $\mathcal{R}^{\pm}$, C and C' are subconcepts of $\hat{C}$ and every nonspecified entry is assumed to be equal to **false**.

We claim the following results:

Theorem 3.6. *Given an $\mathcal{ALCI}$ TBox $\mathcal{T}$, an $\mathcal{ALCI}$ concept $\hat{C}$ defined over the same signature and a tree-like interpretation $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ over the signature of $\hat{C}$, let $T = \mathbf{treeEnc}'(\mathbf{I})$ be the tree built according to the previously described construction; $\mathbf{Aut}'(\mathcal{T}, \hat{C})$ accepts T iff $\varepsilon \in \hat{C}^{\mathbf{I}}$.*

Proof. The proof is very similar to the one for Theorem 3.2: we will use structural induction to show that an automaton starting in the configuration (v, C) admits a partial run iff $v \in C^{\mathcal{I}}$.

First of all, we will note that the definition of $\delta(q_{\mathbf{wf}}, \cdot)$ guarantees that the role labeling of the tree is consistent with the constraints imposed by the construction:

- the root is labeled with a pair whose first component is R_ε;
- if $i = \mathbf{idx}(\exists R.D)$, the i-th child of a given node is either labeled with $q_{\mathbf{ne}}$ or with a pair that has R as the first component.

For the states in $\mathbf{sub}(\hat{C}) \cup \mathbf{sub}(C_{\mathcal{T}})$, we can use an inductive argument similar to the one in the proof of Theorem 3.1. The only induction case that differs from the previous construction can be shown to hold as follows:

- $C = \forall R.D$ and $\tau(v) = (R^-, c)$ By construction, $v \in C^{\mathcal{I}}$ iff $\mathbf{init}(v) \in D^{\mathcal{I}}$ and, for every $i \in \mathbf{all}(C)$, $v \cdot i$ is either undefined (i.e., $\tau(v \cdot i) = \sigma_{\mathbf{ne}}$) or is an individual in $D^{\mathcal{I}}$. By inductive hypothesis, the condition can be restated as follows:

  1. $\mathbf{Aut}'(\mathcal{T}, \hat{C})$ admits a partial run starting from $(\mathbf{init}(v), D)$;
  2. for every $i \in \mathbf{all}(C)$, $\mathbf{Aut}'(\mathcal{T}, \hat{C})$ admits either a partial run $PR_E^{v \cdot i}$ or a partial run $PR_{q_{\mathbf{ne}}}^{v \cdot i}$.

  A partial run $PR_D^w = (N_D^w, \rho_D^w)$ can therefore be built as follows:

$$(N_X^{v \cdot i}, \rho_X^{v \cdot i}) = \begin{cases} PR_E^{v \cdot i} & \text{if } v \cdot i \in D^{\mathcal{I}} \\ PR_{q_{\mathbf{ne}}}^{v \cdot i} & \text{otherwise} \end{cases}$$

$$N_D^w = \{\varepsilon\} \cup \left\{ 0 \cdot \nu' \,:\, \nu' \in N_D^{\mathbf{init}(\nu)} \right\} \cup \left\{ i \cdot \nu' \,:\, i \in \mathbf{all}(C), \nu' \in N_X^{\nu \cdot i} \right\}$$

$$\rho_D^w(\varepsilon) = (\nu, C)$$

$$\rho_D^w(0 \cdot \nu') = \rho_D^{\mathbf{init}(\nu)}(\nu')$$

$$\rho_D^w(i \cdot \nu') = \rho_X^{\nu \cdot i}(\nu')$$

On the other hand, if $\nu \notin C^I$ then $\mathbf{Aut}(C)$ does not admit a partial run starting from (ν, C), since such a run would contain a prefixed version of a partial run from $(\mathbf{init}(\nu), D)$ and, for each $i \in \mathbf{all}(C)$, a prefixed version of a partial run from either $(\nu \cdot i, D)$ or $(\nu \cdot i, q_{\mathbf{ne}})$. By inductive hypothesis this would imply that $\mathbf{init}(\nu) \in D^I$ and, for every $i \in \mathbf{all}(C)$, $\nu \cdot i$ is either undefined or a member of D^I: this would imply that $\nu \in C^I$, causing a contradiction.

□

Once again, we can reduce the problem of concept satisfiability to an emptiness check of an alternating looping automaton.

Theorem 3.7. *An $\mathcal{ALCI}$ concept $\hat{C}$ is consistent in an $\mathcal{ALCI}$ TBox $\mathcal{T}$ if and only if the language accepted by the automaton $\mathbf{Aut}'(\mathcal{T}, \hat{C})$ is not empty.*

As a consequence, we can claim the following corollaries.

Corollary 3.8. *An $\mathcal{ALCI}$ TBox $\mathcal{T}$ is satisfiable if and only if the language accepted by the automaton $\mathbf{Aut}'(\mathcal{T}, \top)$ is not empty.*

Corollary 3.9. *An $\mathcal{ALCI}$ TBox $\mathcal{T}$ implies an inclusion assertion $C \sqsubseteq D$ if and only if the language accepted by the automaton $\mathbf{Aut}'(\mathcal{T}, C \sqcap \neg D)$ is not empty.*

3.3 Automata for $\mathcal{ALC}^{f}$ Concepts

The Description Logic $\mathcal{ALC}^{f}$ is a nonstandard extension of $\mathcal{ALC}$ which permits a limited usage of functionality restrictions (see the definitions given in Section 2.3). An $\mathcal{ALC}^{f}$ TBox over a signature $(\mathcal{C}, \mathcal{R})$ can contain two kind of assertions:

- *inclusion assertions* of the form $C \sqsubseteq D$, where C and D are $\mathcal{ALC}$ concept expressions over the signature $(\mathcal{C}, \mathcal{R})$;
- *global functionality assertions* of the form $\top \sqsubseteq (\leqslant 1\, P.\, \top)$, often abbreviated as $(\textbf{func}\ P)$, where P is an atomic role from $\mathcal{R}$.

We are interested in $\mathcal{ALC}^{f}$ as it is exactly the language supported by the current prototype of the *TreeHug* reasoner, whose inner workings we will discuss at length in Chapter 6.

Consider an arbitrary be an $\mathcal{ALC}^{f}$ TBox $\mathcal{T}$ over a signature $(\mathcal{C}, \mathcal{R})$. Let $\tilde{\mathcal{T}}$ be the TBox obtained by removing all the global functional assertions from $\mathcal{T}$, and let $\mathcal{R}_f$ be the set of functional roles in $\mathcal{T}$:

$$\tilde{\mathcal{T}} = \{\, C_i \sqsubseteq D_i \in \mathcal{T} \ :\ C_i, D_i \text{ are } \mathcal{ALC} \text{ concepts} \,\}$$

$$\mathcal{R}_f = \{\, P \ :\ (\textbf{func}\ P) \in \mathcal{T} \,\}$$

Clearly, $\tilde{\mathcal{T}}$ is an $\mathcal{ALC}$ TBox. Let $C_{\tilde{\mathcal{T}}}$ be the concept obtained from the internalization of $\tilde{\mathcal{T}}$ (see Theorem 2.5).

We will claim the following result:

Lemma 3.10. *The concept $\hat{C}$ is consistent in the $\mathcal{ALCI}^{\mathrm{f}}$ TBox $\mathcal{T}$ iff there exists a tree model for $\hat{C} \sqcap \forall u^*.C_{\tilde{\mathcal{T}}}$ such that, for each role $P \in \mathcal{R}_{\mathrm{f}}$, $P^{\mathbf{I}}$ is a functional relation (i.e., every individual in $\Delta_{\mathbf{I}}$ has at most one P-successor).*

Proof. Any model $\mathbf{I} = (\Delta_{\mathbf{I}}, \cdot^{\mathbf{I}})$ for $\mathcal{T}$ where $(\hat{C})^{\mathbf{I}} \neq \emptyset$ can be transformed into a tree model by following an unraveling procedure similar to the one used in the proof of Theorem 2.4.

On the other hand, a tree model for $\hat{C} \sqcap \forall u^*.C_{\tilde{\mathcal{T}}}$ where all the roles in $\mathcal{R}_{\mathrm{f}}$ are functional is obviously a model of $\mathcal{T}$, and therefore a witness of the consistency of $\hat{C}$ in $\mathcal{T}$. □

Fix an arbitrary bijection $\mathbf{ridx} : \mathcal{R}_{\mathrm{f}} \to [1 : |\mathcal{R}_{\mathrm{f}}|]$. Let **sub**, **exs** and $\mathbf{idx}_C$ (for any concept C) be the functions defined in Section 2.3. For every concept C, let $\mathbf{exs}^{\mathcal{R}_{\mathrm{f}}}(C)$ be the set of all the existential subconcepts that do not refer to a functional role:

$$\mathbf{exs}^{\mathcal{R}_{\mathrm{f}}}(C) = \{\, \exists P.D \in \mathbf{exs}(C) \;:\; P \notin \mathcal{R}_{\mathrm{f}} \,\}$$

Without loss of generality, assume that every function $\mathbf{idx}_C$ enjoys the following property: for any concept $\exists P.D$ in $\mathbf{exs}(C)$ with $P \notin \mathcal{R}_{\mathrm{f}}$, $\mathbf{idx}_C(\exists P.D) \leqslant |\mathbf{exs}^{\mathcal{R}_{\mathrm{f}}}(C)|$. Let $\mathbf{idx}_C^{\mathcal{R}_{\mathrm{f}}}$ be a function defined as follows:

$$\mathbf{idx}_C^{\mathcal{R}_{\mathrm{f}}}(\exists P.D) = \begin{cases} \mathbf{idx}_C(\exists P.D) & \text{if } P \notin \mathcal{R}_{\mathrm{f}} \\ |\mathbf{exs}^{\mathcal{R}_{\mathrm{f}}}(C)| + \mathbf{ridx}_{\mathcal{R}_{\mathrm{f}}}(P) & \text{otherwise} \end{cases}$$

We will use this new "lookup function" analogously to how we used **idx** in the previous constructions. The main difference lies in the fact that, this time, all the existential subconcepts dealing with a given functional role must be

satisfied by a single successor. This is reflected by the fact that, for any two subconcepts of the form $\exists P.D$ and $\exists P.D'$, if $P \in \mathcal{R}_f$ then $\mathbf{idx}_C^{\mathcal{R}_f}(\exists P.D) = \mathbf{idx}_C^{\mathcal{R}_f}(\exists P.D')$.

Let $\hat{C}$ be an $\mathcal{ALC}^f$ concept defined over the same signature of $\mathcal{T}$. The results obtained in Section 2.3.3 can easily be extended to $\mathcal{ALC}^f$, showing therefore that $\hat{C}$ is consistent in $\mathcal{T}$ iff there exists an interpretation $\mathbf{I} = (\Delta_I, \cdot^I)$ which enjoys the following properties:

- The set Δ_I is a tree;
- For every concept $P \in \mathcal{R}_f$, P^I is a functional relation;
- The root node ε belongs to $\hat{C} \sqcap \forall u^*.C_{\tilde{\mathcal{T}}}$.

Let $\mathbf{idx}_f$ be a shorthand for the function $\mathbf{idx}^{\mathcal{R}_f}_{\hat{C}\sqcap\forall u^*.C_{\tilde{\mathcal{T}}}}$. Given an interpretation $\mathbf{I} = (\Delta_I, \cdot^I)$ such that $\mathbf{I} \models \mathcal{T}$ and $\varepsilon \in \hat{C}^I$, we will build a tree model $\mathbf{J} = (\Delta_J, \cdot^J)$ by rearranging sibling nodes such that the following properties hold:

- For any $v \in \Delta_J$ and every concept $\exists P.D \in \mathbf{exs}^{\mathcal{R}_f}(\hat{C} \sqcap \forall u^*.C_{\tilde{\mathcal{T}}})$ with $P \notin \mathcal{R}_f$, if $v \in (\exists P.D)^I$ then $(v, v{\cdot}\mathbf{idx}(\exists P.D)) \in P^I$ and $v{\cdot}\mathbf{idx}(\exists P.D) \in D^I$;
- For any role $P \in \mathcal{R}_f$ and any pair $(v, v{\cdot}j) \in P^I$, $j = |\mathbf{exs}^{\mathcal{R}_f}(C)| + \mathbf{ridx}_{\mathcal{R}_f}(P)$.

The second property follows from the fact that P^I is functional, and therefore any given node will have at most one P-successor.

We will call models which enjoy these properties *subconcept ordered with respect to the function* $\mathbf{idx}_f$. Note that this idea slightly differs from the previous definition of subconcept ordering: intuitively, this time we take into account

the fact that all the existential subconcepts that "talk about" a functional role $P \in \mathcal{R}_f$ will be satisfied by a single successor.

Let $k = |\mathbf{exs}^{\mathcal{R}_f}(C)| + |\mathcal{R}_f|$. Let $\Sigma = \{\sigma_{\mathbf{ne}}\} \cup 2^{\mathcal{C}}$, where $\sigma_{\mathbf{ne}}$ is a fresh symbol. We define the function $\mathbf{all}_f : \mathcal{R} \to 2^{[1:k]}$ as follows:

$$\mathbf{all}_f(P) = \{i \ : \ \exists\, C.\ \mathbf{idx}_f(\exists P.C) = i\}$$

i.e., $\mathbf{all}_f(P)$ is a set containing the indexes of all the existential subconcepts that "talk about" the role P – only, this time we refer to the function $\mathbf{idx}_f$.

Let $\mathbf{Aut}_f(\tilde{\mathcal{T}}, \mathcal{R}_f, \hat{C}) = (\Sigma, Q, \hat{C}, k, \delta)$ be an alternating looping automaton, where Σ is the previously defined alphabet, $Q = \{q_0, q_{\mathbf{tbox}}, q_{\mathbf{ne}}\} \cup \mathbf{sub}(\hat{C}) \cup \mathbf{sub}(C_{\tilde{\mathcal{T}}})$ and δ behaves *almost* exactly like the transition function defined in Section 3.1. We only change the definition of δ in two cases:

$$\delta(\exists P.C, c) = (\mathbf{idx}_f(\exists P.C), C)$$
$$\delta(\forall P.C, c) = \bigwedge_{i \in \mathbf{all}_f(P)} (i, q_{\mathbf{ne}}) \vee (i, C)$$

i.e., we now use our modified "lookup function" $\mathbf{idx}_f$. We claim the following result:

Theorem 3.11. *Let* $\mathbf{J} = (\Delta_{\mathbf{J}}, \cdot^{\mathbf{J}})$ *be a tree-shaped interpretation which is subconcept-ordered with respect to the function* $\mathbf{idx}_f$. *Let* $T = \mathbf{treeEnc}(\mathbf{J})$ *(where* **treeEnc** *is the function defined in Section 3.1). The automaton* $\mathbf{Aut}_f(\tilde{\mathcal{T}}, \mathcal{R}_f, \hat{C})$ *accepts* T *iff* $C_{\mathcal{T}} = \Delta_{\mathbf{I}}$ *and* $\varepsilon \in \hat{C}^{\mathbf{I}}$.

The proof is virtually identical to the one for the $\mathcal{ALC}$ case. Just like in the case of $\mathcal{ALC}$ and $\mathcal{ALCI}$, we have all the tools we need to reduce decision problems

on $\mathcal{ALC}^{\mathtt{f}}$ TBoxes to decision problems on tree automata, as expressed in the following corollaries.

Corollary 3.12. *An $\mathcal{ALC}^{\mathtt{f}}$ TBox $\mathcal{T}$ is satisfiable if and only if the language accepted by the automaton $\mathbf{Aut}_{\mathtt{f}}(\tilde{\mathcal{T}}, \mathcal{R}_{\mathtt{f}}, \top)$ is not empty.*

Corollary 3.13. *An $\mathcal{ALC}^{\mathtt{f}}$ TBox $\mathcal{T}$ implies an inclusion assertion $\mathsf{C} \sqsubseteq \mathsf{D}$ if and only if the language accepted by the automaton $\mathbf{Aut}_{\mathtt{f}}(\tilde{\mathcal{T}}, \mathcal{R}_{\mathtt{f}}, \mathsf{C} \sqcap \neg \mathsf{D})$ is not empty.*

Chapter 4

Automata Decision Procedures

In the previous chapter we have shown a way to reduce DL reasoning tasks to emptiness checking problems over tree automata; the "missing piece" is an algorithm that decides efficiently whether the language accepted by an automaton is empty or not.

In this chapter we will introduce a family of algorithms developed with a strong focus on simplicity and implementability. Notably, these algorithms manage to avoid the main drawback of automata-based decision procedures: all the objects created during the intermediate steps can be constructed lazily and disposed early.

Note that in the rest of this chapter we will make extensive use of the acronyms introduced in Section 2.2: *2-ALT* for two-way alternating looping tree automata, *ALT* for (one-way) alternating looping tree automata, *ZLT* for the nonstandard family of zero-free looping tree automaton, and *NLT* for nondeterministic looping tree automata.

We will begin by showing an algorithm that decides whether the language

recognized by a NLT is empty. In the following section we will describe an algorithm which removes zero-transitions from an alternating automata, therefore transforming a zero-layered ALT into a ZLT; this will pave the way for the next section, where we introduce an algorithm that, starting from a ZLT, builds a NLT that recognizes the same language. As a consequence, we will be able to decide emptiness of a zero-layered ALT by deciding the emptiness of the equivalent NLT.

We will conclude the chapter by showing a procedure that, starting from a zero-layered 2-ALT, builds a zero-layered ALT that accepts an annotated version of the trees accepted by the first automaton. As a consequence, we will be able to reuse the previously introduced chain of reductions to decide emptiness of zero-layered 2-ALTs.

4.1 Emptiness of an NLT

Let $A_{\mathbf{n}} = (\Sigma, \Omega, \omega_0, k, \delta_{\mathbf{n}})$ be a nondeterministic looping automaton over Σ-labeled k-ary trees. We define the function $\mathbf{step} : 2^{\Omega} \to 2^{\Omega}$ as follows:

$$\mathbf{step}(\Omega') = \left\{ \omega \in \Omega' \ : \ \exists\, \vec{\omega}' \in (\Omega')^k, c \in \Sigma.\ \mathbf{makeSet}(\vec{\omega}') \models \delta_{\mathbf{n}}(\omega, c) \right\}$$

where $\mathbf{makeSet}((\omega'_1, \ldots, \omega'_k)) = \left\{ (1, \omega'_1), \ldots, (k, \omega'_k) \right\}$. Intuitively, given a set $\Omega' \subseteq \Omega$, the set $\mathbf{step}(\Omega')$ contains all the states in Ω' for which the transition function can be possibly satisfied by using only states in Ω'.

Example 4.1. Let $A_{\mathbf{n}} = (\Sigma, Q, q_0, k, \delta)$, where the components of the tuple are defined as follows:

$$\Sigma = \{ a, b \}$$

$$\Omega = \{ q_0, \omega_a, \omega_b \}$$

$$k = 2$$

$$\delta(q_0, a) = (1, \omega_a) \wedge (2, \omega_b)$$

$$\delta(\omega_a, a) = (1, \omega_a) \wedge (2, \omega_a)$$

$$\delta(\omega_b, b) = (1, \omega_b) \wedge (2, \omega_b)$$

We can see that $q_0 \notin \textbf{step}(\{ q_0, \omega_a \})$, as neither $\delta(q_0, a)$ nor $\delta(q_0, b)$ can be satisfied by the set $[1:2] \times \{ q_0, \omega_a \}$. On the other hand, ω_a is clearly contained in $\textbf{step}(\{ \omega_a \})$, since $\delta(\omega_a, a)$ can be satisfied by $\{ (1, \omega_a), (2, \omega_a) \}$. ■

Let $\Omega_{\textbf{fix}}$ be the greatest fixed point of **step**, i.e. the biggest subset of Ω such that $\textbf{step}(\Omega_{\textbf{fix}}) = \Omega_{\textbf{fix}}$. It is clearly the case that, for any two sets $\Omega', \Omega'' \in 2^{\Omega}$, if $\Omega' \subseteq \Omega''$ then $\textbf{step}(\Omega') \subseteq \textbf{step}(\Omega'')$: the function **step** is monotone increasing. We can therefore apply the Knaster-Tarski fixed point theorem, which states that the greatest fixed point $\Omega_{\textbf{fix}}$ is guaranteed to exist and can be calculated as follows:

$$\Omega_{\textbf{fix}} = \textbf{step}^{|\Omega|}(\Omega) = \underbrace{(\textbf{step} \circ \cdots \circ \textbf{step})}_{|\Omega| \text{ times}}(\Omega)$$

Example 4.2. *(cont. from Example 4.1)* In this case it is easy to check that $\textbf{step}(\Omega) = \Omega$, and therefore $\Omega_{\textbf{fix}} = \Omega$. ■

We can claim the following theorem:

Theorem 4.1. *The language recognized by an automaton* $\mathcal{A}_{\textbf{n}} = (\Sigma, Q, q_0, k, \delta)$ *is not empty iff* $q_0 \in \textbf{step}^{|\Omega|}(\Omega)$.

Algorithm 4.1 Construction of a non-emptiness witness for a NLT

let $N_T := \emptyset$, $N_R := \{\varepsilon\}$, $\rho(\varepsilon) := (\varepsilon, q_0)$
while $N_R \setminus N_T \neq \emptyset$ **do**
  let $\hat{N} := N_R \setminus N_T$
  for each v **in** $\hat{N}$ **do**
    let $N_T := N_T \cup \{v\}$
    let $\omega_w := \mathbf{second}(\rho(v))$
    let $\tau(v) := \mathbf{sym}(\omega_w)$
    for each (i, ω_i) **in** $\mathbf{adv}(\omega_w)$ **do**
      let $N_R := N_R \cup \{v \cdot i\}$
      let $\rho(v \cdot i) := (v \cdot i, \omega_i)$
    end for
  end for
end while
return $T = (N_T, \tau)$, $R = (N_R, \rho)$

Proof. (⇐ direction) Given the set $\Omega_{\mathbf{fix}} = \mathbf{step}^{|\Omega|}(\Omega)$ and knowing that $q_0 \in \Omega_{\mathbf{fix}}$, we will build a tree that is recognized by $A_{\mathbf{n}}$. For every $\omega \in \Omega_{\mathbf{fix}}$, we define the following functions:

- $\mathbf{sym}(q)$ is a symbol $c \in \Sigma$ such that $\delta(q, c)$ is satisfiable;
- $\mathbf{adv}(q)$ is a set of the form $\{(1, \omega_1), \ldots, (k, \omega_k)\} \subseteq [1 : k] \times \Omega_{\mathbf{fix}}$ that satisfies $\delta(q, \mathbf{sym}(q))$.

These objects are guaranteed to exist by hypothesis. Moreover, from the definition of NLT it follows that a number in $[1 : k]$ can appear at most once as an index in $\mathbf{adv}(q)$.

Let $T = (N_T, \tau)$ be a Σ-labeled k-tree and $R = (N_R, \rho)$ be a $(N_T \times \Omega)$-labeled tree constructed using the **sym** function to determine the labeling, and the

adv function to determine the successors of the current node, as described in Algorithm 4.1. It is easy to check that N_R is a valid run tree for $A_{\mathbf{n}}$ over T:

- $\rho(\varepsilon) = (\varepsilon, q_0)$;
- for every $r \in N_R$, with $\rho(r) = (\nu, q)$, the labeling of the child nodes satisfies $\delta(q, \tau(\nu))$ by virtue of the definition of the functions **sym** and **adv**.

Since $A_{\mathbf{n}}$ recognizes T, the language $\mathcal{L}(A_{\mathbf{n}})$ is obviously non-empty.

(⇒ direction) Let $T = (N_T, \tau)$ be a tree recognized by $A_{\mathbf{n}}$, and let $R = (N_R, \rho)$ be an accepting run of $A_{\mathbf{n}}$ over T; let Ω' be the set defined as follows:

$$\Omega' = \{\, q \;:\; \exists\, r \in N_R, i \in \mathbb{N}.\ \rho(r) = (i, \omega) \,\}$$

First of all, we will show that $\mathbf{step}(\Omega') = \Omega'$. Every state $\omega \in \Omega'$ appears at least once in the run tree, so there exists a node $r \in N_R$ such that $\rho(r) = (\nu, \omega)$. Since R is a run tree, the labeling of the child nodes of r must satisfy $\delta(\omega, \tau(\nu))$. By the construction of Ω', every child node of r is labeled with a state from Ω', and therefore $q \in \mathbf{step}(\Omega')$. This entails that $\Omega' \subseteq \mathbf{step}(\Omega')$; by definition, though, $\mathbf{step}(\Omega') \subseteq \Omega'$, and therefore $\mathbf{step}(\Omega') = \Omega'$.

Clearly, $\mathbf{step}^{|\Omega|}(\Omega') = \Omega'$. Since **step** is a monotone increasing function and $\Omega' \subseteq \Omega$, it follows that $\mathbf{step}^{|\Omega|}(\Omega') \subseteq \mathbf{step}^{|\Omega|}(\Omega)$ and, therefore, $\Omega' \subseteq \mathbf{step}^{|\Omega|}(\Omega)$. The root node of R is labeled with the initial state, therefore $\omega_0 \in \Omega'$; it follows that $q_0 \in \mathbf{step}^{|\Omega|}(\Omega)$. □

4.2 Removing Zero Transitions

As mentioned in Section 2.2, an ALT $A_{\mathbf{a}} = (\Sigma, Q, q_0, k, \delta)$ is said to be *zero-layered* if there exists a bijection $\mathbf{depth} : Q \to [1 : |Q|]$ such that, for any two states q, q' and any symbol c, if $\mathbf{depth}(q) \leqslant \mathbf{depth}(q')$ then the atom $(0, q')$ does not appear in the PBF $\delta(q, c)$.

We will start by claiming the following theorem:

Theorem 4.2. *All the automata resulting from the constructions described in Chapter 3 are zero-layered.*

Proof. The initial state q_0 does not appear in any value of the transition function, and can therefore be assigned the maximal depth $|Q|$. The atom $(0, q_{\mathbf{tbox}})$ only appears in the transition function for q_0: $q_{\mathbf{tbox}}$ can therefore be assigned the depth $|Q| - 1$. Analogously, for the case of $\mathcal{ALCI}$ automata, the atom $(0, q_{\mathbf{wf}})$ only appears in the transition function for q_0, and can be assigned the depth $|Q| - 2$. The state $q_{\mathbf{ne}}$ never appears in a zero-transition, and can therefore be assigned any arbitrary depth.

The other states are all DL concepts. The transition formula for a concept C only includes atoms whose second component is a subconcept of C. These states can therefore be ordered such that, for any two concepts C and D, if D is a subconcept of C then $\mathbf{depth}(D) < \mathbf{depth}(C)$. □

Given a zero-layered alternating looping automaton $A_{\mathbf{a}} = (\Sigma, Q, q_0, k, \delta)$, we define a function $\delta_0 : Q \times \Sigma \to \mathbf{PBF}([1 : k] \times Q)$ and, for every state q and every symbol c, a substitution $\sigma_{q,c}$:

$$\sigma_{q,c} = \left\{ \big((0, q'),\ \delta_0(q', c)\big) \ :\ q' \in Q,\ \mathbf{depth}(q') < \mathbf{depth}(q) \right\}$$

$$\delta_0(q, c) = \delta(q, c)\sigma_{q,c}$$

Informally, the substitution $\sigma_{q,c}$ collapses all the zero-transitions involving states "less deep" than q, by replacing every atom of the form $(0, q')$ with the PBF that specifies the transition for q' with respect to the current input character.

The definition given above is well-formed and removes all the zero-indexed atoms: since $A_{\mathbf{a}}$ is zero-layered, an atom of the form $(0, q')$ can only appear in δ if the depth of q' is smaller than the depth of q. As a consequence, the value of $\delta_0(q, c)$ only depends on the value of δ_0 for the states having a depth smaller than the depth of q. It is therefore possible to compute δ_0 by applying the substitution $\sigma_{q,c}$ to the state with depth 1, and proceding up to the state with depth $|Q|$. The automaton $A_{\mathbf{z}} = (\Sigma, Q, q_0, k, \delta_0)$ is therefore zero-free – i.e., it does not contain zero-transitions.

In order to help us with the proof, we will claim the two following lemmas.

Lemma 4.3. *Given a state-symbol pair* (q, c) *and a set* $\Gamma \subseteq [1 : k] \times Q$*, if* Γ *satisfies* $\delta_0(q, c)$ *there exists a set* $Q^+(\Gamma) \subseteq Q$ *such that:*

1. *the set* $\Gamma^+ = \Gamma \cup (\{0\} \times Q^+(\Gamma))$ *satisfies* $\delta(q, c)$*;*
2. *for each* $q' \in Q^+(\Gamma)$*,* Γ *satisfies* $\delta_0(q', c)$*.*

Proof. Starting from a set Γ that satisfies $\delta_0(q, c)$, we will construct a set $Q^+(\Gamma)$ that satisfies the given conditions.

Without loss of generality, suppose that $\delta(q, c)$ is a formula in disjunctive normal form (it is always possible to obtain a formula in DNF which is equivalent to a given formula). Let therefore $\delta(q, c) = \Theta_1 \vee \cdots \vee \Theta_n$, where each Θ_i is

a conjunction of the form $\omega_{i,1} \wedge \cdots \wedge \omega_{i,m_i}$, and each $\omega_{i,j}$ is an atom from $[1 : k] \times Q$.

Moreover, each formula Θ_i can be thought of as being made by two separate formulae $\Psi_i \wedge \Phi_i$, such that $\Psi_i = (0, q_{i,1}) \wedge \cdots \wedge (0, q_{i,h_i})$ is a conjunction of all the atoms in Θ_i whose first component is zero, and Φ_i is a conjunction of all the other atoms.

By definition, $\delta_0(q, c) = \delta(q, c)\sigma_{q,c}$. By applying the substitution to the formula above, we obtain that

$$\delta_0(q, c) = \Theta_1\sigma_{q,c} \vee \cdots \vee \Theta_n\sigma_{q,c}$$

Since Γ satisfies $\delta_0(q, c)$, there must exist a subformula $\Theta_i\sigma_{q,c}$ which is satisfied by Γ. Let $Q^+(\Gamma)$ be the set of states that appear in atoms of Θ_i whose first component is 0:

$$Q^+(\Gamma) = \{\, q' \in Q \;:\; (0, q') \text{ appears in } \Theta_i \,\}$$

We can rewrite Θ_i as follows:

$$\Theta_i = \Theta_i' \wedge \bigwedge_{q' \in Q^+(\Gamma)} (0, q')$$

where Θ_i' is a conjunction of all the atoms in Θ_i whose first component is not zero. By the definition of $\sigma_{q,c}$, we can infer that

$$\Theta_i\sigma_{q,c} = \Theta_i' \wedge \bigwedge_{q' \in Q^+(\Gamma)} \delta_0(q', c)$$

Therefore, since Γ satisfies $\Theta_i\sigma_{q,c}$, it must also satisfy $\delta_0(q', c)$ for all the q'

in $Q^+(\Gamma)$. Moreover, since Γ satisfies Θ_i, it is easy to check that the set $\Gamma^+ = \Gamma \cup (\{0\} \times Q^+(\Gamma))$ satisfies $\delta(q, c)$, as the new set contains all the conjuncts from $\delta(q, c)$ that do not appear in $\delta_0(q, c)$. □

Let $T = (N, \tau)$ be a Σ-labeled k-tree for which $A_{\mathbf{a}}$ admits an accepting run $R = (N_R, \rho)$. We define the *zero-closure* of a node $r \in N_R$ as follows:

$$\textbf{zeroClos}(r) = \left\{ r' \in N_R \ : \begin{array}{l} r' = r \cdot c_1 \cdots c_n \ \wedge \ \rho_1(r') \neq \rho_1(r) \wedge \\ \forall i \in [1 : n-1].\ \rho_1(r \cdot c_1 \cdots c_i) = \rho_1(r) \end{array} \right\}$$

where $\rho_1(v)$ is the first component of $\rho(v)$. Intuitively, the zero-closure of a node in the run tree contains all the nodes that can be reached by successive zero-transitions. Let $Z : N_R \to [1, k] \times Q$ be a function defined as follows:

$$\textbf{zeroLabels}(r) = \left\{ (\textbf{last}(v'), q') \ : \ (v', q') \in \rho(r'),\ r' \in \textbf{zclos}(r) \right\}$$

Lemma 4.4. *For any node* r *in* N_R*, with* $\rho(r) = (q, v)$*, the set* **zeroLabels**(r) *satisfies* $\delta_0(q, \tau(v))$.

Proof. We will proceed by induction on the value of **depth**(q). If **depth**$(q) = 1$, the formula $\delta(q, \tau(v))$ cannot contain any atom whose first component is zero, and therefore $\delta_0(q, \tau(v)) = \delta(q, \tau(v))$. The set **zeroClos**$(r)$ will contain all the child nodes of r, whose labels satisfy by definition $\delta(q, \tau(v))$ and, therefore, also $\delta_0(q, \tau(v))$.

If **depth**$(q) = n$, the formula $\delta(q, \tau(v))$ can contain atoms whose first component is zero and whose second component is a state q', with **depth**$(q') < n$. Let $L = \{(i_1, q_1), \ldots, (i_m, q_m)\}$ be a set that satisfies $\delta(q, \tau(v))$, such that for every $j \in [1, m]$ there exists a child node r^j of r having $\rho(r^j) = (v \cdot i_j, q_j)$; R is

a run tree, therefore this set is guaranteed to exist.

For every j such that $i_j = 0$, we know that $\mathbf{depth}(q_j) < n$: as a consequence, by inductive hypothesis, the set $\mathbf{zeroLabels}(r^j)$ satisfies $\delta_0(q_j, \tau(\nu))$. Clearly, if L satisfies $\delta(q, \tau(\nu))$ then the set $(L \setminus \{ (0, q_j) \}) \cup \mathbf{zeroLabels}(r^j)$ satisfies the formula $\delta(q, \tau(\nu))\big[(0, q_j) \,/\, \delta_0(q_j, \tau(\nu))\big]$. By repeating this substitution process for every atom of the form $(0, q_j)$, we can infer that the set $\mathbf{zeroLabels}(r)$ satisfies the formula $\delta(q, \tau(\nu))\sigma_{q,c} = \delta_0(q, \tau(\nu))$. □

Theorem 4.5. *Given a zero-layered alternating looping automaton, there exists a zero-free alternating looping automaton that accepts the same language.*

Proof. Consider a zero-layered ALT $A_{\mathbf{a}} = (\Sigma, Q, q_0, k, \delta)$ and the resulting zero-free ALT $A_{\mathbf{z}} = (\Sigma, Q, q_0, k, \delta_0)$; let $T = (N_T, \tau)$ be a Σ-labeled k-tree. We will show that $A_{\mathbf{a}}$ accepts T iff $A_{\mathbf{z}}$ accepts T.

(⇒ direction) Let $R = (N_R, \rho)$ be a run tree for $A_{\mathbf{a}}$ over T; we can obtain a run tree $R^0 = (N^0_R, \rho^0)$ for $A_{\mathbf{z}}$ over T by starting from R and collapsing all the zero-transitions according to the following procedure:

1. Let **zeroClos**, **zeroLabels** be the functions defined in the proof of Lemma 4.4;
2. Let $N^0_R = \{ \varepsilon \}$;
3. Let ν be a mapping from N^0_R to N_R, and let $\nu(\varepsilon) = \varepsilon$;
4. For every $r \in N^0_R$ where $\rho^0(r)$ is not yet defined:
   - Let $\rho^0(r) = \rho(\nu(r))$;
   - For each $(i_j, q_j) \in \mathbf{zeroLabels}(\nu(r))$, add a node $r \cdot j$ to N^0_R and let $\nu(r \cdot j)$ be a node r' from $\mathbf{zeroClos}(\nu(r))$ such that $\rho(r')$ is equal to

$(\nu \cdot i_j, q_j)$ (one such node is guaranteed to exist by the definition of **zeroLabels**);

5. Repeat the previous step until a fixpoint is reached.

It is possible to check that, by virtue of Lemma 4.4, every "collapsed" run step in R^0 satisfies δ_0; also, by construction, $\rho^0(\varepsilon) = (\varepsilon, q_0)$. Therefore, R^0 is a valid run tree for $A_{\mathbf{z}}$ over T.

(⇐ direction) Let $R^0 = (N_R^0, \rho^0)$ be a run tree for $A_{\mathbf{z}}$ over T; we can obtain a run tree $R = (N_R, \rho)$ for $A_{\mathbf{a}}$ over T as follows:

1. Define the function **childLabels** : $N_R^0 \to [1 : k] \times Q$ as follows:

$$\textbf{childLabels}(r^0) = \left\{ (i, q') \;:\; \rho^0(r^0) = (\nu, q) \wedge \exists j.\ \rho^0(r^0 \cdot j) = (\nu \cdot i, q') \right\}$$

2. Let $N_R = \{ \varepsilon \}$;

3. Let μ be a mapping from N_R to $Q \times N_R^0$, and let $\mu(\varepsilon) = (q_0, \varepsilon)$;

4. For every $r \in N_R$ where $\rho(r)$ is not yet defined:

   - Let $(q, r^0) = \mu(r)$; let $(\nu^0, q^0) = \rho^0(r^0)$;
   - Let $\rho(r) = (\nu^0, q)$;
   - Let $\Gamma =$ **childLabels**(r^0), and let $Q^+(\Gamma)$ be the set that satisfies the conditions specified in Lemma 4.3;
   - For each node $r^0 \cdot j \in N_R^0$, with $\rho^0(r^0 \cdot j) = (\nu_j^0, q_j^0)$, add a node $r \cdot j$ to N_R and let $\mu(r \cdot j) = (q_j^0, r^0 \cdot j)$;
   - For each state $q^+ \in Q^+(\Gamma)$, add a (fresh) node $r \cdot h$ to N_R and let $\mu(r \cdot h) = (q^+, r^0)$;

5. Repeat the previous step until a fixpoint is reached.

It is possible to check that, by virtue of Lemma 4.3 every run step in the "expanded" run R satisfies δ; also, by construction, $\rho(\varepsilon) = (\varepsilon, q_0)$. Therefore, R is a valid run tree for $A_{\mathbf{a}}$ over T. □

4.3 Reducing ALT emptiness to NLT emptiness

Let $A_{\mathbf{z}} = (\Sigma, Q, q_0, k, \delta_0)$ be a zero-free alternating automaton on Σ-labeled k-trees. Let $\mathbf{group} : 2^{([1:k]\times Q)} \to (2^Q)^k$ be a function defined as follows:

$$\mathbf{group}(\Gamma) = \{\, (i, \mathbf{collect}(\Gamma, i)) \;:\; i \in [1:k] \,\}$$

where $\mathbf{collect} : 2^{([1:k]\times Q)} \times [1:k] \to (2^Q)$ is a function defined as follows:

$$\mathbf{collect}(\Gamma, i) = \{\, q \in Q \;:\; (i, q) \in \Gamma \,\}$$

For instance, supposing that $k = 3$:

$$\mathbf{group}\Big(\big\{(1, q_a), (1, q_b), (3, q_c)\big\}\Big) = \Big\{\big(1, \{\, q_a, q_b \,\}\big), \big(2, \emptyset\big), \big(3, \{\, q_c \,\}\big)\Big\}$$

Let $\delta_{\mathbf{n}} : 2^Q \times \Sigma \to \mathbf{PBF}([1:k] \times Q)$ be a function defined as follows:

$$\delta_{\mathbf{n}}(\emptyset, c) = \mathbf{true}$$

$$\delta_{\mathbf{n}}(\omega, c) = \bigvee_{\Gamma_j \models \bigwedge_{q\in\omega} \delta_0(q,c)} \left(\bigwedge_{(j,\omega_i^j)\in \mathbf{group}(\Gamma_j)} (i, \omega_i^j) \right)$$

It is easy to verify that the automaton $A_{\mathbf{n}} = (\Sigma, 2^Q, \{q_0\}, k, \delta_{\mathbf{n}})$ is a nondeterministic looping automaton. We claim the two following lemmas:

Lemma 4.6. *Given a zero-free ALT* $A_{\mathbf{z}} = (\Sigma, Q, q_0, k, \delta)$*, let* $A_{\mathbf{n}}$ *be the automaton built according to the procedure described above. For any symbol* c *and any set of states* $\omega \in 2^Q$*, a set* $\Gamma \subseteq [1:k] \times Q$ *models the formula* $\bigwedge_{q\in\omega} \delta_0(q,c)$ *iff the set* $\mathbf{group}(\Gamma)$ *models the formula* $\delta_{\mathbf{n}}(\omega, c)$*.*

Proof. If $\omega = \emptyset$, by convention $\bigwedge_{q\in\omega} \delta_0(q,c) = \mathbf{true}$; also, by definition, $\delta_{\mathbf{n}}(\omega,c) = \mathbf{true}$, and therefore the lemma trivially holds. We will therefore consider the case where $\omega \neq \emptyset$.

If $\mathbf{group}(\Gamma)$ is a model of $\delta_{\mathbf{n}}(\omega,c)$, it must satisfy one of the disjuncts in its definition; this means that there exists a set Γ' such that $\Gamma' \models \bigwedge_{q\in\omega} \delta_0(q,c)$ and $\mathbf{group}(\Gamma') \subseteq \mathbf{group}(\Gamma)$. As a consequence, since $\bigwedge_{q\in\omega} \delta_0(q,c)$ does not contain any negated atom by definition, $\Gamma \models \bigwedge_{q\in\omega} \delta_0(q,c)$.

If $\Gamma \models \bigwedge_{q\in\omega} \delta_0(q,c)$, there exists a disjunct in $\delta_{\mathbf{n}}(\omega,c)$ which has the form

$$\bigwedge_{(i,\omega_i)\in\mathbf{group}(\Gamma)} (i,\omega_i)$$

The set $\mathbf{group}(\Gamma)$ obviously satisfies this disjunct, and therefore it also satisfies $\delta_{\mathbf{n}}(\omega,c)$. □

Lemma 4.7. *A NLT accepts a tree* $T = (N_T, \tau)$ *iff there exists a run tree* $R' = (N'_R, \rho')$ *such that:*

1. $N'_R \subseteq N_T$*;*
2. *for every node* $v \in N'_R$*, the first component of* $\rho'(v)$ *is* v*.*

Proof. By definition, for any state-symbol pair (ω, c) the formula $\delta_{\mathbf{n}}(\omega, c)$ can be satisfied by sets of the form $\{ (1, \omega'_1), \ldots, (k, \omega'_k) \}$. A minimal run tree will therefore have a branching degree of at most k.

Moreover, any node in T will be visited at most once: the automaton moves strictly forward, and every formula arising from its transition function can be satisfied by a set which does not contain atoms with repeated indexes.

As a consequence, given an arbitrary run tree, it is possible to create a new run tree that satisfies the conditions above by just rearranging nodes. □

We claim the following theorem:

Theorem 4.8. *Given a zero-free alternating looping automaton, there exists a nondeterministic looping automaton that accepts the same language.*

Proof. We will show that a zero-free ALT $A_{\mathbf{z}} = (\Sigma, Q, q_0, k, \delta)$ accepts a tree iff the nondeterministic automaton $A_{\mathbf{n}} = (\Sigma, 2^Q, \{ q_0 \}, k, \delta_{\mathbf{n}})$ also accepts it.

(⇒ direction) Let $T = (N_T, \tau)$ be a Σ-labeled k-tree, and $R = (N_R, \rho)$ be a run tree for $A_{\mathbf{z}}$ over T; we can construct a run tree $R' = (N_T, \rho')$ for $A_{\mathbf{n}}$ as follows:

$$\rho'(v) = \big(v,\ \{ q \in Q\ :\ \exists r.\ \rho(r) = (v, q) \}\big)$$

Without loss of generality, we will suppose that R is a minimal run tree (as defined in Subsection 2.2.1). It can be easily shown that, if R is a minimal run tree of a zero-free alternating automaton, the depth of a node in the run tree is equal to the length of the first component of its label: we will do so with an inductive argument. The empty string is the only string having length 0, and

$\rho(\varepsilon) = (\varepsilon, q_0)$: therefore, the statement clearly holds for $|r| = 0$. For every node $r \in N_R$, with $|r| = n$, let $\rho(r) = (v, q)$; by inductive hypothesis, $|v| = |r|$. Since the transition function does not contain any atom whose first component is -1 or 0, the formula $\delta_0(q, \tau(v))$ can be satisfied by labeling each child node $r \cdot j$ with a pair whose first component is a word of the form $v \cdot i$: clearly $|r \cdot j| = |v \cdot i|$, and therefore the statement holds.

As a consequence of the previous statement, the root of R' will be labeled with the pair $(\varepsilon, \{ q_0 \})$, satisfying therefore the first condition in the definition of a run tree.

In order to verify the second condition, fix an arbitrary node $v \in T$ with $\rho'(v) = (v, \omega)$; we have to prove that the formula $\delta_{\mathbf{n}}(\omega, \tau(v))$ is satisfied by the set $\Gamma' = \mathbf{lab}_{R'}(\omega)$, the labeling according to ρ' of the child nodes of ω.

For every state $q \in \omega$, there exists by construction a node $r_{v,q} \in N_R$ such that $\rho(r_{v,q}) = (v, q)$. Without loss of generality, we will suppose that, for every node $r'_{v,q}$ in R labeled with (v, q), $\mathbf{lab}_R(r'_{v,q}) = \mathbf{lab}_R(r_{v,q})$ (i.e., its child nodes have the same labeling as the child nodes of $r_{v,q}$). Let $\Gamma_q = \mathbf{lab}_R(r_{v,q})$, and $\Gamma = \bigcup_{q \in \omega} \Gamma_q$.

Since R is a valid run tree for $A_{\mathbf{z}}$ over T, we know that each set Γ_q satisfies $\delta_0(q, \tau(v))$; since $\delta_0(q, \tau(v))$ is a positive boolean formula (and therefore does not contain any negated atom), we can infer that $\Gamma \models \bigwedge_{q \in \omega} \delta_0(q, \tau(v))$

Let (i, q) be an index-state pair, and let ω_i be the set of states such that $(i, \omega_i) \in \Gamma'$: we will show that $q \in \omega_i$ iff $(i, q) \in \Gamma$ and, as a consequence, $\Gamma' = \mathbf{group}(\Gamma)$.

By construction, there exists a node $r \in N_R$ such that $(i, q) \in \mathbf{lab}_R(r)$. Re-

member that, by construction, the following condition holds:

$$\rho'(\nu \cdot i) = \big(\nu \cdot i,\ \{\, q' \in Q \ :\ \exists\, r.\ \rho(r) = (\nu \cdot i, q') \,\}\big)$$

Since q satisfies the conditions it is clearly included in the second component of $\rho'(\nu \cdot i)$, and therefore it appears in ω_i. Similarly, if (i, q) does not belong to Γ then q does not satisfy the condition $\exists\, r.\ \rho(r) = (\nu \cdot i, q')$, and therefore does not appear in ω_i.

Since $\Gamma' = \mathbf{group}(\Gamma)$, by Lemma 4.6 we can infer that $\Gamma' \models \delta_{\mathbf{n}}(\omega, \tau(\nu))$. As a consequence, R' also satisfies the second condition in the definition of run tree.

(⇐ direction) Let $T = (N_T, \tau)$ be a tree, and let $R' = (N'_R, \rho')$ be a run tree for $A_{\mathbf{n}}$ over T.

According to Lemma 4.7 we will suppose without loss of generality that $N'_R \subseteq N_T$ and that, for each word $\nu \in T$, the first component of $\rho'(\nu)$ is ν. Let $R = (N_R, \rho)$ be a tree built as follows:

- Let $\varepsilon \in N_R$, with $\rho(\varepsilon) = (\varepsilon, q_0)$.
- For each $r \in N_R$, with $\rho(r) = (\nu, q)$, let Γ_ν be a set defined as follows:

$$\Gamma_\nu = \{\, (i, q') \ :\ \rho'(\nu \cdot i) = (\nu \cdot i, \omega'),\ q' \in \omega' \,\}$$

  Let $(i_1, q_1), \ldots, (i_h, q_h)$ be an ordering of the elements of Γ_ν: for each $j \in [1 : h]$, let $r \cdot j \in N_R$ and $\rho(r \cdot j) = (\nu \cdot i_j, q_j)$.

Since $\rho(\varepsilon) = (\varepsilon, q_0)$, R clearly satisfies the first condition in the definition of run tree. In order to verify the second condition, fix an arbitrary node $r \in N_R$

with $\rho(r) = (\nu, q)$; let $\rho'(\nu) = (\nu, \omega)$. The set Γ_ν described above is constructed such that $\mathbf{group}(\Gamma_\nu)$ models $\delta_{\mathbf{n}}(\omega, \tau(\nu))$; by the previous Lemma 4.6, it follows that Γ_ν models $\delta_0(q', \tau(\nu))$ for each $q' \in \omega$. By construction $q \in \omega$, and therefore $\Gamma_\nu \models \delta_0(q, \tau(\nu))$: as a consequence, the set $R = (N_R, \rho)$ built as described above is a run tree for $A_{\mathbf{z}}$ over T. □

4.4 Reducing 2-ALT emptiness to ALT emptiness

The toolchain we have developed thus far allows us to check the emptiness of a zero-layered alternating looping automaton. In order to handle the same reasoning problems over TBoxes and concepts whose definitions make use of inverse roles, though, we need to extend our techniques to handle two-way alternating automata.

Let $A_{\mathbf{2}} = (\Sigma, Q, q_0, k, \delta)$ be a two-way alternating automaton over Σ-labeled k-trees: we will build a (one-way) alternating automaton that recognizes an annotated version of the language accepted by $A_{\mathbf{2}}$, and use the annotation to eliminate the need for upwards transitions.

Intuitively, a run of the new automaton over a given input tree is composed of two threads of execution:

- The first thread runs a modified version of $A_{\mathbf{2}}$, which checks the presence of a state q in the annotation of the current node instead of performing a $(-1, q)$ transition.
- The second thread checks that the annotation is consistent: whenever a child of the current node is annotated with a state q, the automaton admits a partial run starting from the current node in the state q.

Let $\Sigma' = \Sigma \times 2^Q$ be the new labeling alphabet used by the new automaton, and let Q' be a set defined as follows:

$$Q' = \{\, q'_0, q_\pi \,\} \cup \{\, (\overline{\mathbf{up}}\, q) \;:\; q \in Q \,\}$$

Let δ' be a function from $Q \times \Sigma'$ to the set $\mathbf{PBF}([1,k] \times Q')$ defined as follows:

$$\begin{aligned}
\delta'(q'_0, (c,z)) &= (0, q_\pi) \wedge (0, q_0) \\
\delta'(q_\pi, (c,z)) &= \bigwedge_{i \in [1,k]} (i, q_\pi) \wedge \bigwedge_{q \in Q} \left(\bigwedge_{i \in [1,k]} (i, (\overline{\mathbf{up}}\, q)) \vee (0, q) \right) \\
\delta'((\overline{\mathbf{up}}\, q), (c,z)) &= \begin{cases} \mathbf{true} & \mathbf{if}\ q \notin z \\ \mathbf{false} & \mathbf{otherwise} \end{cases} \\
\delta'(q, (c,z)) &= \delta(q,c)\sigma_z \quad \text{for each } q \in Q
\end{aligned}$$

where, for each $z \in 2^Q$, the substitution σ_z is defined as follows:

$$\begin{aligned}
\sigma_z = &\{\, (\,(-1, q'), \mathbf{true}\,) \;:\; q' \in z \,\} \cup \\
&\cup \{\, (\,(-1, q'), \mathbf{false}\,) \;:\; q' \in Q \setminus z \,\}
\end{aligned}$$

The substitution σ_z is used to get rid of -1 transitions by "trusting" the annotation: every transition of the form $(-1, q')$ with $q' \in z$ is considered to be satisfied; vice versa, any transition of the form $(-1, q')$ with $q' \notin z$ is assumed to fail.

The automaton $A_1 = (\Sigma', Q', q'_0, \delta', k)$ is clearly a one-way alternating automaton. We claim the following result.

Theorem 4.9. *The automaton A_2 accepts an input tree $T = (N_T, \tau)$ iff there exists an annotation $\zeta : N_T \to 2^Q$ such that the automaton A_1 accepts the tree*

$T' = (N_T, \tau')$, *with* $\tau'(v) = (\tau(v), \zeta(v))$ *for every node* $v \in N_T$.

Proof. (⇒ direction) Given a tree T and a minimal run $R = (N_R, \rho)$ for A_2 over T, we will build an annotation ζ and a run $R' = (N'_R, \rho')$ for A_1 over $T' = (N_T, \tau')$, with $\tau'(v) = (\tau(v), \zeta(v))$ for every $v \in N_T$.

A node $r \in N_R$ is an *inversion point* for R iff, for some v, i, q, q', $\rho(r) = (v, q)$ and $\rho(\mathbf{init}(r)) = (v \cdot i, q')$. Intuitively, an inversion point is the first node after an "upwards" transition of the form $(-1, q)$.

We will say that a node r is *reachable without inversions* from a node r' iff there exists a path from r' to r that does not include inversion points except, eventually, for r. Formally, r is reachable without inversions from r' iff $r = r'c_1 \dots c_n$ and, for all $i \in [1, n-1]$, the node $r'c_1 \dots c_i$ is not an inversion point.

Given a node $r \in N_R$, we define the tree $\mathbf{invReach}(R, r) = (N_{r\downarrow}, \rho_{r\downarrow})$ as follows:

- $N_{r\downarrow}$ is the set of all words r' such that the node $r \cdot r'$ is reachable without inversions from r;
- if $r \cdot r'$ is not an inversion point for R, then $\rho_{r\downarrow}(r') = \rho(r \cdot r')$;
- if $r \cdot r'$ is an inversion point for R, with $\rho(\mathbf{init}(r \cdot r')) = (v \cdot j, q')$ and $\rho(r \cdot r') = (v, q)$, then $\rho_{r\downarrow}(r') = (v \cdot j, (\mathbf{up}\ q))$.

Moreover, let $\mathbf{invReach_{pfx}}(R, r, r_{\mathbf{pfx}})$ be a pair $(N_{r\downarrow}^{r_{\mathbf{pfx}}}, \rho_{r\downarrow}^{r_{\mathbf{pfx}}})$ defined as follows:

$$N_{r\downarrow}^{\mathbf{pfx}} = \{\, r_{\mathbf{pfx}} \cdot r' \ :\ r' \in N_{r\downarrow} \,\}$$
$$\rho_{r\downarrow}^{\mathbf{pfx}}(r_{\mathbf{pfx}} \cdot r') = \rho_{r\downarrow}(r')$$

where $(N_{r\downarrow}, \rho_{r\downarrow}) = \textbf{invReach}(R, r)$.

Intuitively, the tree $IR^R_r = \textbf{invReach}(R, r)$ contains an "image" of all the nodes of R that are reachable without inversion from r. Namely, for each node r' of IR^R_r, the node $r \cdot r'$ is reachable without inversion from r in R and the labeling of r' in IR^R_r is equal to the labeling of $r \cdot r'$ in R.

The value of $\textbf{invReach}_{\textbf{pfx}}(R, r, r_{\textbf{pfx}})$ is a version of $\textbf{invReach}(R, r)$ where each node is prefixed with $r_{\textbf{pfx}}$: note that this structure is not a tree, since its "root" node is $r_{\textbf{pfx}}$.

Let ζ be the function defined as follows:

$$\zeta(\nu) = \left\{ q \in Q \ : \ \exists r, i, q'.\ \rho(\textbf{init}(r)) = (\nu \cdot i, q'),\ \rho(r) = (\nu, q) \right\}$$

Intuitively, $\zeta(\nu)$ contains all the states q such that, somewhere in the run R, there exists an inversion point labeled with the pair (ν, q).

Let $R' = (N'_R, \rho')$ be the tree defined as follows:

- The empty word ε belongs to N'_R, with $\rho'(\varepsilon) = (\varepsilon, q'_0)$.
- Let $(N^1_{\varepsilon\downarrow}, \rho^1_{\varepsilon\downarrow}) = \textbf{invReach}_{\textbf{pfx}}(R, \varepsilon, 1)$; for every node $r' \in N^1_{\varepsilon\downarrow}$, $r' \in N_R$ and $\rho(r') = \rho^1_{\varepsilon\downarrow}(r')$.
- Every node $\{ 2 \cdot \nu \ : \ \nu \in N_T \}$ belongs to N'_R, with $\rho'(2 \cdot \nu) = (\nu, q_\pi)$.
- For every node $\nu \in N_T$ and every state $q \in Q$, one of the following conditions holds:
  - If $q \notin \zeta(\nu)$, then N'_R contains k nodes $2 \cdot \nu \cdot i_1, \ldots, 2 \cdot \nu \cdot i_k$ whose

labeling is defined as follows:

$$\rho'(2 \cdot \nu \cdot i_j) = (\nu \cdot j, (\overline{\mathbf{up}}\, q))$$

– If $q \in \zeta(\nu)$, there exists (by construction of ζ) a node $\tilde{r} \in N_R$ with $\rho(\tilde{r}) = (\nu, q)$. In this case, there exists an index h such that the following conditions hold:

$$(\tilde{N}, \tilde{\rho}) = \mathbf{invReach_{pfx}}(R, \tilde{r}, 2 \cdot \nu \cdot h)$$
$$\tilde{N} \subseteq N'_R$$
$$\rho'(r') = \tilde{\rho}(r') \text{ for every } r' \in \tilde{N}$$

We now need to prove that R' is a run tree for A_1 over T. The labeling of the root node is obviously in accordance to the definition of run tree, and therefore the first condition is trivially satisfied.

In order to check the second condition we fix an arbitrary node $r \in N'_R$, with $\rho'(r) = (\nu, q)$, and check that the labeling of the child nodes $\mathbf{lab}'_R(r)$ satisfies $\delta'(q, \tau'(\nu))$. We distinguish five cases:

- $q = q'_0$ This state only occurs at the root of the run, with $\nu = \varepsilon$. The node 1, by construction, has the same labeling as the root node of $\mathbf{invReach}(R, \varepsilon)$. By definition, the root of $\mathbf{invReach}(R, \varepsilon)$ is ε, whose labeling is (ε, q_0). The node 2 is always labeled with (ε, q_π). The labeling of the child nodes therefore satisfies the transition condition.

- q is a state of the form $(\overline{\mathbf{up}}\, q')$ By construction, this only occurs when $q' \notin \zeta(\nu)$; therefore, $\delta'((\overline{\mathbf{up}}\, q'), \tau'(\nu)) = \mathbf{true}$.

- $q \in Q$ Let $\sigma_{\zeta(\nu)}$ be the substitution defined in the construction of δ'.

By construction, there exists a node $\tilde{r} \in N_R$ such that $\rho(\tilde{r}) = (v, q)$ and, for every atom $\gamma \in \mathbf{lab}_R(\tilde{r})$, the atom $\gamma\sigma_{\zeta(v)}$ belongs to $\mathbf{lab}_{R'}(r)$. Since $\mathbf{lab}_R(\tilde{r}) \models \delta(q, \tau(v))$, we can therefore infer that $\mathbf{lab}_{R'}(r)$ is a model of the formula $\delta'(q, \tau'(v)) = \delta(q, \tau(v))\sigma_{\zeta(v)}$.

- $q = q_\pi$ The formula $\delta'(q_\pi, \tau'(v))$ is the conjunction of a "propagation section", the subformula $\bigwedge_{i\in[1,k]}(i, q_\pi)$, and a "verification section", the subformulas of the form $\bigwedge_{i\in[1,k]}(i, (\overline{\mathbf{up}}\, q)') \vee (0, q')$ for every $q' \in Q$. By construction, for every $i \in [1 : k]$ there exists a child node of r labeled with $(v \cdot i, q_\pi)$; the propagation section is therefore satisfied. For every subformula $\bigwedge_{i\in[1,k]}(i, (\overline{\mathbf{up}}\, q)') \vee (0, q')$ belonging to the verification section, we can distinguish two cases:

  - For some $i \in [1 : k]$, $q' \in \zeta(v \cdot i)$. In this case, by construction, there exists a node $r \cdot h$ labeled with the pair (v, q'): the second disjunct is therefore satisfied.
  - For every $i \in [1 : k]$, $q' \notin \zeta(v \cdot i)$. In this case, by construction, there exist k nodes $r \cdot j_1, \ldots, r \cdot j_k$ such that $\rho'(r \cdot j_i) = (v \cdot i, (\overline{\mathbf{up}}\, q)')$; as a consequence, the first disjunct is satisfied.

  Since both the propagation section and all the subformulas in the verification section are satisfied by the labeling of the child nodes, the formula $\delta'(q_\pi, \tau'(v))$ is therefore satisfied as well.

(⇐ direction) Given a tree $T' = (N_T, \tau')$, with $\tau'(v) = (\tau(v), \zeta(v))$ for every $v \in N_T$, and a run $R' = (N'_R, \rho')$ for A_1 over $T = (N_T, \tau)$, we will build a run $R = (N_R, \rho)$ for A_2 over T.

Without loss of generality, suppose that R' is minimal; moreover, suppose that $\rho'(1) = (\varepsilon, q_0)$ and $\rho'(2) = (\varepsilon, q_\pi)$. Let R be a tree and μ be a mapping from

N_R to N'_R defined as follows:

- The empty word ε belongs to N_R, with $\rho(\varepsilon) = (\varepsilon, q_0)$ and $\mu(\varepsilon) = 1$.
- For every node $r \in N_R$, for every node r'_i of the form $\mu(r) \cdot i$ in N'_R, let $(\nu_i, q_i) = \rho'(r'_i)$. The node $r \cdot i$ belongs to N_R. We distinguish two cases:
  - If q_i is a state from Q, then $\mu(r \cdot i) = r' \cdot i$.
  - If q_i is a state of the form $(\mathbf{up}\, q')$, $\mu(r \cdot i)$ is a node of the form $2 \cdot \mathbf{init}(\nu_i) \cdot h$, with $\rho(\mu(r \cdot i)) = (\mathbf{init}(\nu_i), q)$; its existence is guaranteed by the definition of A_1.
- For every node $r \in N_R$, $\rho(r) = \rho'(\mu(r))$.

The tree R trivially satisfies the first condition in the definition of run tree. It is easy to check that it also satisfies the second condition: for an arbitrary node $r \in N_R$ with $\rho(r) = (\nu, q)$, every atom of the form $(0, (\mathbf{up}\, q'))$ in $\mathbf{lab}_{R'}(\mu(r))$ is substituted in $\mathbf{lab}_R(r)$ by the atom $(-1, q')$, "mirroring" the substitution $\sigma_{\zeta(\nu)}$. Since $\mathbf{lab}_{R'}(\mu(r))$ satisfies $\delta(q, \tau(\nu))\sigma_{\zeta(\nu)}$, it is clearly the case that $\mathbf{lab}_R(r)$ satisfies $\delta(q, \tau(\nu))$. □

It is easy to check that, if the original 2-ALT is zero-layered, the resulting ALT will preserve this property for two reasons:

- This construction does not introduce any new zero-transition on already existing states;
- Any ordering for the states in the 2-ALT can be extended to the additional states q'_0 and q_π by letting $\mathbf{depth}(q'_0) = |Q| + 2$, $\mathbf{depth}(q_\pi) = |Q| + 1$ (where $|Q|$ is the number of states of the 2-ALT).

4.5 Complexity Considerations

In this section, we will analyze the worst-case complexity behaviour of the proposed algorithms. The results show that our techniques match the known tight bounds for all the problems at hand.

In this section we will use the notation $|\delta|$ to refer to the maximum size of the formulas in the transition function of a given automaton. Formally, given an automaton $A = (\Sigma, Q, q_0, k, \delta)$, we define $|\delta|$ as follows:

$$|\delta| = \mathbf{max}_{(q,c)\in Q\times\Sigma}|\delta(q,c)|$$

4.5.1 Emptiness of a NLT

Consider a NLT $A_{\mathbf{n}} = (\Sigma, \Omega, \omega_0, k, \delta_{\mathbf{n}})$. We recall the definition of the function **step** given in Section 4.1:

$$\mathbf{step}(\Omega') = \left\{ \omega \in \Omega' \,:\, \exists\, \vec{\omega}' \in (\Omega')^k, c \in \Sigma.\ \mathbf{makeSet}(\vec{\omega}') \models \delta_{\mathbf{n}}(\omega, c) \right\}$$

where $\mathbf{makeSet}((\omega'_1, \ldots, \omega'_k)) = \left\{ (1, \omega'_1), \ldots, (k, \omega'_k) \right\}$. We will show that $\Omega_{\mathbf{fix}} = \mathbf{step}^{|\Omega|}(\Omega)$ can be computed in polynomial time with respect to the size of Ω, Σ and δ.

Theorem 4.10. *Given a nondeterministic looping tree automaton $A_{\mathbf{n}} = (\Sigma, \Omega, \omega_0, k, \delta_{\mathbf{n}})$, it is possible to check the emptiness of $\mathcal{L}(A_{\mathbf{n}})$ in a number of steps polynomial in $|\Omega|$, $|\Sigma|$ and $|\delta_{\mathbf{n}}|$.*

Proof. By Theorem 4.1, $\mathcal{L}(A_{\mathbf{n}})$ is empty iff $\omega_0 \notin \mathbf{step}^{|\Omega|}(\Omega)$.

Algorithm 4.2 Emptiness Checking for NLT Automata

```
function InStep(A_n, ω, Ω')
    let (Σ, Ω, ω_0, k, δ_n) := A_n
    for each ω⃗' in (Ω')^k do
        for each c in Σ do
            if makeSet(ω⃗') ⊨ δ(ω, c) then
                return true
            end if
        end for
    end for
    return false
end function

function IsEmpty(A_n)
    let (Σ, Ω, ω_0, k, δ_n) := A_n
    let Ω' := Ω
    for i := 1 to |Ω| do
        let Ω' := { ω ∈ Ω' : InStep(A_n, q, Ω') }
    end for
    return ω_0 ∈ Ω'
end function
```

Let us consider the problem of checking whether, given a set $\Omega' \subseteq \Omega$ and a state $\omega \in \Omega'$, $\omega \in \mathbf{step}(\Omega')$. A possible way to solve this problem is shown in the function **InStep** of Algorithm 4.2. The procedure simply enumerates all the possible vectors $\vec{\omega}' \in (\Omega')^k$, and checks whether $\mathbf{makeSet}(\vec{\omega}')$ satisfies $\delta_{\mathbf{n}}(\omega, c)$ for some symbol $c \in \Sigma$. The number of vectors in the enumeration is clearly bounded by $|\Omega|^k$. For each vector $\vec{\omega}'$, the algorithm must compute $\mathbf{makeSet}(\vec{\omega})$ (which obviously takes k steps) and check that it satisfies one of Σ different formulas. Each check can clearly be performed in polynomial time with regards to $|\delta_{\mathbf{n}}|$.

It is therefore easy to verify that, for a set $\Omega' \subseteq \Omega$, computing $\mathbf{step}(\Omega')$ only

takes at most $|\Omega|$ invocations of the **InStep** procedure, which is polynomial in $|\Omega|$, $|\Sigma|$ and $|\delta_{\mathbf{n}}|$. Computing $\mathbf{step}^{|\Omega|}(\Omega)$ will therefore involve $\mathcal{O}(|\Omega|^2)$ executions of the **InStep** procedure: the **IsEmpty** procedure will still be polynomial in $|\Omega|$, and its complexity in $|\Sigma|$ and $|\delta_{\mathbf{n}}|$ is unchanged. □

4.5.2 Emptiness of a Zero-Layered ALT

We claim the following result:

Theorem 4.11. *Given a zero-layered alternating looping tree automaton* $A_{\mathbf{a}} = (\Sigma, Q, q_0, k, \delta)$*, it is possible to check the emptiness of* $\mathcal{L}(A_{\mathbf{a}})$ *in a number of steps exponential in* $|Q|$ *and polynomial in* $|\Sigma|$ *and* $|\delta|$*.*

Proof. By successively applying the constructions described in Section 4.2 and Section 4.3, starting from a zero-layered ALT $A_{\mathbf{a}} = (\Sigma, Q, q_0, k, \delta)$ we can obtain a ZLT $A_{\mathbf{z}} = (\Sigma, Q, q_0, k, \delta_0)$ and a NLT $A_{\mathbf{n}} = (\Sigma, 2^Q, \{q_0\}, k, \delta_{\mathbf{n}})$ such that they all recognize the same language (i.e., $\mathcal{L}(A_{\mathbf{a}}) = \mathcal{L}(A_{\mathbf{z}}) = \mathcal{L}(A_{\mathbf{n}})$). The theorem follows by simply applying the complexity results obtained for NLT emptiness checking to $A_{\mathbf{n}}$.

The set of states of the NLT automaton $A_{\mathbf{n}}$ equivalent to $A_{\mathbf{a}}$ is exponentially bigger than the set of states of $A_{\mathbf{a}}$, hence the exponential blowup. The unfortunate explosion is unavoidable: emptiness checking for ALT automata has been shown to be EXPTIME-complete in the number of states. □

Note that the zero elimination construction can potentially determine a huge increase in the size of the transition function. For instance, consider the case of the automaton $A_x = (\{c\}, \{q', q_1, \ldots, q_n\}, q_n, 2, \delta_x)$ with $\delta_x(q', c) = (1, q_1) \wedge (2, q_1)$, $\delta_x(q_1, c) = (q', c) \wedge (q', c)$ and $\delta_x(q_i, c) = (0, q_{i-1}) \wedge$

Algorithm 4.3 Determining whether a set Γ models $\delta_0(q, c)$.

function **ModelsDeltaZero**($\mathcal{A}_{\mathbf{a}}, \Gamma$)
  let $(\Sigma, Q, q_0, k, \delta) := \mathcal{A}_{\mathbf{n}}$
  let $\Gamma' := \Gamma$
  for $i := 1$ to **depth**$(q) - 1$ **do**
    let $q_i := \textbf{depth}^{-1}(i)$
    if $\Gamma' \models \delta(q_i, c)$ **then**
      let $\Gamma' := \Gamma' \cup \{(0, q_i)\}$
    end if
  end for
  return $\Gamma' \models \delta(q, c)$
end function

$(0, q_{i-1})$ for every $i \in [2, n]$. It is easy to verify that $|\delta^0_x| = \mathcal{O}(2^n)|\delta_x|$; since any satisfaction checking algorithm must (at least) scan the formula, this would result an exponential slowdown.

While this explosion does not alter the asymptotic complexity of emptiness checking with regards to $|Q|$ and $|\delta|$ (it only comes up as a multiplicative factor), it is nevertheless possible to avoid it by deferring the calculation of δ_0 as shown in Algorithm 4.3.

By using this simple procedure, the complexity of checking whether a given set models $\delta_0(q, c)$ for some state-symbol pair $(q, c) \in Q \times \Sigma$ is polynomial in both $|\delta|$ and $|Q|$, since the algorithm only has to check if Γ models (at most) $|Q|$ formulas of size (at most) $|\delta|$.

4.5.3 Emptiness of a Zero-Layered 2-ALT

We claim the following result:

Theorem 4.12. *Given a zero-layered two-way alternating looping tree automaton* $A_2 = (\Sigma, Q, q_0, k, \delta)$*, it is possible to check the emptiness of* $\mathcal{L}(A_2)$ *in a number of steps exponential in* $|Q|$ *and polynomial in* $|\Sigma|$ *and* $|\delta|$*.*

Proof. By applying the procedure described in Section 4.4, it is possible to obtain an automaton $A_1 = (\Sigma', Q', q_0', k, \delta')$ such that $\mathcal{L}(A_1)$ is empty iff $\mathcal{L}(A_2)$ is empty. It is straightforward to verify that the size of Q' is linear in the size of Q; $|\delta'|$ and $|\delta|$ are equal up to a constant factor; $|\Sigma'|$ is linear in $|\Sigma|$, and exponential in $|Q|$.

By combining these observations with the previous results, we can infer that this construction does not affect the asymptotic complexity of the algorithm: the alphabet size only figures as a multiplicative coefficient, and therefore the $2^{|Q|}$ expression does not determine a further exponential blowup in the overall complexity measure with respect to $|Q|$. □

Chapter 5

A Practical Decision Procedure

The algorithms described in Chapter 4 have a major drawback: they all require the NLT to be fully constructed beforehand. This would lead to quick exhaustion of the available memory: in order to decide any property of a small TBox containing 30 axioms, we would need to construct, store and inspect a NLT which potentially has 2^{60} states — *a billion billions!*

As we have seen, the exponential blowup is unavoidable: deciding whether an $\mathcal{ALC}$ TBox is satisfiable or whether an $\mathcal{ALC}$ concept is consistent in a TBox is EXPTIME-hard. What we can do, though, is devise a way to reduce the portion of the state space we *actually* inspect in order to obtain an answer. The algorithms we can derive will still have their "weak spots" (corner cases where exponential behaviour is observable), but experience has shown that it is indeed possible to formulate practical decision procedures for DL problems.

5.1 The Decision Procedure

We will start our search for a practical decision procedure with an observation: we only need to find a set of states that make it possible to build a run which starts from the initial state. It is possible — probable, indeed — that the wide majority of the elements in 2^{Q} (which we will call *multistates* from now on) will not be touched by the "core problem".

We will therefore formulate an iterative procedure, that operates on subsets of $|2^{Q}|$ containing so-called *active* and *dead* multistates. The main idea can be described as follows:

- The set **Dead** contains multistates that are provably not in $\mathbf{step}^{|2^{Q}|}(2^{Q})$;
- At each step, we will move from **Act** to **Dead** those multistates that cannot be satisfied without using multistates which are already in **Dead**;
- Next, we will add enough multistates to **Act** such that every state in (the previous value of) **Act** is satisfied by sets that only contain states in **Act**.

Formally, we will procede as follows.

1. Let $i = 0$, $\mathbf{Act}_0 = \{\{q_0\}\}$, and $\mathbf{Dead}_0 = \emptyset$.
2. Let $\mathbf{Succ}_i$ be a function that associates to each multistate $\omega \in \mathbf{Act}_i$ a value defined as follows:
   - If $i > 0$ and $\mathbf{Succ}_{i-1}(\omega)$ does not contain elements from $\mathbf{Dead}_i$, the vector $\mathbf{Succ}_{i-1}(\omega)$;
   - Otherwise, a vector $\vec{y} \in (2^{Q} \setminus \mathbf{Dead}_i)^k$ such that $\mathbf{makeSet}(\vec{y})$ models $\bigvee_{c \in \Sigma} \delta_{\mathbf{n}}(\omega, c)$;

- If no such vector exists, the special symbol *unsat*.

3. Let $\tilde{\mathrm{D}}_i = \{\, \omega \in \mathbf{Act}_i \;:\; \mathbf{Succ}_i(\omega) = \mathit{unsat} \,\}$.
4. Let $\tilde{\mathrm{A}}_i = \left\{\, \omega' \;:\; \omega' \text{ is a component of } \mathbf{Succ}_i(\omega),\ \omega \in \mathbf{Act}_i \setminus \tilde{\mathrm{D}}_i \,\right\}$.
5. Let $\mathbf{Dead}_{i+1} = \mathbf{Dead}_i \cup \tilde{\mathrm{D}}_i$.
6. Let $\mathbf{Act}_{i+1} = \left(\mathbf{Act}_i \setminus \tilde{\mathrm{D}}_i\right) \cup \tilde{\mathrm{A}}_i$.
7. If $\mathbf{Act}_{i+1} \neq \mathbf{Act}_i$, increase i and go to step 2.
8. Return **true** if $\{\, q_0 \,\} \in \mathbf{Act}_i$, **false** otherwise.

First of all, we need to show that the procedure terminates in a finite number of steps. We claim the following result:

Lemma 5.1. *The procedure terminates after at most* $2^{|Q|+1}$ *iterations.*

Proof. At every step i, one of the following conditions must hold:

- $\mathbf{Act}_{i+1}$ contains a multistate that did not appear in $\mathbf{Act}_i \cup \mathbf{Dead}_i$;
- A state from $\mathbf{Act}_i$ is moved to $\mathbf{Dead}_{i+1}$;
- $\mathbf{Act}_{i+1} = \mathbf{Act}_i$, and therefore the procedure terminates.

There are $|2^Q|$ multistates: in the worst case, they get added to the active set one at a time, and successively moved to the dead set one at a time. It follows that the procedure must terminate after at most $2 \cdot |2^Q|$ steps. □

The size of Q is linear in the size of $\mathcal{T}$ and $\hat{\mathrm{C}}$; moreover, each step in the procedure only has to deal with objects whose size is bounded by $(2^{|Q|})^k$. Therefore,

the procedure matches the EXPTIME complexity bound for deciding concept satisfiability in an arbitrary $\mathcal{ALC}^{f}$ TBox[1].

Remember the definition of **step**:

$$\textbf{step}(\Omega') = \left\{ \omega \in \Omega' \ : \ \exists\, \vec{\omega}' \in (\Omega')^{k}, c \in \Sigma.\ \textbf{makeSet}(\vec{\omega}') \models \delta_{\mathbf{n}}(\omega, c) \right\}$$

Let $\Omega_{\mathbf{fix}} = \textbf{step}^{|2^{Q}|}(2^{Q})$; by Theorem 4.1, we know that the language recognized by $A_{\mathbf{n}}$ is non-empty iff $\{ q_0 \}$, the initial state of $A_{\mathbf{n}}$, is in $\Omega_{\mathbf{fix}}$.

In the next two lemmas we will show how the sets created by the decision procedure are connected to the **step** function and to $\Omega_{\mathbf{fix}}$.

Lemma 5.2. *At every step* i *of the decision procedure,* $\textbf{Act}_i \cap \textbf{Act}_{i+1} \subseteq \textbf{step}(\textbf{Act}_{i+1})$.

Proof. The following equality holds by definition:

$$\textbf{step}(\textbf{Act}_{i+1}) = \left\{ \omega \in \textbf{Act}_{i+1} \ : \ \exists\, \vec{\omega}' \in (\textbf{Act}_{i+1})^{k}, c \in \Sigma.\ \textbf{makeSet}(\vec{\omega}') \models \delta_{\mathbf{n}}(\omega, c) \right\}$$

For every $\omega \in \textbf{Act}_i$, there exists a k-vector $\vec{\omega}'$ containing elements from $\textbf{Act}_{i+1}$ iff $\textbf{Succ}_i(\omega) \neq \textit{unsat}$. The set $\textbf{Act}_i \cap \textbf{Act}_{i+1}$ includes, by definition, all the multistates in $\textbf{Act}_i$ for which $\textbf{Succ}_i(\omega) \neq \textit{unsat}$. □

Lemma 5.3. *At every step* i *of the decision procedure,* $\textbf{Dead}_i \subseteq 2^{Q} \setminus \Omega_{\mathbf{fix}}$.

Proof. We will prove this lemma by induction. Since $\textbf{Dead}_0 = \emptyset$, the base case trivially holds. For the inductive step we need to prove that, if $\textbf{Dead}_i \subseteq 2^{Q} \setminus \Omega_{\mathbf{fix}}$, then also $\textbf{Dead}_{i+1} \subseteq 2^{Q} \setminus \Omega_{\mathbf{fix}}$.

[1] The EXPTIME bound holds for $\mathcal{ALC}$ (a sublanguage of $\mathcal{ALC}^{f}$) and for $\mathcal{ALCF}$ (of which $\mathcal{ALC}^{f}$ is a sublanguage).

By definition, $\mathbf{Dead}_{i+1} = \mathbf{Dead}_i \cup \tilde{D}_i$. By inductive hypothesis we know that $\mathbf{Dead}_i \subseteq 2^Q \setminus \Omega_{\mathbf{fix}}$: we only need to show that $\tilde{D}_i \subseteq 2^Q \setminus \Omega_{\mathbf{fix}}$.

The set $\tilde{D}_i$ is defined as follows:

$$\tilde{D}_i = \{\, \omega \in \mathbf{Act}_i \;:\; \mathbf{Succ}_i(\omega) = \mathit{unsat} \,\}$$

Fix a multistate $\omega \in D_i$. We know that $\mathbf{Succ}_i(\omega) = \mathit{unsat}$ iff, for every symbol $c \in \Sigma$ and every vector $\vec{\omega}'$, if $\mathbf{makeSet}(\vec{\omega}') \models \delta_{\mathbf{n}}(\omega, c)$ then $\vec{\omega}'$ contains elements from $\mathbf{Dead}_i$.

By definition, $\mathbf{step}(\Omega_{\mathbf{fix}}) = \Omega_{\mathbf{fix}}$. Since every element from $\mathbf{Dead}_i$ is *not* in $\Omega_{\mathbf{fix}}$, it follows that $\omega \notin \mathbf{step}(\Omega_{\mathbf{fix}})$.

Therefore, every element in $\tilde{D}_i$ is in $2^Q \setminus \Omega_{\mathbf{fix}}$. □

These two results lead us to the following final theorem:

Theorem 5.4. *The language recognized by* $A_{\mathbf{n}}$ *is non-empty iff the procedure above returns* ***true****.*

Proof. By Lemma 5.1, the procedure terminates after a finite number of steps. Suppose the procedure terminates after $\hat{\imath}$ steps. By Lemma 5.2, the following condition holds:

$$\mathbf{Act}_{\hat{\imath}} \cap \mathbf{Act}_{\hat{\imath}+1} \subseteq \mathbf{step}(\mathbf{Act}_{\hat{\imath}+1})$$

The procedure terminates after $\hat{\imath}$ steps iff $\mathbf{Act}_{\hat{\imath}} = \mathbf{Act}_{\hat{\imath}+1}$. The condition can therefore be rewritten as follows:

$$\mathbf{Act}_{\hat{\imath}} \subseteq \mathbf{step}(\mathbf{Act}_{\hat{\imath}})$$

By definition, $\mathbf{step}(\mathbf{Act}_{\hat{\imath}}) \subseteq \mathbf{Act}_{\hat{\imath}}$; as a consequence,

$$\mathbf{Act}_{\hat{\imath}} = \mathbf{step}(\mathbf{Act}_{\hat{\imath}}) = \mathbf{step}^{|2^{Q}|}(\mathbf{Act}_{\hat{\imath}})$$

Since **step** is a monotone increasing function and $\mathbf{Act}_{\hat{\imath}} \subseteq 2^{Q}$, it follows that $\mathbf{step}^{|2^{Q}|}(\mathbf{Act}_{\hat{\imath}}) \subseteq \Omega_{\mathbf{fix}}$ and, therefore, $\mathbf{Act}_{\hat{\imath}} \subseteq \Omega_{\mathbf{fix}}$.

As we know from the previous section, the language recognized by $A_{\mathbf{n}}$ is non-empty iff the initial state $\{ q_0 \}$ is in $\Omega_{\mathbf{fix}}$. Since $\{ q_0 \} \in \mathbf{Act}_0$ (by definition), we know that at the $\hat{\imath}$-th step one of the two following conditions holds:

- $\{ q_0 \} \in \mathbf{Act}_{\hat{\imath}}$ We have shown that $\mathbf{Act}_{\hat{\imath}} \subseteq \Omega_{\mathbf{fix}}$, and therefore $\{ q_0 \} \in \Omega_{\mathbf{fix}}$. It immediately follows that the language accepted by $A_{\mathbf{n}}$ is non-empty, and the procedure correctly returns **true**.
- $\{ q_0 \} \notin \mathbf{Act}_{\hat{\imath}}$ By construction, $\{ q_0 \} \in \mathbf{Act}_0$. It is easy to see that, once a multistate is in $\mathbf{Act}_0$, it remains in $\mathbf{Act}_{\imath} \cup \mathbf{Dead}_{\imath}$ for every $\imath > 0$: as a consequence, $\{ q_0 \} \in \mathbf{Dead}_{\hat{\imath}}$. By Lemma 5.3, $\mathbf{Dead}_{\hat{\imath}} \subseteq 2^{Q} \setminus \Omega_{\mathbf{fix}}$. It immediately follows that $\{ q_0 \} \notin \Omega_{\mathbf{fix}}$, and therefore the language accepted by $A_{\mathbf{n}}$ is empty. Again, the procedure is correct in returning **false**.

As a result, the language recognized by $A_{\mathbf{n}}$ is non-empty iff the procedure above returns **true**. □

Note that, once a multistate is in $\mathbf{Dead}_{\imath}$ for some $\imath$, it cannot appear in $\mathbf{Act}_{\jmath}$ for any $\jmath > \imath$. Suppose that, after the $\tilde{\imath}$-th iteration, the multistate $\{ q_0 \}$ is in **Dead**: the procedure does not need to continue until a fixpoint is reached, as we are sure that $\{ q_0 \} \notin \mathbf{Act}_{\imath}$ for every $\imath > \tilde{\imath}$.

We can therefore improve the algorithm to introduce an early termination condition: the procedure can immediately return **false** as soon as $\{ q_0 \}$ is found to be in $\mathbf{Dead}_i$.

5.2 Another View on the Transition Function

At each iteration of the decision procedure, computing $\mathbf{Succ}_i$ requires a potentially huge number of operations on formulas of the form $\bigvee_{c \in \Sigma} \delta_{\mathbf{n}}(\omega, c)$. In this section we will introduce a different formulation of the transition function which will make it easier to compute $\mathbf{Succ}_i$ by delegating most of the work to a SAT solver.

Fix an $\mathcal{ALC}$ TBox $\mathcal{T}$, an $\mathcal{ALC}$ concept $\hat{C}$ defined over a signature $(\mathcal{C}, \mathcal{R})$, and a set of functional roles $\mathcal{R}_f \subseteq \mathcal{R}$. Let $C_{\mathcal{T}}$ be the internalized version of $\mathcal{T}$, as defined in Section 2.3.3:

$$C_{\mathcal{T}} = \mathbf{nnf}\left(\bigsqcap_{C \sqsubseteq D \in \mathcal{T}} \neg C \sqcup D\right)$$

Consider the automaton $\mathbf{Aut}_f(\mathcal{T}, \mathcal{R}_f, \hat{C}) = (\Sigma, Q, q_0, k, \delta)$, as defined in Section 3.3. We start by noticing that, for "non-atomic" concepts (existentials, universals, unions and disjunctions), the value of the transition does not depend upon the current symbol. For atomic and negated atomic concepts, the value of the transition depends upon the presence (resp. absence) of an atomic concept in the current symbol, rather than on the "whole" symbol.

Let $\mathbf{SPBF}([1 : k] \times Q, \mathcal{C})$ be the language of the "semi-positive" propositional formulas built over the symbols from $\Sigma \cup [1 : k] \times Q$, where only atoms from $\mathcal{C}$ are allowed to appear under the scope of a negation operator. We will use

these formulas to model the transition function in a more succinct form, by removing the redundancies described above.

Let $\Delta : Q \to \mathbf{SPBF}([1:k] \times Q,\ \mathcal{C})$ be a function defined as follows:

$$\begin{aligned}
\Delta(q_0) &= \Delta(q_{\mathbf{tbox}}) \wedge \Delta(\hat{C}) \\
\Delta(q_{\mathbf{tbox}}) &= \Delta(C_{\mathcal{T}}) \wedge \bigwedge_{i \in [1:k]} (i, q_{\mathbf{tbox}}) \vee (i, q_{\mathbf{ne}}) \\
\Delta(q_{\mathbf{ne}}) &= \mathbf{false} \\
\Delta(\top) &= \mathbf{true} \\
\Delta(\bot) &= \mathbf{false} \\
\Delta(A) &= A \\
\Delta(\neg A) &= \neg A \\
\Delta(C \sqcap C') &= \Delta(C) \wedge \Delta(C') \\
\Delta(C \sqcup C') &= \Delta(C) \vee \Delta(C') \\
\Delta(\exists P.C) &= (\mathbf{idx}_f(\exists P.C), C) \\
\Delta(\forall P.C) &= \bigwedge_{i \in \mathbf{all}_f(P)} (i, q_{\mathbf{ne}}) \vee (i, C)
\end{aligned}$$

Let $\hat{\Delta} : 2^Q \to \mathbf{SPBF}([1:k] \times Q,\ \mathcal{C})$ be a function defined as follows:

$$\begin{aligned}
\hat{\Delta}(\{ q_{\mathbf{ne}} \}) &= \mathbf{true} \\
\hat{\Delta}(\omega) &= \bigwedge_{q \in \omega} \Delta(q)
\end{aligned}$$

where $\omega \neq \{ q_{\mathbf{ne}} \}$.

Let $\mathcal{A}_{\mathbf{n}} = (\Sigma, 2^Q, \{ q_0 \}, k, \delta_{\mathbf{n}})$ be the NLT derived from $\mathbf{Aut}_f(\mathcal{T}, \mathcal{R}_f, \hat{C})$ by applying the zero-elimination procedure of Section 4.2 and the subset construc-

tion of Section 4.3.

For every symbol $c \in 2^{\mathcal{C}}$, let $\xi(c)$ be the substitution that maps every symbol $A \in c$ to **true** and every other symbol in $2^{\mathcal{C}} \setminus c$ to **false**. We claim the following result.

Theorem 5.5. *Let δ_0 be the zero-closure of δ as defined in Section 4.2. For any state $q \in Q$ and any symbol $c \in 2^{\mathcal{C}}$, the following equality holds:*

$$\delta_0(q, c) = \Delta(q)\xi(c)$$

Proof. We will show that the equivalence holds for all the states from $\mathbf{sub}(\hat{C}) \cup \mathbf{sub}(C_{\mathcal{T}})$ by structural induction. We distinguish these base cases:

- $C = A$, where A is an atomic concept. The formula $\delta_0(A, c)$ is **true** iff $A \in c$, and is **false** otherwise. By definition, $\Delta(C) = A$. If $A \in c$ then, by definition, $\Delta(A)\xi(c) =$ **true**; if $A \notin c$ then $\Delta(A)\xi(c) =$ **false**.
- $C = \neg A$, where A is an atomic concept. The formula $\delta_0(A, c)$ is **true** iff $A \notin c$, and is **false** otherwise. By definition, $\Delta(C) = \neg A$. If $A \in c$ then, by definition, $\Delta(A)\xi(c) =$ **false**; if $A \notin c$ then $\Delta(A)\xi(c) =$ **true**.
- $C = \top$, $C = \bot$. The result trivially holds.

The inductive step can be shown to hold as follows:

- $C = D \sqcap D'$. By definition, $\delta_0(D \sqcap D', c)$ is equal to $\delta_0(D, c) \wedge \delta_0(D', c)$. By inductive hypothesis, $\delta_0(D, c) = \Delta(D)\xi(c)$ and $\delta_0(D', c) = \Delta(D')\xi(c)$. The following chain of equalities holds:

$$\delta_0(D \sqcap D', c) = \Delta(D)\xi(c) \wedge \Delta(D')\xi(c)$$

$$= (\Delta(D) \wedge \Delta(D'))\xi(c)$$
$$= \Delta(D \sqcap D')\xi(c)$$

- $C = D \sqcup D'$. By definition, $\delta_0(D \sqcup D', c)$ is equal to $\delta_0(D, c) \vee \delta_0(D', c)$. By inductive hypothesis, $\delta_0(D, c) = \Delta(D)\xi(c)$ and $\delta_0(D', c) = \Delta(D')\xi(c)$. The following chain of equalities holds:

$$\begin{aligned}\delta_0(D \sqcup D', c) &= \Delta(D)\xi(c) \vee \Delta(D')\xi(c) \\ &= (\Delta(D) \vee \Delta(D'))\xi(c) \\ &= \Delta(D \sqcup D')\xi(c)\end{aligned}$$

- $C = \exists R.D$, $C = \forall R.D$. In these cases the value of $\delta_0(C, c)$ does not depend on c and, by definition, is always equal to $\Delta(C)$.

The equivalence clearly holds also for q_0, $q_{\mathbf{tbox}}$ and $q_{\mathbf{ne}}$. □

By Lemma 4.6 we know that, for any set of states $\omega \in 2^Q$ and any symbol $c \in \Sigma$, there exists a correspondence between the models of $\bigwedge_{q \in \omega} \delta_0(q, c)$ and the models of $\delta_{\mathbf{n}}(\omega, c)$. In particular, a set Γ models $\bigwedge_{q \in \omega} \delta_0(q, c)$ iff the set $\mathbf{group}(\Gamma)$ models $\delta_{\mathbf{n}}(\omega, c)$, where $\mathbf{group}$ is a bijection from $2^{[1:k] \times Q}$ to $2^{[1:k] \times 2^Q}$ defined as follows:

$$\mathbf{group}(\Gamma) = \{\, (i, \mathbf{collect}(\Gamma, i)) \;:\; i \in [1:k] \,\}$$
$$\mathbf{collect}(\Gamma, i) = \{\, q \in Q \;:\; (i, q) \in \Gamma \,\}$$

This allows us to claim the following immediate result:

Lemma 5.6. *For any set of states* $\omega \in 2^Q$ *and any symbol* $c \in 2^{\mathcal{C}}$*, a set* $\Gamma' \subseteq [1:k] \times 2^Q$ *models* $\delta_{\mathbf{n}}(\omega, c)$ *iff the set* $\Gamma = \mathbf{group}^{-1}(\Gamma')$ *models the formula* $\Delta(\omega)\xi(c)$.

This lemma, in turn, allows us to claim a further property of Δ whose significance will be clear soon.

Theorem 5.7. *For any set of states* $\omega \in 2^Q$*, a set* $\Gamma' \subseteq [1:k] \times 2^Q$ *models the formula* $\bigvee_{c \in \Sigma} \delta_{\mathbf{n}}(\omega, c)$ *iff there exists a symbol* c' *such that the set*

$$\Gamma_{c'} = \{ c' \} \cup \mathbf{group}^{-1}(\Gamma')$$

models the formula $\Delta(\omega)$.

Proof. First of all, we will prove the statement for $\omega = \{ q_{\mathbf{ne}} \}$. In this case, the formula $\delta_{\mathbf{n}}(\{ q_{\mathbf{ne}} \}, c)$ is satisfied by $c = \sigma_{\mathbf{ne}}$. Accordingly, $\hat{\Delta}(\{ q_{\mathbf{ne}} \}) = \mathbf{true}$.

From now on we shall only consider multistates different from $\{ q_{\mathbf{ne}} \}$. This means that we are only interested in symbols in $2^{\mathcal{C}}$: by definition, the symbol $\sigma_{\mathbf{ne}}$ can not satisfy any multistate different from $\{ q_{\mathbf{ne}} \}$.

Γ' models $\bigvee_{c \in \Sigma} \delta_{\mathbf{n}}(\omega, c)$ iff there exists a symbol c' such that $\Gamma' \models \delta_{\mathbf{n}}(\omega, c')$. As we have seen, this symbol c' must be a member of $2^{\mathcal{C}}$. By the previous lemma, $\Gamma' \models \delta_{\mathbf{n}}(\omega, c')$ iff $\mathbf{group}^{-1}(\Gamma') \models \hat{\Delta}(\omega)\xi(c')$.

The substitution $\xi(c')$ replaces the atom c' with **true**, and all the other atoms from Σ with **false**. It is clearly the case that a set X models a formula of the form $\phi\xi(c')$ iff the set $X \cup \{ c' \}$ models ϕ. It follows that $\Gamma' \models \delta_{\mathbf{n}}(\omega, c')$ iff $\mathbf{group}^{-1}(\Gamma') \cup \{ c' \} \models \hat{\Delta}(\omega)$. □

In order to calculate $\mathbf{Succ}_i(\omega)$, the reasoner can therefore procede as follows:

1. Determine a model Y for $\hat{\Delta}(\omega)$;
2. Let $\Omega_Y = \{\, \omega \;:\; \exists\, i \in [1:k].\ (i, w) \in Y \,\}$;
3. If $\Omega_Y \cap \mathbf{Dead}_i = \emptyset$, then $\mathbf{Succ}_i(\omega) = Y$;
4. If $\hat{\Delta}(\omega)$ does not have any other model, then $\mathbf{Succ}_i(\omega) = \mathit{unsat}$;
5. Otherwise, return to step 1.

The first step can be delegated to a SAT solver — a software tool that, given a propositional formula, produces a satisfying assignment. While being worst-case exponential (they have to solve a NP-complete problem on a deterministic machine), SAT solvers are able to efficiently handle formulas containing thousands of variables.

Our formulas include a number of variables that is roughly linear in the size of the inputs (TBox and concept), and is often much less than that. This means that, by using SAT solvers to handle this part of the problem, we can take advantage of decades of research and optimization and (hopefully) obtain a good level of performance.

5.3 Further Optimizations

In this section we will describe some techniques that try to take advantage of patterns arising from the general structure of automata corresponding to DL problems.

First of all, we need to make sure that, after having checked a model Y, the system does not have to check a model Y' such that $Y \setminus \Sigma \subseteq Y' \setminus \Sigma$. We can achieve this by adding to $\hat{\Delta}(\omega)$ a conjunct of the form $\bigvee_{y \in Y \setminus \Sigma} \neg y$.

The number of propositional letters appearing in $\hat{\Delta}(\omega)$ depends linearly on the number of concepts in the signature. Therefore, there are potentially $\mathcal{O}(2^{|\mathcal{C}|})$ different sets Y' such that $Y \setminus \Sigma = Y' \setminus \Sigma$. While potentially taxing on the SAT solver, this optimization removes a huge bottleneck.

Moreover, we can attach a different annotation to atoms of the form (i, C), in order to distinguish whether they are originated from an existential or from a universal concept. We will rewrite the definition of Δ as follows:

$$\Delta(\exists R.C) = (\mathbf{idx}_f(\exists R.C), C)_\exists$$
$$\Delta(\forall R.C) = \bigwedge_{i \in \mathbf{all}_f(R)} (i, C)_\forall$$

Whenever a model of $\hat{\Delta}(\omega)$ contains an atom of the form $(i, C)_\forall$ and does *not* contain any atom of the form $(i, C')_\exists$, we will simply drop the first atom from the set. Intuitively, we force the SAT solver to pick the "cheaper" option and assume that there is no i-th node, unless an existential concept is forcing its existence. There is no need of adding $(i, q_{\mathbf{ne}})$ to the model: the multistate $\{ q_{\mathbf{ne}} \}$ is always known to be satisfiable (by the symbol $\sigma_{\mathbf{ne}}$).

Most SAT solvers have algorithms that operate on formulas in CNF. In particular, the input is a set of clauses (i.e., a set of sets of atoms) of the form $\{\gamma_1, \ldots, \gamma_n\}$. When given this input, the SAT solver will output a model for the formula $\bigvee_{i \in [1:n]} \bigwedge_{a \in \gamma_i} a$.

In general, the CNF expansion of a given formula can be exponentially bigger. The alternate construction of $\Delta(\forall R.C)$ removes one source of the exponential blowup: as a result, only disjunctive subconcepts result in a problematic situation. Clearly, the size of disjunctive subconcepts resulting from the internalization of a TBox is limited by the size of the inclusion assertions, which is

usually much smaller than the number of assertions in the TBox.

Moreover, the structure of $\hat{\Delta}$ makes it particularly easy to obtain a CNF expansion of $\hat{\Delta}(\omega)$. For each $q \in \omega$, let $\mathcal{D}(q)$ be a set of clauses (i.e., a set of sets of atoms) such that the following equivalence holds:

$$\Delta(q_i) \equiv \bigwedge_{\gamma \in \mathcal{D}(q_i)} \left(\bigvee_{a \in \gamma} a \right)$$

It is easy to check that the following equivalence holds:

$$\hat{\Delta}(\omega) = \bigwedge_{q \in \omega} \Delta(q) \equiv \bigwedge_{q \in \omega, \gamma \in \mathcal{D}(q)} \left(\bigvee_{a \in \gamma} a \right)$$

Therefore, we can compute the CNF expansion of $\hat{\Delta}(\omega)$ as the union, for every $q \in \omega$, of the clauses in the CNF expansion of $\Delta(q)$.

Chapter 6

The *TreeHug* Reasoner

In this chapter we will introduce *TreeHug*, a DL reasoner based upon the techniques exposed in the previous chapters. To the best of our knowledge, this is the first DL reasoner that implements an automata-based decision procedure.

We will then proceed by giving a quick overview of the major architectural choices connected with the implementation, and wrap up the chapter with some performance results on the current version of the code.

6.1 Architecture of the System

The *TreeHug* reasoner implements the decision procedure described in this chapter, with some simple optimizations.

The core of the reasoner is written in *Scala*[1], a language that integrates features of object-oriented and functional programming. This choice addresses two key concerns: the Scala programming language is more flexible than Java

[1] http://www.scala.org/

(arguably, it is a strict superset thereof), while preserving the ability to interface seamlessly with Java code. This feature is of primary importance, given the vast prevalence of Java as the programming language of choice in the field of Semantic Web technologies.

Another crucial component of the system is the *MiniSAT*[2] SAT solver [ES03], to which the work of calculating models for $\hat{\Delta}$ is delegated. MiniSAT is based on the DPLL algorithm [DLL62], with many enhancements to the handling of several commonly occurring patterns. MiniSAT was chosen mainly for the fact that it represents a good compromise between solid performances and ease of integration: the solver exposes a clean interface, and the extensive comments in the source code make it possible to circumvent the relative lack of documentation.

The interface between these two components is a custom-built JNI library written in C++, that creates, manages and wraps MiniSAT instances. The resulting decoupling between the reasoner and the SAT solver makes it possible to switch to a different SAT library (should the necessity arise), and enables us to take advantage of low-level operations to speed up some tasks (e.g., feeding a large formula to the solver).

6.2 Benchmarks

The reasoning algorithm implemented in *TreeHug* has an unavoidably high worst case complexity. The hope is, however, that the system performs well in realistic applications. To check the validity of this claim, it would be necessary

[2] http://www.minisat.se/

to test the performance of the system with the widest possible range of "real-world" ontologies, similarly to the approach taken in [GHT06].

Within the limitations of the current prototype of *TreeHug*, though, this approach is not yet feasible. As a consequence, we have opted for a set of synthetic benchmarks aimed at showing the scalability of *TreeHug*. While probably not indicative of any real-world usage pattern, the system shows a promising level of performance even in its current immature state.

Benchmark 1 Let n be an integer and fix a signature containing an atomic role R, and $3(n+1)$ atomic concepts labeled C_i, D_i and E_i for $i \in [1 : n+1]$. Let $\mathcal{T}_1$ be a TBox containing assertions of the following form:

$$C_i \sqsubseteq (D_i \sqcap E_i) \sqcup C_{i+1}$$

$$D_i \sqsubseteq \exists R.C_{i+1}$$

$$E_i \sqsubseteq \forall R.\neg C_{i+1}$$

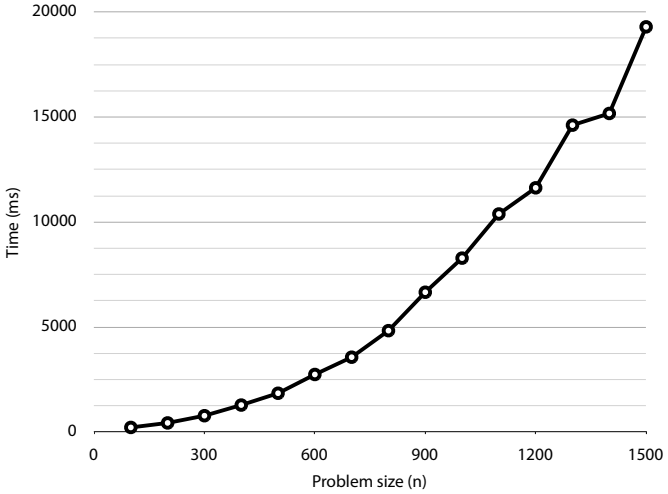
for every $i \in [1 : n]$.

We ask *TreeHug* to check whether the concept $C_1 \sqcap \neg C_{n+1}$ is consistent in $\mathcal{T}_1$, which is false. The results we have obtained are shown in Figure 6.1.

Benchmark 2 Let n be an integer and fix a signature containing two atomic roles, R and S, and $3(n+1)$ atomic concepts labeled C_i, D_i and E_i for $i \in [1 : n+1]$. Let $\mathcal{T}_2$ be a TBox containing assertions of the following form:

$$C_i \sqsubseteq (D_i \sqcap E_i) \sqcup \exists S.C_{i+1}$$

$$D_i \sqsubseteq \exists R.C_{i+1}$$



n	min	avg	max
100	155	217	430
200	326	431	713
300	719	775	827
400	1227	1283	1366
500	1800	1845	1884
600	2662	2739	2825
700	3495	3562	3608
800	4736	4827	4913
900	6338	6657	6854
1000	8176	8282	8335
1100	10225	10384	10547
1200	11480	11630	11823
1300	14396	14616	14855
1400	14907	15173	15360
1500	19059	19298	19587

Figure 6.1: Results of Benchmark 1

$$E_i \sqsubseteq \forall R.\neg C_{i+1}$$

for every $i \in [1 : n]$, plus the assertion

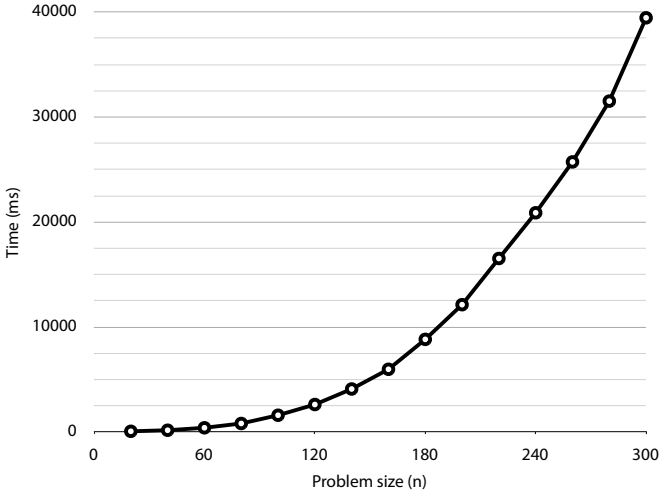
$$C_{n+1} \sqsubseteq \bot$$

We ask *TreeHug* to check whether the concept C_1 is consistent in $\mathcal{T}_1$, which is false. The results we have obtained are shown in Figure 6.3.

Benchmark 3 The third benchmark is derived from the second by omitting the assertion $C_{n+1} \sqsubseteq \bot$. Let n be an integer and fix a signature containing two atomic roles, R and S, and $3(n+1)$ atomic concepts labeled C_i, D_i and E_i for $i \in [1 : n+1]$. Let $\mathcal{T}_3$ be a TBox containing assertions of the following form:

$$C_i \sqsubseteq (D_i \sqcap E_i) \sqcup \exists S.C_{i+1}$$

$$D_i \sqsubseteq \exists R.C_{i+1}$$



n	min	avg	max
20	51	65	93
40	150	166	191
60	368	399	441
80	787	809	839
100	1574	1595	1627
120	2549	2617	2776
140	4000	4095	4253
160	5809	5973	6260
180	8692	8824	9012
200	11964	12128	12379
220	16182	16529	17521
240	20830	20883	20957
260	25170	25728	27580
280	31345	31521	31848
300	39282	39458	39830

Figure 6.2: Results of Benchmark 2

$$E_i \sqsubseteq \forall R.\neg C_{i+1}$$

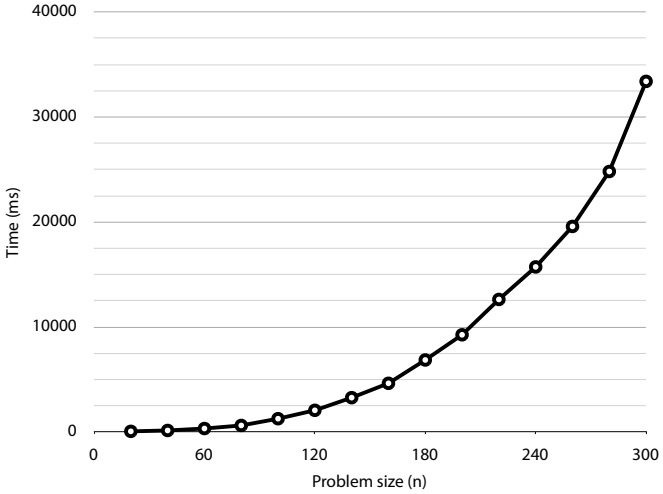
for every $i \in [1 : n]$. We ask *TreeHug* to check whether the concept C_1 is consistent in $\mathcal{T}_3$, which is true. The results we have obtained are shown in Figure 6.3.

6.2.1 Methodology

For each value of n, the results represent the minimum, average and maximum running time of 5 executions of the decision procedure. Two warm-up runs are discarded.

The results can be duplicated by invoking the `benchmarks/run` script contained in the source distribution with the following parameters:

```
benchmarks/run 1 100 100 1000
benchmarks/run 2 20 20 300
benchmarks/run 3 20 20 300
```



n	min	avg	max
20	46	61	81
40	121	144	163
60	284	330	433
80	589	621	728
100	1211	1265	1306
120	1960	2065	2237
140	3193	3270	3470
160	4547	4637	4762
180	6751	6860	6954
200	9050	9259	9551
220	12326	12626	12956
240	15627	15725	15924
260	19179	19583	20107
280	24672	24816	24944
300	30598	33403	37978

Figure 6.3: Results of Benchmark 3

The test system is an Apple MacBook equipped with a 2.4 GHz Intel Core 2 Duo processor[3] and 4 GB of RAM, running Mac OS X 10.6.4, the Java HotSpot 64-bit Server VM (build 16.3-b01-279) and Scala 2.8.0. The version of MiniSAT used for this benchmark is MiniSat 2-070721; we are in the process of migrating the code to the recently released MiniSAT 2.2.0.

[3] CPU speed throttling has been disabled during the tests.

Chapter 7

Related Work

This thesis falls in the intersection of two very active research areas, automata theory and Description Logics. The confluence of these two lines of research has already lead to the discovery of many interesting complexity results for reasoning problems over very expressive DL languages. This is hardly a surprise, given the wealth of results that have arised from the application of automata constructions to a multitude of logic fragments. We will begin this chapter with an overview of these reductions.

In the second section we will describe some of the main decision procedures for automata problems, focusing on the main differences with the algorithms we introduced in the previous chapters. Finally, we will close this chapter with a brief account of tableau-based algorithms, the stream of research which has yielded the most important results in the field of practical DL reasoning.

7.1 Automata Reductions for Problems in Logic

In [Büc60], J. Richard Büchi showed the decidability of the monadic second-order logic of one successor (S1S). Monadic SOL is a fragment of SOL where quantification over relations of arity greater than 1 is not allowed; moreover, the signature of S1S includes a single successor predicate. Intuitively, a S1S formula can be seen as the description of a property of infinite words.

In order to obtain the main result, Büchi showed a reduction from a S1S formula to an automaton over infinite words, whose emptiness was known to be decidable: this proof technique was soon adopted by other logicians to deal with many different logical languages.

Michael O. Rabin, for instance, used a similar reduction in [Rab69] for deciding the monadic second-order logic of two successors (S2S), which includes two successor predicates. An interpretation over a S2S signature can be therefore seen as a binary tree, where the two predicates connect a node to its two child nodes.

The number of results that have followed these initial papers makes it impossible to provide a detailed account of the developments. A survey of the research directions that have been pursued in the field can be found in [Tho90, Tho97]. In a somewhat "gentler" introduction to the topic, [VW07] describes the translations from various logics (S1S, S2S, temporal logics and the modal μ-calculus) to automata decision problems.

The usage of automata-based techniques for Description Logics has its roots in the algorithm for μ-calculus developed by Moshe Vardi in [Var98]. The basic idea has been used to determine complexity bounds for $\mathcal{SHIQ}$ in [Tob01], and

for $\mathcal{ALCIQ}b_{reg}$ in [CDGL02a]. For a thorough review of the field, the reader can refer to [Ort10, Section 3.5].

7.2 Automata Decision Procedures

Many algorithms have been described for deciding the emptiness of specific families of automata. They are characterized by the presence of intermediate steps that operate on "temporary" automata, whose state space is often exponentially bigger than the number of states of the input automaton. This has been one of the major limiting factors in the practicality of the approach: in particular, the "standard technique" [Var98] for deciding alternating tree automata emptiness goes through the complementation of a parity word automaton via the so-called Safra construction [Saf88], which has *"proved to be not too amenable to implementation"*[1]. An alternative, so-called "Safraless" construction has been proposed in [KV05]; an incremental version of the corresponding decision procedure has been explored in [ÜT07a, ÜT07b].

In [PSV06], the authors adopt an approach somewhat similar to ours for the modal logic $\mathcal{K}$. Analogously to what we have done in this thesis, the algorithm by Pan, Sattler and Vardi uses a compact representation of the transition function of the automaton deriving from a modal formula. The resulting decision procedure procedes both top-down (starting from the set of all types and eliminating all the non-feasible types) and bottom-up (starting from the set of types not containing existential subformulas, and adding those containing existential subformulas that can be satisfied). There is little doubt that the approach could be extended to deal with more expressive DL languages. To

[1] In [KV05].

the best of our knowledge, though, there are no further results on the topic since 2006.

One important difference between the two works is the method chosen for representing the transition function. Due to the structure of TBoxes, which can be seen as conjunctions of many "small" inclusion assertions, we chose to work with a CNF representation of the transition function. We have experimented with BDDs (as in the work by Pan et al.), but the scalability of the system was greatly impaired by the sheer number of BDD operations necessary to build the BDD for the "propagation" state $q_{\mathbf{tbox}}$. On the other hand, building the CNF of the transition function is a much more efficient procedure. The exponential blowup is limited in scope to each inclusion assertion; if the size of a single inclusion assertion is bounded, the size of $\hat{\Delta}(q_{\mathbf{tbox}})$ is only linear in the number of assertions.

The principal area of divergence between the two works, though, is the different starting point for the decision algorithm. We adopt a top-down approach, yet avoid analyzing all the multistates of the NLT. We regard the bottom-up approach as unfeasible in our case, due to the great number of satisfiable multistates (exponential in the size of the signature).

Somewhat ironically, [HS03] proposes an algorithm that reduces the emptiness of an alternating looping automata on infinite trees to the satisfiability of a knowledge base expressed in $\mathcal{ELU}_f$, a variant of $\mathcal{ALC}$ which lacks negation and universal quantification and where all the roles are functional. Since $\mathcal{ELU}_f$ is a sublanguage of $\mathcal{ALC}^f$, one could use *TreeHug* to decide the satisfiability of the knowledge base resulting from the translation of a given automaton — by converting it back to a tree automaton, of course!

7.3 Tableau Based Reasoners

Tableau algorithms are by far the most developed technique for reasoning in Description Logics. The natural formulation of the approach, the early availability of efficient implementations and the sheer amount of man-years devoted to optimizing the approach have determined this family of algorithms to be the clear frontrunner.

It would be impossible to give a complete account of the techniques developed in twenty-five years of research, starting from $\mathcal{KRIS}$ [BH91] to the latest reasoners such as *HermiT*[2] [SMH08]: we will refer the reader to [BS01] for an introduction to the general principles of tableau algorithms.

The core idea is to show the satisfiability of a given DL concept by building an interpretation that satisfies it. Variants of the procedure have been developed for Description Logics that do not always admit a finite model: in these cases, it suffices to build a "pre-model" that can be expanded at will to form an (infinite) model for the given concept.

The vast amount of theoretical results in this field is reflected by the real-world performance of reasoners that implement these procedures. For instance, three of the most widely deployed DL reasoners, *FaCT++*[3] [TH06], *Pellet*[4] [SP06] and *RACER*[5] [HM01], are based upon tableau algorithms.

This approach is the inverse of the one we try to follow: instead of building a model, we try to explicitly enumerate the concepts for which a model can *not* be built. It remains to be seen whether this way of "flipping" the approach

[2] http://hermit-reasoner.com/
[3] http://code.google.com/p/factplusplus/
[4] http://www.clarkparsia.com/pellet/
[5] http://www.racer-systems.com/products/racerpro/

can yield an implementation whose results that compete with the established players.

The early results shown in Section 6.2 are probably better than we expected at the beginning of this journey. The vast amount of expertise accumulated in the field of tableau-based reasoning, though, makes it quite hard to foresee a scenario where automata-based reasoners manage to close the performance gap.

Chapter 8

Conclusions

In this thesis we have described a new automata-based procedure for reasoning over expressive Description Logics. The resulting algorithm can decide the consistence of a concept in a TBox in exponential time, matching the known theoretical bound. More remarkably, unlike previous automata based approaches, our algorithm seems well suited for implementation. Indeed, we have developed a prototype reasoner, and its experimental evaluation revealed promising results.

More specifically, we started by showing how to build an automaton that recognizes a family of the models of the given TBox which contain an object that participates in the given concept. By virtue of the tree model property, which we had previously introduced in Section 2.3.2, a concept is consistent in a TBox iff the language recognized by the automaton is not empty.

Successively, we introduced a sequence of reductions that make it possible to decide the emptiness of a two-way alternating automaton. The procedure starts by removing the "upwards" transitions and the zero transitions. Then it

applies a subset construction to transform the resulting alternating automaton into a nondeterministic automaton, and solves the emptiness problem for the NLT.

The key to obtaining a practical decision procedure was to show a different formulation of the algorithm. By focusing on a small but expressive DL fragment, we have been able to obtain a new algorithm that makes it possible to directly solve the emptiness problem of an automaton equivalent to a given DL problem, without going through any intermediate steps.

The *TreeHug* reasoner, which implements this algorithm, showed promising results in our preliminary benchmarks. Of course it still lags behind other established and highly optimized DL reasoners, and it is not clear whether it will be able to catch up. Still, it gives a first positive answer to the practical applicability of automata-based procedures to DL reasoning problems.

8.1 Future Work

Both the theoretical framework presented in this work and the *TreeHug* reasoner are still quite far from being considered "ready". The future work can therefore procede in parallel on both aspects of this thesis.

The theoretical results presented in this work can be extended to number restrictions (as in [CDGL02a]), and integrated with some well-known preprocessing steps for removing role hierarchies and role transitivity assertions (see [Mot06, Section 5.2]). This would make it possible to handle a very expressive DL language such as $\mathcal{SHIQ}$.

Another very important extension would be the support for ABox reasoning. There are several known automata-based algorithms for reasoning over ABoxes or over DL languages which allow individuals to appear in concept expressions (see, for instance, [Ort10]). It is not clear, though, whether the techniques we propose can be extended to solve these problems with a reasonable level of efficiency.

On the implementation side, the performance level obtained with this first prototype of *TreeHug* is encouraging. Adding more constructs to the language, though, can grow the state space by a significant amount. While the asymptotic complexity of the decision procedure is not affected, we need to make sure that using *TreeHug* to reason on languages more expressive than $\mathcal{ALC}^{\mathrm{f}}$ is practically feasible.

In particular, one of the main problems of the current approach lies in the fact that the SAT solver does not have any information about the overall problem. A tighter integration of MiniSAT with the reasoner would allow us to formulate a heuristic for the satisfiability problem that takes into account the "big picture", although the effort required to formulate and implement such an algorithm may prove to be significant.

Bibliography

[Baa91] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, 1991.

[BCM+07] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, 2007.

[BH91] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge (PDK'91)*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 67–86. Springer, 1991.

[BS01] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, 2001.

[Büc60] J. Richard Büchi. On a decision method in restricted second order arithmetic. In E. Nagel et al., editors, *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1960.

[CDGL99] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 84–89, 1999.

[CDGL02a] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 2ATAs make DLs easy. In *Proc. of the 15th Int. Workshop on Description Logic (DL 2002)*, volume 53 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/, pages 107–118, 2002.

[CDGL02b] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski*, volume 2408 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2002.

[CDGL$^+$05] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. *DL-Lite*: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing (SAT 2003)*, pages 502–518, 2003.

[GHT06] Tom Gardiner, Ian Horrocks, and Dmitry Tsarkov. Automated benchmarking of description logic reasoners. In *Proc. of the 19th Int. Workshop on Description Logic (DL 2006)*, 2006.

[HM01] Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[HS03] Jan Hladik and Ulrike Sattler. A translation of looping alternating automata into description logics. In *Proc. of the 19th Int. Conf. on Automated Deduction (CADE 2003)*, pages 90–105, 2003.

[Kri67] Saul Kripke. Semantic analysis of modal logic I, normal propositional calculi. *Zeitschrift für mathemathische Logik und Grundlagen der Mathematik*, 9:67–96, 1967.

[KV05] Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *Proc. of the 46th Annual Symp. on the Foundations of Computer Science (FOCS 2005)*, pages 531–542, 2005.

[Mot06] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, University of Karlsruhe (TH), 2006.

[Ort10] Magdalena Ortiz. *Query Answering in Expressive Description Logics: Techniques and Complexity Results*. PhD thesis, Vienna University of Technology, 2010.

[PSV06] Guoqiang Pan, Ulrike Sattler, and Moshe Y. Vardi. BDD-based decision procedures for the modal logic ||. *J. of Applied Non-Classical Logics*, 16(1-2):169–208, 2006.

[Rab69] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. of the Amer. Mathematical Society*, 141:1–35, 1969.

[Saf88] Shmuel Safra. On the complexity of ω-automata. In *Proc. of the 29th Annual Symp. on the Foundations of Computer Science (FOCS'88)*, pages 319–327, 1988.

[SMH08] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008)*, 2008.

[SP06] Evren Sirin and Bijan Parsia. Pellet system description. In *Proc. of the 19th Int. Workshop on Description Logic (DL 2006)*, volume 189 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/, 2006.

[Str82] Robert S. Streett. Propositional Dynamic Logic of looping and converse is elementarily decidable. *Information and Control*, 54:121–141, 1982.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, pages 292–297, 2006.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*,

volume B, chapter 4, pages 133–192. Elsevier Science Publishers, 1990.

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Language Theory*, volume III, pages 389–455. 1997.

[Tob01] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[ÜT07a] Gulay Ünel and David Toman. An incremental technique for automata-based decision procedures. In *Proc. of the 21st Int. Conf. on Automated Deduction (CADE 2007)*, pages 100–115, 2007.

[ÜT07b] Gulay Ünel and David Toman. Logic programming approach to automata-based decision procedures. In *Proc. of the 23th Int. Conf. on Logic Programming (ICLP 2007)*, pages 165–179, 2007.

[Var98] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of the 25th Int. Coll. on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

[VW07] Moshe Vardi and Thomas Wilke. Automata: From logics to algorithms. In *Proc. of the Automata and Logic Workshop (WAL 2007)*, pages 645–753, December 2007.

[W3C04] W3C OWL Working Group. OWL Web Ontology Language reference. W3C Recommendation, World Wide Web Consortium, February 2004. http://www.w3.org/TR/owl-ref/.

[W3C09] W3C OWL Working Group. OWL 2 Web Ontology Language: Overview. W3C Recommendation, World Wide Web Consortium, October 2009. http://www.w3.org/TR/owl2-overview/.