5
Complexity

In Chapter 4, we looked at concrete algorithms for reasoning in $\mathcal{ALC}$ and some of its extensions. In this chapter, we are taking a more abstract viewpoint and discuss the computational complexity of reasoning, which essentially is the question of how efficient we can expect *any* reasoning algorithm for a given problem to be, even on very difficult ("worst-case") inputs. Although we will concentrate on the basic reasoning problems satisfiability and subsumption for the sake of simple exposition, all results established in this chapter also apply to the corresponding KB consistency problem. In fact, there are very few relevant cases in which the computational complexity of satisfiability and of KB consistency diverge. We start with $\mathcal{ALC}$ and show that the complexity of satisfiability and of subsumption depend on the TBox formalism that is used: without TBoxes and with acyclic TBoxes, it is PSPACE-complete while general TBoxes raise the complexity to EXPTIME-complete. Then we consider two extensions of $\mathcal{ALC}$, $\mathcal{ALCOI}$ and $\mathcal{ALCOIQ}$, and show that satisfiability and subsumption are more difficult in these DLs: in $\mathcal{ALCOI}$, satisfiability and subsumption are EXPTIME-complete already without TBoxes. We show only hardness to illustrate the increase in complexity. In $\mathcal{ALCOIQ}$, reasoning even becomes NEXPTIME-complete (without TBoxes). Again, we show only hardness. Finally, we consider two extensions of $\mathcal{ALC}$ that render reasoning undecidable: role value maps and a certain concrete domain based on the natural numbers and incrementation.

Before starting to analyse the computational complexity of DLs, let us recall some basics of complexity theory. A *complexity class* is a set of problems that share some relevant computational property such as being solvable within the same resource bounds. For example, PTIME

is the class of all problems that can be solved by a deterministic Turing machine in time polynomial in the size of the input. In this chapter, we will mainly be concerned with the following standard complexity classes, which we order according to set inclusion:

$$\text{PTime} \subseteq \text{NP} \subseteq \text{PSpace} \subseteq \text{ExpTime} \subseteq \text{NExpTime}.$$

The reader is referred to standard textbooks on complexity theory for the exact definition of these classes [AB09, Sip97, Pap94]. It is commonly believed that the inclusions shown above are all strict, but proofs have not yet been found. However, it is known that $\text{PTime} \subsetneq \text{ExpTime}$ and $\text{NP} \subsetneq \text{NExpTime}$.

For the purposes of this book, a problem is *hard* for a complexity class $\mathcal{C}$ if every problem in $\mathcal{C}$ can be reduced to it in polynomial time.[1] It is *complete* for $\mathcal{C}$ if it is hard for $\mathcal{C}$ and contained in $\mathcal{C}$. Intuitively, a problem that is $\mathcal{C}$-complete belongs to the hardest problems in $\mathcal{C}$. For example, an ExpTime-complete problem is among the hardest problems in ExpTime. In particular, it is not in PSpace unless $\text{PSpace} = \text{ExpTime}$. Since the inclusion $\text{PTime} \subseteq \text{ExpTime}$ is strict, no ExpTime-hard problem can be solved in polynomial time by a deterministic algorithm. When we prove that a problem P is hard for a complexity class $\mathcal{C}$, we will often call this a *lower bound* because it says that P is *at least* as hard as the other problems in $\mathcal{C}$ (but possibly much harder). Likewise, proving that P is contained in $\mathcal{C}$ will be called an *upper bound* because it means that solving P is *at least* as easy as $\mathcal{C}$-hard problems (but possibly much easier).

5.1 Concept satisfiability in $\mathcal{ALC}$

We begin our journey into the complexity of description logics by looking at concept satisfiability in the basic DL $\mathcal{ALC}$. As has been shown in Theorem 2.17, satisfiability and non-subsumption in $\mathcal{ALC}$ and its extensions can be mutually polynomially reduced. Therefore, we can concentrate on the complexity of satisfiability since it immediately yields the complexity of subsumption as well. Note, however, that the mutual polynomial reduction is between satisfiability and *non*-subsumption, and thus completeness of satisfiability for some complexity class $\mathcal{C}$ implies completeness of subsumption for the *complement* of $\mathcal{C}$. This is not an issue for

[1] This is not a useful definition for the class PTime, but we will not consider PTime-hardness anyway.

complexity classes that are closed under complement such as PTime, PSpace and ExpTime, but it is important for NP and NExpTime, which are not known (or believed) to be closed under complement.

When looking at the complexity of concept satisfiability in $\mathcal{ALC}$, we have to be careful about the TBox formalism that we use. As we shall see, using no TBox at all and using acyclic TBoxes results in satisfiability being PSpace-complete; in contrast, using general TBoxes results in ExpTime-completeness. We start with the former.

5.1.1 Acyclic TBoxes and no TBoxes

We start with proving the upper bound, i.e., that concept satisfiability in $\mathcal{ALC}$ with respect to acyclic TBox is in PSpace.

Upper Bound

Throughout Chapter 5, we develop several algorithms with the aim of proving upper complexity bounds. In this context, we are interested in algorithms that can be described as elegantly as possible, and will not pay attention to their practical feasibility. For example, when proving an ExpTime upper bound, we shall not worry about an algorithm that requires exponential time in the best case (i.e., on *every* input), although this is clearly prohibitive for practically useful implementations.

We know from Theorem 3.24 that $\mathcal{ALC}$ has the tree model property; that is, if a concept C is satisfiable with respect to a TBox $\mathcal{T}$, then C has a tree model with respect to $\mathcal{T}$. We can even strenghten this statement by requiring that the outdegree of the tree model is bounded by the size of C and $\mathcal{T}$ because, intuitively, every element needs at most one successor for each existential restriction that occurs in C and $\mathcal{T}$. When we admit only acyclic TBoxes instead of general ones, we can further strengthen the statement by requiring also that the depth of the tree model is bounded by the size of the input. This suggests the following strategy for deciding satisfiability: when constructing a tree model, traverse it in a depth-first manner until the whole model has been explored; at any given time, keep only the single branch of the tree model in memory on which the algorithm is currently working. With this strategy, the tableau algorithm needs only polynomial space: although the size of the entire tree model is exponential, a single branch can be stored in polynomial space.

Although, in principle, the described strategy can be implemented

with a tableau algorithm similar to those presented in Chapter 4, here we prefer to use an algorithm that can be described in a simpler way. This algorithm, which is very close to the so-called K-worlds algorithm from modal logic, reduces the described strategy to its essence: it nondeterministically "guesses" its way through a tree model in a depth-first manner, exploiting that the deterministic and nondeterministic versions of the complexity class PSPACE coincide by Savitch's theorem.

It is convenient to work with acyclic TBoxes in a particular normal form, which we introduce first. To start with, we assume without loss of generality that (i) the satisfiability of concept *names* with respect to acyclic TBoxes is to be decided and (ii) acyclic TBoxes contain only exact concept definitions, but no primitive ones. For (i), note that a compound concept C is satisfiable with respect to a TBox $\mathcal{T}$ if and only if A is satisfiable with respect to $\mathcal{T} \cup \{A \equiv C\}$, where A is a fresh concept name (that is, it does not appear in C and $\mathcal{T}$). For (ii), we can replace every primitive definition $A \equiv C$ with the exact one $A \equiv A' \sqcap C$, with A' a fresh concept name.

A precursor to the normal form is *negation normal form* (NNF). An acyclic TBox $\mathcal{T}$ is in NNF if negation is applied only to primitive concept names in $\mathcal{T}$, but neither to defined concept names nor to compound concepts. There is a close relation to the negation normal form of concepts defined in Chapter 4: if $\mathcal{T}'$ is the expansion of an acyclic TBox $\mathcal{T}$ in NNF, then all concepts on the right-hand side of concept definitions in $\mathcal{T}'$ are in NNF.

Proposition 5.1. *There is a polynomial time transformation of each acyclic TBox $\mathcal{T}$ into an acyclic TBox $\mathcal{T}'$ in NNF such that, for all concept names A occurring in $\mathcal{T}$, A is satisfiable with respect to $\mathcal{T}$ if and only if A is satisfiable with respect to $\mathcal{T}'$.*

Proof. Let $\mathcal{T}$ be an acyclic TBox. We proceed in three steps:

- For each defined concept name A in $\mathcal{T}$, introduce a fresh concept name $\overline{A}$. Extend $\mathcal{T}$ with the concept definition $\overline{A} \equiv \neg C$, for all $A \equiv C \in \mathcal{T}$.
- Convert the right-hand sides of all concept definitions into NNF in the sense of Chapter 4, not distinguishing between primitive and defined concept names.
- In all concept definitions, replace every subconcept $\neg A$, where A is a defined concept name, with $\overline{A}$.

The resulting TBox $\mathcal{T}'$ is as required. As an exercise, the reader might want to prove correctness of this procedure. □

An additional ingredient in our normal form is that concepts occurring on the right-hand side of concept definitions cannot be deeply nested. An acyclic TBox $\mathcal{T}$ is *simple* if all concept definitions are of the form

$$A \equiv P,\ A \equiv \neg P,\ A \equiv B_1 \sqcap B_2,\ A \equiv B_1 \sqcup B_2,\ A \equiv \exists r.B_1,\ \text{ or } \ A \equiv \forall r.B_1,$$

where P is a primitive concept and B_1, B_2 are defined concept names. This is the normal form used by our algorithm. Observe that every simple TBox is in NNF.

Lemma 5.2. *Let A_0 be a concept name. There is a polynomial time transformation of each acyclic TBox $\mathcal{T}$ into a simple TBox $\mathcal{T}'$ such that A_0 is satisfiable with respect to $\mathcal{T}$ if and only if A_0 is satisfiable with respect to $\mathcal{T}'$.*

Proof. Let A_0 be a concept name and $\mathcal{T}$ an acyclic TBox. By Lemma 5.1, we can assume $\mathcal{T}$ to be in NNF. Apply the following additional modifications:

- To break down a concept definition $A \equiv C_1 \sqcap C_2$, with C or D not a defined concept name, introduce fresh concept names B_1 and B_2, and replace $A \equiv C \sqcap D$ with $A \equiv B_1 \sqcap B_2$, $B_1 \equiv C_1$ and $B_2 \equiv C_2$. Similarly for $A \equiv C \sqcup D$, $A \equiv \exists r.C$, and $A \equiv \forall r.C$.
- Delete each concept definition $A \equiv B$ with B a defined concept name and replace all occurrences of A with B if $A \neq A_0$, and all occurrences of B with A otherwise. □

As justified by Lemmas 5.1 and 5.2, the algorithm for deciding the satisfiability of $\mathcal{ALC}$ concepts with respect to acyclic TBoxes takes as input a concept name A_0 and a simple TBox $\mathcal{T}$. The central notion underlying our algorithm is that of a *type*.

Definition 5.3. Let $\mathcal{T}$ be a simple TBox. Let $\mathsf{Def}(\mathcal{T})$ denote the set of defined concept names in $\mathcal{T}$. A *type* for $\mathcal{T}$ is a set $\tau \subseteq \mathsf{Def}(\mathcal{T})$ such that the following conditions are satisfied:

(i) $A \in \tau$ implies $B \notin \tau$, if $A \equiv P$ and $B \equiv \neg P$ in $\mathcal{T}$;
(ii) $A \in \tau$ implies $B \in \tau$ and $B' \in \tau$, if $A \equiv B \sqcap B' \in \mathcal{T}$;
(iii) $A \in \tau$ implies $B \in \tau$ or $B' \in \tau$, if $A \equiv B \sqcup B' \in \mathcal{T}$.

Intuitively, a type describes the concept memberships of an element d in an interpretation $\mathcal{I}$. This description is partial since we do not require a type to contain, for each defined concept name, either it or its negation (as enforced by the semantics). We could add this requirement,

```
define procedure ALC-worlds(A_0, T)
    i = rd(A_0)
    guess a set τ ⊆ Def_i(T) with A_0 ∈ τ
    recurse(τ, i, T)

define procedure recurse(τ, i, T)
    if τ is not a type for T then return false
    if i = 0 then return true
    for all A ∈ τ with A ≡ ∃r.B ∈ T do
        S = {B} ∪ {B' | ∃A' : A' ∈ τ and A' ≡ ∀r.B' ∈ T}
        guess a set τ' ⊆ Def_{i-1}(T) with S ⊆ τ'
        if recurse(τ', i − 1, T) = false then return false
    return true
```

Fig. 5.1. Algorithm for concept satisfiability with respect to simple TBoxes.

but it is not necessary. Observe that Conditions (ii) and (iii) resemble the tableau rules for dealing with conjunction and disjunction, and that Condition (i) resembles clash-freeness.

The satisfiability algorithm constructs tree models whose depth is bounded by the *role depth* of the input concept name, which describes the nesting depth of existential and universal restrictions in the (unfolded!) definition of the concept name. Formally, we define the role depth of a defined concept name A by induction as follows:

- If $A \equiv (\neg)P \in \mathcal{T}$, then $\mathsf{rd}(A) = 0$.
- If $A \equiv B_1 * B_2 \in \mathcal{T}$ with $* \in \{\sqcap, \sqcup\}$, then $\mathsf{rd}(A) = \max(\mathsf{rd}(B_1), \mathsf{rd}(B_2))$.
- If $A \equiv Q\, r.B \in \mathcal{T}$ with $Q \in \{\exists, \forall\}$, then $\mathsf{rd}(A) = \mathsf{rd}(B) + 1$.

For $i \geq 0$, we define $\mathsf{Def}_i(\mathcal{T}) = \{A \in \mathsf{Def}(\mathcal{T}) \mid \mathsf{rd}(A) \leq i\}$.

The algorithm is given in Figure 5.1. It checks the existence of a tree model $\mathcal{I}$ of A_0 and $\mathcal{T}$, considering one element of $\Delta^{\mathcal{I}}$ in each recursion step. Intuitively, recusive calls correspond to a single application of the tableau rule for existential restrictions, together with multiple applications of the tableau rule for universal restrictions.

To show that the algorithm is correct and terminating and runs in polynomial space, it is convenient to work with recursion trees, which give a structured representation of the recursion calls made during a run of the algorithm. Such a recursion tree is a tuple $T = (V, E, \ell)$, with

(V, E) a tree and ℓ a node-labelling function that assigns with each node $v \in V$ the arguments $\ell(v) = (\tau, i, \mathcal{T})$ of the recursive call corresponding to v. Thus, $(v, v') \in E$ means that the call v' occurred during v.

The depth of the recursion tree is bounded by $\mathsf{rd}(A_0)$ since i is initialised to this value, decremented in each call, and never becomes negative. The outdegree is obviously bounded by the number of concept definitions in $\mathcal{T}$. This gives termination. Since $\mathsf{rd}(A_0)$ is bounded by the size of $\mathcal{T}$ (defined in Section 3.4) and the data stored in each call is polynomial in the size of the input, it also means that the algorithm only needs space polynomial in the size of $\mathcal{T}$. Thus, it remains to prove correctness.

Lemma 5.4. $\mathcal{ALC}$-$\mathsf{worlds}(A_0, \mathcal{T}) = \mathsf{true}$ *if and only if* A_0 *is satisfiable with respect to* $\mathcal{T}$.

Proof. (only if) Let $T = (V, E, \ell)$ be the recursion tree of a successful run of $\mathcal{ALC}$-worlds on A_0 and $\mathcal{T}$, with root $v_0 \in V$. For each node $v \in V \setminus \{v_0\}$, let $\sigma(v)$ be the role name that the **for all** loop was processing when making recursion call v. Set $\Delta^{\mathcal{I}} = V$ and define, for each primitive concept name P and role name r,

$$\begin{aligned} P^{\mathcal{I}} &= \{v \in \Delta^{\mathcal{I}} \mid \exists A : A \in \ell(v) \text{ and } A \equiv P \in \mathcal{T}\} \\ r^{\mathcal{I}} &= \{(v, v') \mid (v, v') \in E \text{ and } \sigma(v') = r\}. \end{aligned}$$

For $A, B \in \mathsf{Def}(\mathcal{T})$, set $A \prec B$ if $A \equiv C \in \mathcal{T}$ and B is a subconcept of C. Let $\prec^+$ be the transitive closure of $\prec$. The interpretation of the defined concept names is defined by induction on $\prec^+$, setting

$$A^{\mathcal{I}} = C^{\mathcal{I}} \text{ if } A \equiv C \in \mathcal{T}.$$

Note that, since $\mathcal{T}$ is acyclic, $C^{\mathcal{I}}$ is well defined when we use it to define $A^{\mathcal{I}}$. Since $\mathcal{I}$ is a model of $\mathcal{T}$ by definition, it remains to show that it is also a model of A_0. To this end, one can prove the following by induction on $\prec^+$.

Claim. *For all* $A \in \mathsf{Def}(\mathcal{T})$ *and all* $v \in V$, $A \in \ell(v)$ *implies* $v \in A^{\mathcal{I}}$.

We leave the detailed proof to the reader and only consider the case $A \equiv \neg P$ as an example. Let $A \in \ell(v)$. By Property (i) of types, there is no $B \in \ell(v)$ with $B \equiv P \in \mathcal{T}$. By definition of $\mathcal{I}$, this yields $v \notin P^{\mathcal{I}}$ as required.

(if) Assume that A_0 is satisfiable with respect to $\mathcal{T}$. Let $\mathcal{I}$ be a model

of A_0 and $\mathcal{T}$, and $d_0 \in A_0^{\mathcal{I}}$. For $d \in \Delta^{\mathcal{I}}$ and $i \leq \mathsf{rd}(A_0)$, set

$$\mathsf{tp}_i(d) = \{A \in \mathsf{Def}_i(\mathcal{T}) \mid d \in A^{\mathcal{I}}\}.$$

We use $\mathcal{I}$ to guide the nondeterministic decisions of the algorithm. To do this, it is convenient to pass an element $d \in \Delta^{\mathcal{I}}$ as a virtual fourth argument to the procedure recurse such that $d \in A^{\mathcal{I}}$ for all A in the first argument τ.

Initially, we guide the algorithm to guess $\mathsf{tp}_{\mathsf{rd}(A_0)}(d_0)$ as the set τ in $\mathcal{ALC}$-worlds. Now let recurse be called with arguments $(\tau, i, \mathcal{T}, d)$, and assume that the **for all** loop is processing $A \in \tau$ with $A \equiv \exists r.B \in \mathcal{T}$. Then $d \in A^{\mathcal{I}}$ and thus there is a $d' \in B^{\mathcal{I}}$ with $(d, d') \in r^{\mathcal{I}}$. We guide the algorithm to guess $\mathsf{tp}_{i-1}(d')$ as the set τ. It remains to show that, when guided in this way, the algorithm returns true. This boils down to showing that all the guessed sets τ are types, which is straightforward using the semantics. □

We have thus proved the following result.

Theorem 5.5. *In $\mathcal{ALC}$, concept satisfiability and subsumption with respect to acyclic TBoxes are in* PSPACE.

Lower Bound

We now prove that the PSPACE upper bound from Theorem 5.5 is optimal by showing that concept satisfiability in $\mathcal{ALC}$ is PSPACE-hard, even without TBoxes. This implies that concept satisfiability is PSPACE-complete, both without TBoxes and with acyclic TBoxes.

The most common way to prove hardness for a complexity class $\mathcal{C}$ is to find an appropriate problem P that is already known to be hard for $\mathcal{C}$ and then to exhibit a polynomial time reduction from P to the problem at hand. In our case, the problem P is related to a game played on formulas of propositional logic and known to be PSPACE-complete [SC79].

A *finite Boolean game* (FBG) is a triple $(\varphi, \Gamma_1, \Gamma_2)$ with φ a formula of propositional logic and $\Gamma_1 \uplus \Gamma_2$ a partition of the variables used in φ into two sets of identical cardinality. The game is played by two players. Intuitively, Player 1 controls the variables in Γ_1 and Player 2 controls the variables in Γ_2. The game proceeds in $n = |\Gamma_1 \uplus \Gamma_2|$ rounds, with the players alternating. We assume that the variables in Γ_1 and Γ_2 are ordered. Player 1 moves first by choosing a truth value for the first variable from Γ_1. In the next round, Player 2 chooses a truth value for the first variable from Γ_2. Next, it is again Player 1's turn, who assigns

a truth value to the second variable in Γ_1, and so on. After n rounds, Player 1 wins the game if the resulting truth assignment satisfies the formula φ; otherwise, Player 2 wins. The decision problem associated with this game is as follows: given a game $(\varphi, \Gamma_1, \Gamma_2)$, decide whether Player 1 has a *winning strategy*, i.e., whether he can force a win no matter what Player 2 does.

Before reducing FBGs to concept satisfiability in $\mathcal{ALC}$, we give a formal definition of winning strategies. Fix a game $G = (\varphi, \Gamma_1, \Gamma_2)$, and let $n = |\Gamma_1 \uplus \Gamma_2|$. We assume that $\Gamma_1 = \{p_1, p_3, \ldots, p_{n-1}\}$ and $\Gamma_2 = \{p_2, p_4, \ldots, p_n\}$. A *configuration* of G is a word $t \in \{0,1\}^i$, for some $i \leq n$. Intuitively, the kth symbol in this word assigns a truth value to the variable p_k. Thus, if the current configuration is t, then a truth value for $p_{|t|+1}$ is selected in the next round. This is done by Player 1 if $|t|$ is even and by Player 2 if $|t|$ is odd. The *initial configuration* is the empty word ε. A *winning strategy* for Player 1 in G is a finite node-labelled tree (V, E, ℓ) of depth n, where ℓ assigns to each node $v \in V$ a configuration $\ell(v)$. We say that a node $v \in V$ is of *depth* i if v is reachable from the root by travelling along i successor edges. Winning strategies are required to satisfy the following conditions:

- the root is labelled with the initial configuration;
- if v is a node of depth $i < n$ with i even and $\ell(v) = t$, then v has one successor v' with $\ell(v') \in \{t0, t1\}$;
- if v is a node of depth $i < n$ with i odd and $\ell(v) = t$, then v has two successors v' and v'' with $\ell(v') = t0$ and $\ell(v'') = t1$;
- if v is a node of depth n and $\ell(v) = t$, then t satisfies φ.

Consider the game $G = (\varphi, \{p_1, p_3\}, \{p_2, p_4\})$, with

$$\varphi = \big(\, \neg p_1 \rightarrow p_2 \,\big) \wedge \big(\, (p_1 \wedge p_2) \rightarrow (p_3 \vee p_4) \,\big) \wedge \big(\, \neg p_2 \rightarrow (p_4 \rightarrow \neg p_3) \,\big).$$

Figure 5.2 shows a winning strategy for Player 1 in G. Intuitively, a winning strategy tells Player 1 how to play the game, no matter what Player 2 does. For example, if the current game configuration is 10, then Player 1 can look into the strategy tree for the (unique!) node $v \in V$ with $\ell(v) = 10$ and at its (unique!) successor v'. It is labelled 100, which advises Player 1 to set the truth value of p_3 to 0.

To reduce the existence of winning strategies in FBGs to the satisfiability of $\mathcal{ALC}$ concepts, we transform a game $G = (\varphi, \Gamma_1, \Gamma_2)$ into an $\mathcal{ALC}$ concept C_G such that Player 1 has a winning strategy in G if and only if C_G is satisfiable. The idea is to craft C_G such that every model

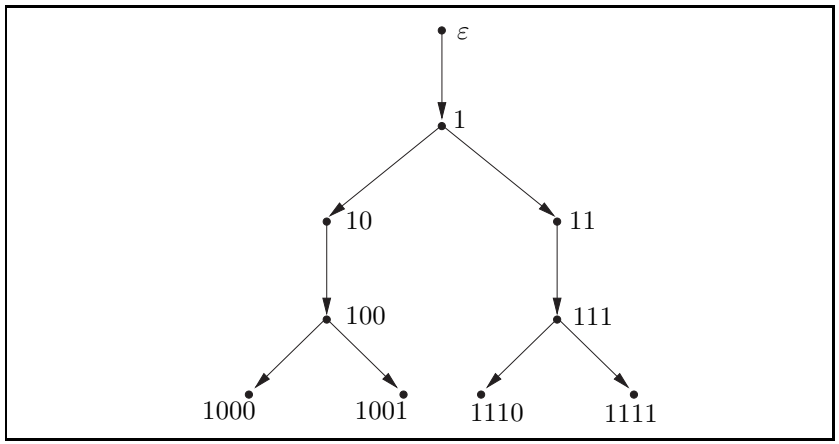

Fig. 5.2. A winning strategy for Player 1 in G.

of C_G describes a winning strategy for Player 1 in G and, vice versa, every such winning strategy gives rise to a model of C_G. The concept C_G uses a single role name r to represent the successor relation of the strategy tree. To describe the values of propositional variables, we use concept names $P_1, \dots, P_n$. Throughout this chapter, we use $C \to D$ as an abbreviation for $\neg C \sqcup D$, for better readability. Now, C_G is a conjunction whose conjuncts we define step by step, along with intuitive explanations of their meaning:

- For each node of odd depth i (i.e., Player 2 is to move), there are two successors, one for each possible truth value of p_{i+1}:

$$C_1 = \bigsqcap_{i \in \{1,3,\dots,n-1\}} \forall r^i.\big(\exists r.\neg P_{i+1} \sqcap \exists r.P_{i+1} \big),$$

  where $\forall r^i.C$ denotes the i-fold nesting $\forall r.\cdots\forall r.C$.
- For each node of even depth i (i.e., Player 1 is to move), there is one successor:

$$C_2 = \bigsqcap_{i \in \{0,2,\dots,n-2\}} \forall r^i.\exists r.\top.$$

  Note that, since P_{i+1} must be either true or false at the generated successor, a truth value for p_{i+1} is chosen "automatically".
- Once a truth value is chosen, it remains fixed:

$$C_3 = \bigsqcap_{1 \le i \le j < n} \forall r^j.\big((P_i \to \forall r.P_i) \sqcap (\neg P_i \to \forall r.\neg P_i) \big).$$

- At the leaves, the formula φ is true:

$$C_4 = \forall r^n.\varphi^*,$$

where φ^* denotes the result of converting φ into an $\mathcal{ALC}$ concept by replacing each p_i with P_i, $\sqcap$ with $\wedge$, and $\sqcup$ with $\vee$.

Now, we define $C_G = C_1 \sqcap \cdots \sqcap C_4$. It is easily verified that the length of C_G is quadratic in n, and that C_G can be constructed in time polynomial in n. The next lemma states that the reduction is correct.

Lemma 5.6. *Player 1 has a winning strategy in G if and only if C_G is satisfiable.*

Proof. (only if) Assume that Player 1 has a winning strategy (V, E, ℓ) with root $v_0 \in V$. We define an interpretation $\mathcal{I}$ by setting

- $\Delta^{\mathcal{I}} = V$,
- $r^{\mathcal{I}} = E$,
- $P_i^{\mathcal{I}} = \{v \in V \mid \ell(v) \text{ sets } p_i \text{ to } 1\}$ for $1 \leq i \leq n$.

We leave it as an exercise to verify that $v_0 \in C_G^{\mathcal{I}}$.

(if) Let $\mathcal{I}$ be a model of C_G, and let $d_0 \in C_G^{\mathcal{I}}$. We define a winning strategy (V, E, ℓ) with $V \subseteq \mathbb{N} \times \Delta^{\mathcal{I}}$. The construction will be such that

$(*)$ if $(i, d) \in V$, then d is reachable from d_0 in $\mathcal{I}$ by travelling i steps along r.

Start by setting $V = \{(0, d_0)\}$, $E = \emptyset$ and $\ell(0, d_0)$ to the initial configuration. We proceed in rounds $1, \ldots, n$. In each odd round i, iterate over all nodes $(i-1, d) \in V$ and do the following:

- select a $d' \in \Delta^{\mathcal{I}}$ such that $(d, d') \in r^{\mathcal{I}}$ (which exists since $d_0 \in C_2^{\mathcal{I}}$ and $(*)$ is satisfied by induction);
- add (i, d') to V, $((i-1, d), (i, d'))$ to E, and set $\ell(i, d') = tj$, where $t = \ell(i-1, d)$ and j is 1 if $d' \in P_i^{\mathcal{I}}$ and 0 otherwise.

In each even round i, iterate over all nodes $(i-1, d) \in V$ and do the following:

- select $d', d'' \in \Delta^{\mathcal{I}}$ such that $d' \in P_i^{\mathcal{I}}$, $d'' \notin P_i^{\mathcal{I}}$ and $\{(d, d'), (d, d'')\} \subseteq r^{\mathcal{I}}$ (which exist since $d_0 \in C_1^{\mathcal{I}}$ and $(*)$ is satisfied by induction);
- add (i, d') and (i, d'') to V, $((i-1, d), (i, d'))$ and $((i-1, d), (i, d''))$ to E, and set $\ell(i, d') = t1$ and $\ell(i, d'') = t0$, where $t = \ell(i-1, d)$.

Since $d_0 \in C_3^{\mathcal{I}}$ and $d_0 \in C_4^{\mathcal{I}}$, it is easy to prove that the resulting tree is a winning strategy for Player 1. □

We have thus established PSPACE-hardness of concept satisfiability in $\mathcal{ALC}$. Together with Theorem 5.5, we obtain the following result.

Theorem 5.7. *In $\mathcal{ALC}$, concept satisfiability and subsumption without TBoxes and with acyclic TBoxes are* PSPACE*-hard, thus* PSPACE*-complete.*

As in the case of $\mathcal{ALC}$, it is rather often the case that satisfiability without TBoxes and with acyclic TBoxes have the same complexity. However, there are also notable exceptions. One example is $\mathcal{ALC}$ extended with concrete domains. For some natural concrete domains, satisfiability without TBoxes is PSPACE-complete, and with respect to acyclic TBoxes it is NEXPTIME-complete.

5.1.2 General TBoxes

The aim of this section is to show that, in $\mathcal{ALC}$, the transition from acyclic TBoxes to general TBoxes increases the computational complexity from PSPACE to EXPTIME.

Upper Bound

We prove an EXPTIME upper bound for satisfiability with respect to general $\mathcal{ALC}$ concepts using a so-called *type elimination* algorithm. The central notion of such an algorithm is that of a type, which is defined similarly to the types introduced in Section 5.1.1.

Let $\mathcal{T}$ be a general TBox. It can be seen that $\mathcal{T}$ is equivalent to the TBox

$$\top \sqsubseteq \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} \neg C \sqcup D :$$

see Point (v) of Lemma 2.16 for a similar observation. We can thus assume without loss of generality that general TBoxes $\mathcal{T}$ have the form $\{\top \sqsubseteq C_{\mathcal{T}}\}$. Moreover, we can assume that $C_{\mathcal{T}}$ is in negation normal form (NNF); compare Chapter 4. As in Section 3.4, we use $\mathsf{sub}(C)$ to denote the set of subconcepts of the concept C. If $\mathcal{T}$ is a TBox, we set $\mathsf{sub}(\mathcal{T}) = \mathsf{sub}(C_{\mathcal{T}})$.

Definition 5.8. Let $\mathcal{T}$ be a general TBox. A *type* for $\mathcal{T}$ is a set $\tau \subseteq \mathsf{sub}(\mathcal{T})$ such that the following conditions are satisfied:

(i) $A \in \tau$ implies $\neg A \notin \tau$, for all $\neg A \in \mathsf{sub}(\mathcal{T})$;
(ii) $C \sqcap D \in \tau$ implies $C \in \tau$ and $D \in \tau$, for all $C \sqcap D \in \mathsf{sub}(\mathcal{T})$;
(iii) $C \sqcup D \in \tau$ implies $C \in \tau$ or $D \in \tau$, for all $C \sqcup D \in \mathsf{sub}(\mathcal{T})$;
(iv) $C_{\mathcal{T}} \in \tau$.

As in Section 5.1.1, a type (partially) describes the concept memberships of a single domain element.

The algorithm takes as input a concept name A_0 and general TBox $\mathcal{T}$ such that A_0 occurs in $\mathcal{T}$. If we want to decide satisfiability of a compound concept C with respect to $\mathcal{T}$, we can simply introduce a fresh concept name A_0, and add $A_0 \sqsubseteq C$ to $\mathcal{T}$. Obviously, the assumption that A_0 occurs in $\mathcal{T}$ can be made without loss of generality. The general idea is that the algorithm generates all types for $\mathcal{T}$, then repeatedly eliminates types that cannot occur in any model of $\mathcal{T}$, and finally checks whether A_0 is contained in one of the surviving types. The following definition serves to make the elimination step more precise.

Definition 5.9. Let Γ be a set of types and $\tau \in \Gamma$. Then τ is *bad* in Γ if there exists an $\exists r.C \in \tau$ such that the set

$$S = \{C\} \cup \{D \mid \forall r.D \in \tau\}$$

is no subset of any type in Γ.

Intuitively, a type τ is bad in Γ if there is an existential restriction $\exists r.C$ that cannot be satisfied in any interpretation in which the type of all domain elements is from Γ. Note the similarity of the set S in the above definition and the set S generated by the algorithm in Section 5.1.1. In both cases, the purpose is a combined treatment of existential and universal restrictions.

The algorithm is given in Figure 5.3. The algorithm terminates and runs in exponential time since (i) the number of types for $\mathcal{T}$ is exponential in the size of $\mathcal{T}$, (ii) in each execution of the **repeat** loop, at least one type is eliminated, and (iii) computing the set Γ_i inside the **repeat** loop can be done in time polynomial in the cardinality of Γ_{i-1} (thus in time exponential in the size of $\mathcal{T}$). Next, we prove correctness.

Lemma 5.10. $\mathcal{ALC}$-$\mathsf{Elim}(A_0, \mathcal{T}) = \mathsf{true}$ *if and only if* A_0 *is satisfiable with respect to* $\mathcal{T}$.

Proof. (only if) Assume that $\mathcal{ALC}$-$\mathsf{Elim}(A_0, \mathcal{T})$ returns true, and let Γ_i be the set of remaining types. Then there is a $\tau_0 \in \Gamma_i$ such that $A_0 \in \tau_0$. Define an interpretation $\mathcal{I}$ as follows:

define procedure $\mathcal{ALC}$-$\mathsf{Elim}(A_0, \mathcal{T})$
  set Γ_0 to the set of all types for $\mathcal{T}$
  $i = 0$
  repeat
    $i = i + 1$
    $\Gamma_i = \{\tau \in \Gamma_{i-1} \mid \tau \text{ is not bad in } \Gamma_{i-1}\}$
  until $\Gamma_i = \Gamma_{i-1}$
  if there is $\tau \in \Gamma_i$ with $A_0 \in \tau$ **then return** true
  else return false

Fig. 5.3. Algorithm for concept satisfiability with respect to general TBoxes.

- $\Delta^{\mathcal{I}} = \Gamma_i$,
- $A^{\mathcal{I}} = \{\tau \in \Gamma_i \mid A \in \tau\}$,
- $r^{\mathcal{I}} = \{(\tau, \tau') \in \Gamma_i \times \Gamma_i \mid \forall r.C \in \tau \text{ implies } C \in \tau'\}$.

By induction on the structure of C, we can prove, for all $C \in \mathsf{sub}(\mathcal{T})$ and all $\tau \in \Gamma_i$, that $C \in \tau$ implies $\tau \in C^{\mathcal{I}}$. Most cases are straightforward, using the definition of $\mathcal{I}$ and the induction hypothesis. We only do the case $C = \exists r.D$ explicitly:

- Let $\exists r.D \in \tau$. Since τ has not been eliminated from Γ_i, it is not bad. Thus, there is a $\tau' \in \Gamma_i$ such that

$$\{C\} \cup \{D \mid \forall r.D \in \tau\} \subseteq \tau'.$$

  By definition of $\mathcal{I}$, we have $(\tau, \tau') \in r^{\mathcal{I}}$. Since $\tau' \in C^{\mathcal{I}}$ by induction hypothesis, we obtain $\tau \in (\exists r.C)^{\mathcal{I}}$ by the semantics.

By Condition (iv) from Definition 5.8, we thus have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all $C \sqsubseteq D \in \mathcal{T}$. Hence, $\mathcal{I}$ is a model of $\mathcal{T}$. Since $A_0 \in \tau_0$, it is also a model of A_0.

(if) If A_0 is satisfiable with respect to $\mathcal{T}$, then there is a model $\mathcal{I}$ of A_0 and $\mathcal{T}$. Let $d_0 \in A_0^{\mathcal{I}}$. For all $d \in \Delta^{\mathcal{I}}$, set

$$\mathsf{tp}(d) = \{C \in \mathsf{sub}(\mathcal{T}) \mid d \in C^{\mathcal{I}}\}.$$

Define $\Psi = \{\mathsf{tp}(d) \mid d \in \Delta^{\mathcal{I}}\}$ and let $\Gamma_0, \Gamma_1, \ldots, \Gamma_k$ be the sequence of type sets computed by $\mathcal{ALC}$-$\mathsf{Elim}(A_0, \mathcal{T})$. It is possible to prove by induction on i that no type from Ψ is ever eliminated from any set Γ_i, for $i \leq k$. Since $A_0 \in \mathsf{tp}(d_0) \in \Psi$, the algorithm returns "true". □

This finishes the proof of the upper bound.

Theorem 5.11. *In $\mathcal{ALC}$, concept satisfiability and subsumption with respect to general TBoxes are in* ExpTime.

Lower Bound

Our objective is to establish an ExpTime lower bound for concept satisfiability in $\mathcal{ALC}$ with respect to general TBoxes. In Section 5.1.1, a PSpace lower bound for concept satisfiability in $\mathcal{ALC}$ without TBoxes was proved by reducing the existence of winning strategies for finite Boolean games (FBGs). To show ExpTime-hardness with general TBoxes, we use a similar kind of game, which proceeds over an infinite number of rounds.

An *infinite Boolean game* (IBG) is a tuple $(\varphi, \Gamma_1, \Gamma_2, t_0)$ with φ a formula of propositional logic, $\Gamma_1 \uplus \Gamma_2$ a partition of the variables used in φ, and t_0 an initial truth assignment for the variables in $\Gamma_1 \uplus \Gamma_2$. The game is played by two players, with Player 1 controlling the variables in Γ_1 and Player 2 controlling the variables in Γ_2. The game starts in configuration t_0 and Player 1 moves first. The players alternate, in each move choosing a variable they control and flipping its truth value. A skip move, in which all variables retain their truth values, is also allowed. Player 1 wins the game if the formula φ ever becomes true (no matter which player moved to make this happen). Player 2 wins if he manages to keep the game running forever, without φ ever becoming true.

Thus, the main difference between this game and the one in Section 5.1.1 is that players are not forced to choose variables in a fixed ordering. In particular, the same variable can be chosen more than once during the same game, and thus the game may continue indefinitely. Deciding the existence of a winning strategy is ExpTime-complete, for both Player 1 and Player 2. In the reduction to $\mathcal{ALC}$ concept satisfiability, it is much easier to describe winning strategies for Player 2. Thus, the decision problem associated with IBGs is to decide, given a game $(\varphi, \Gamma_1, \Gamma_2, t_0)$, whether Player 2 has a winning strategy. We formally define such strategies in what follows.

Fix a game $G = (\varphi, \Gamma_1, \Gamma_2, t_0)$. A *configuration* of G has the form (i, t) with $i \in \{1, 2\}$ the player to move next and t a truth assignment for all variables in $\Gamma_1 \uplus \Gamma_2$. The *initial* configuration is $(1, t_0)$. A truth assignment t' is a *p-variation* of a truth assignment t, for $p \in \Gamma_1 \cup \Gamma_2$, if $t' = t$ or t' is obtained from t by flipping the truth value of p. It is a *Γ_i-variation* of t if it is a p-variation of t for some $p \in \Gamma_i$, $i \in \{1, 2\}$. A *winning strategy* for Player 2 in G is an infinite node-labelled tree

(V, E, ℓ), where ℓ assigns to each node $v \in V$ a configuration $\ell(v)$ such that

- the root is labelled with the initial configuration;
- if $\ell(v) = (2, t)$, then v has one successor v' with $\ell(v') = (1, t')$, t' a Γ_2-variation of t;
- if $\ell(v) = (1, t)$, then v has successors $v_0, \dots, v_{|\Gamma_1|}$, $\ell(v_i) = (2, t_i)$ for $i < |\Gamma_1|$, such that $t_0, \dots, t_{|\Gamma_1|}$ are all Γ_1-variations of t;
- if $\ell(v) = (i, t)$, then t does not satisfy φ.

Note that every configuration in which Player 1 is to move has $|\Gamma_1| + 1$ successors: one for each variable in Γ_1 that he can choose to flip and one for the skip move. In contrast to the finite strategies used in Section 5.1.1, the strategies above are trees in which every branch is infinite.

To reduce the existence of winning strategies for Player 2 in IBGs to satisfiability with respect to general $\mathcal{ALC}$ TBoxes, we transform a game instance $G = (\varphi, \Gamma_1, \Gamma_2, t_0)$ into a TBox $\mathcal{T}_G$ and select a concept name I such that Player 2 has a winning strategy in G if and only if I is satisfiable with respect to $\mathcal{T}_G$. Similarly to what was done in Section 5.1.1, the idea is that every joint model of I and $\mathcal{T}_G$ describes a winning strategy for Player 2 in G and, vice versa, every such winning strategy gives rise to a model of I and $\mathcal{T}_G$. Let $\Gamma_1 = \{p_1, \dots, p_m\}$ and $\Gamma_2 = \{p_{m+1}, \dots, p_n\}$. The TBox $\mathcal{T}_G$ uses a single role name r to represent the edges of the strategy tree, concept names $P_1, \dots, P_n$ to describe truth values of the variables, T_1, T_2 to describe whether it is the turn of Player 1 or Player 2, and $F_1, \dots, F_n$ to indicate which variable has been flipped in order to reach the current configuration. We now assemble $\mathcal{T}_G$:

- The initial configuration is as required:

$$I \sqsubseteq T_1 \sqcap \bigsqcap_{1 \le i \le n,\ t_0(p_i)=0} \neg P_i \ \sqcap \bigsqcap_{1 \le i \le n,\ t_0(p_i)=1} P_i.$$

- If it is the turn of Player 1, then there are $|\Gamma_1|+1$ successors:

$$T_1 \sqsubseteq \exists r.(\neg F_0 \sqcap \dots \sqcap \neg F_{n-1}) \sqcap \bigsqcap_{1 \le i \le m} \exists r.F_i.$$

- If it is the turn of Player 2, then there is one successor:

$$T_2 \sqsubseteq \exists r.(\neg F_0 \sqcap \dots \sqcap \neg F_{n-1}) \sqcup \bigsqcup_{m < i \le n} \exists r.F_i.$$

- At most one variable is flipped in each move:

$$\top \sqsubseteq \bigsqcap_{1 \le i < j \le n} \neg(F_i \sqcap F_j).$$

- Variables that are flipped change their truth value:

$$\top \sqsubseteq \bigsqcap_{1 \le i \le n} \big(\, (\, P_i \to \forall r.(F_i \to \neg P_i) \,) \sqcap (\, \neg P_i \to \forall r.(F_i \to P_i) \,) \, \big).$$

- Variables that are not flipped keep their truth value:

$$\top \sqsubseteq \bigsqcap_{1 \le i \le n} \big(\, (\, P_i \to \forall r.(\neg F_i \to P_i) \,) \sqcap (\, \neg P_i \to \forall r.(\neg F_i \to \neg P_i) \,) \, \big).$$

- The players alternate:

$$T_1 \sqsubseteq \forall r.T_2 \qquad \text{and} \qquad T_2 \sqsubseteq \forall r.T_1.$$

- The formula φ is never satisfied:

$$\top \sqsubseteq \neg \varphi^*,$$

  where φ^* denotes the result of converting φ into an $\mathcal{ALC}$ concept by replacing each p_i with P_i, $\sqcap$ with $\wedge$, and $\sqcup$ with $\vee$.

The TBox $\mathcal{T}_G$ is simply the set of all the TBox statements listed above. It is easily verified that the size of $\mathcal{T}_G$ is polynomial in that of G, and that $\mathcal{T}_G$ can be computed from G in polynomial time. The next lemma states that the reduction is correct.

Lemma 5.12. *Player 2 has a winning strategy in G if and only if I is satisfiable with respect to $\mathcal{T}_G$.*

The proof is similar to that of Lemma 5.6. Details are left as an exercise. Thus, we have established the desired ExpTime lower bound. Together with Theorem 5.11, we obtain the following.

Theorem 5.13. *In $\mathcal{ALC}$, concept satisfiability and subsumption with respect to general TBoxes are* ExpTime*-hard, thus* ExpTime*-complete.*

Comparing Theorem 5.13 with the PSpace results obtained in Section 5.1.1, one may wonder whether it is the particular shape of acyclic TBoxes or their acyclicity that makes reasoning with them easier than with general TBoxes. We show here that the latter is the case. Let a *classical TBox* $\mathcal{T}$ be an acyclic TBox with the acyclicity condition dropped; that is, all statements in $\mathcal{T}$ are of the form $A \equiv C$ or $A \sqsubseteq C$ with A a concept name, and left-hand sides have to be unique. For example, $\mathcal{T} = \{A \equiv \exists r.A\}$ is a classical TBox, but not an acyclic one. We

show that concept satisfiability and subsumption with respect to classical TBoxes is not simpler than with respect to general TBoxes, namely ExpTime-complete. To do this, it suffices to observe that satisfiability of concepts with respect to general TBoxes can be polynomially reduced to satisfiability of concepts with respect to classical TBoxes.

Lemma 5.14. *Let $\mathcal{T}$ be a general $\mathcal{ALC}$ TBox, C an $\mathcal{ALC}$ concept and A a concept name not appearing in $\mathcal{T}$ and C. Then C is satisfiable with respect to $\mathcal{T}$ if and only if it is satisfiable with respect to the classical TBox*

$$\mathcal{T}' = \{\, A \equiv \neg A \sqcup (\bigsqcap_{C \sqsubseteq D \in \mathcal{T}} C \to D) \,\}.$$

Proof. (only if) Let $\mathcal{I}$ be a model of C and $\mathcal{T}$, and let $\mathcal{J}$ be obtained from $\mathcal{I}$ by setting $A^{\mathcal{J}} = \Delta^{\mathcal{I}}$. It is easily seen that $\mathcal{J}$ is a model of C and $\mathcal{T}'$.

(if) Let $\mathcal{I}$ be a model if C and $\mathcal{T}'$. We show that $\mathcal{I}$ is also a model of $\mathcal{T}$. Take a $C \sqsubseteq D \in \mathcal{T}$ and a $d \in \Delta^{\mathcal{I}}$. Assume $d \notin A^{\mathcal{I}}$. Then reading the concept definition in $\mathcal{T}'$ from right to left, we get $d \in A^{\mathcal{I}}$, which is a contradiction. Thus $d \in A^{\mathcal{I}}$. Now read the same concept definition from left to right to deduce that $d \in (C \to D)^{\mathcal{I}}$. Since this holds independently of the choice of $C \sqsubseteq D$ and d, we conclude that $\mathcal{I}$ is a model of $\mathcal{T}$. □

5.2 Concept satisfiability beyond $\mathcal{ALC}$

Adding more expressive power to $\mathcal{ALC}$ sometimes leads to an increase in computational complexity, and sometimes not. For example, the DLs $\mathcal{ALCI}$, $\mathcal{ALCQ}$ and $\mathcal{ALCIQ}$ introduced in Chapter 3 behave like $\mathcal{ALC}$: reasoning is PSpace-complete without TBoxes and with acyclic TBoxes, and it is ExpTime-complete with general TBoxes. In this section, we review two extensions of $\mathcal{ALC}$ that are less well behaved. We prove only lower bounds to illustrate the complications introduced by the additional constructors. For corresponding upper bounds, we refer to the literature.

5.2.1 $\mathcal{ALC}$ with inverse roles and nominals

Recall that $\mathcal{ALCOI}$ is the extension of $\mathcal{ALC}$ with inverse roles and nominals. In this DL, satisfiability with respect to general TBoxes has the same complexity as in $\mathcal{ALC}$, namely ExpTime-complete. Interestingly,

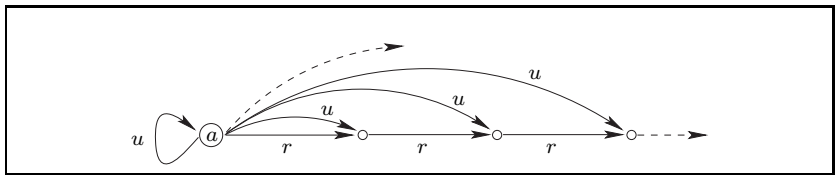

Fig. 5.4. A model of the concept C.

the complexity of concept satisfiability in $\mathcal{ALCOI}$ remains ExpTime-complete if we disallow TBoxes or allow only acyclic TBoxes, and thus $\mathcal{ALCOI}$ is more difficult than $\mathcal{ALC}$ in these cases. The increase in complexity is due to an interaction between inverse roles and nominals that leads to a more complex model theory. For example, in $\mathcal{ALC}$ (and $\mathcal{ALCI}$ and $\mathcal{ALCO}$), it is not possible to enforce that a model contains an infinite (possibly cyclic) r-chain for a role r without using a general TBox. In $\mathcal{ALCOI}$, this is easy:

$$C = \{a\} \sqcap \exists u.\{a\} \sqcap \forall u.\exists r.\exists u^{-}.\{a\}.$$

This concept enforces an infinite r-chain, as shown in Figure 5.4. PSpace algorithms such as $\mathcal{ALC}$-World cannot deal with such models since they rely on polynomially depth-bounded models. A variation of the above concept can be used for proving ExpTime-hardness.

Theorem 5.15. *In $\mathcal{ALCOI}$, concept satisfiability and subsumption (without TBoxes) are* ExpTime*-hard.*

Proof. We reduce satisfiability of $\mathcal{ALC}$ concepts with respect to general TBoxes. Let C be an $\mathcal{ALC}$ concept and $\mathcal{T}$ a general $\mathcal{ALC}$ TBox. Let $r_0, \dots, r_{k-1}$ be all role names that occur in C and $\mathcal{T}$ and their inverses. Construct an $\mathcal{ALCOI}$ concept

$$D = C \sqcap \{a\} \sqcap \exists u.\{a\} \sqcap \forall u.\Big(\bigsqcap_{C \sqsubseteq D \in \mathcal{T}} C \to D\Big) \sqcap \forall u.\Big(\bigsqcap_{i<k} \forall r_i.\exists u^{-}.\{a\}\Big),$$

where u is a fresh role name. Then C is satisfiable with respect to $\mathcal{T}$ if and only if D is satisfiable.

(only if) Let $\mathcal{I}$ be a model of C and $\mathcal{T}$, and let $d_0 \in C^{\mathcal{I}}$. Modify $\mathcal{I}$ by setting $a^{\mathcal{I}} = d_0$ and $u^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. It is easily seen that the modified interpretation is a model of D.

(if) Let $\mathcal{I}$ be a model of D, and $d_0 \in D^{\mathcal{I}}$. Let $\Delta^{\mathcal{J}}$ be the restriction of $\Delta^{\mathcal{I}}$ to those elements d such that d is reachable from d_0 by travelling an arbitrary number of steps along roles $r_0, \dots, r_{k-1}$, and let $\mathcal{J}$ be the

restriction of $\mathcal{I}$ to $\Delta^{\mathcal{J}}$. To make sure that all nominals are mapped to the restricted domain, put $b^{\mathcal{J}} = d_0$ for all individual names b (note that, consequently, $a^{\mathcal{J}} = a^{\mathcal{I}} = d_0$). By induction on the structure of E, it is possible to prove the following.

Claim. *For all $\mathcal{ALCOI}$ concepts E that contain no nominals except $\{a\}$ and all $d \in \Delta^{\mathcal{J}}$, we have $d \in E^{\mathcal{I}}$ if and only if $d \in E^{\mathcal{J}}$.*

By this claim, $d_0 \in D^{\mathcal{J}}$, and thus $d_0 \in C^{\mathcal{J}}$. It thus remains to prove that $\mathcal{J}$ is a model of $\mathcal{T}$. Let $E \sqsubseteq F \in \mathcal{T}$ and $d \in E^{\mathcal{J}}$. By the claim, $d \in E^{\mathcal{I}}$. Since d is reachable from d_0 along the roles $r_0, \dots, r_{k-1}$ and by definition of D, $d \in (E \to F)^{\mathcal{I}}$, and thus $d \in F^{\mathcal{I}}$. By applying the claim once more, we get $d \in F^{\mathcal{J}}$ as required. □

It is interesting that a single nominal suffices to prove this lower bound.

5.2.2 Further adding number restrictions

If we extend the description logic $\mathcal{ALCOI}$ from the previous section with qualifying number restrictions, the computational complexity further increases. In the resulting DL $\mathcal{ALCOIQ}$, concept satisfiability is NExpTime-complete whether or not TBoxes are present. Indeed, satisfiability of $\mathcal{ALCOIQ}$ concepts with respect to general TBoxes can be polynomially reduced to satisfiability of $\mathcal{ALCOIQ}$ concepts without TBoxes using the construction from the proof of Theorem 5.15, and thus satisfiability with and without TBoxes is of identical complexity. In this section, we prove NExpTime-hardness of satisfiability in $\mathcal{ALCOIQ}$ with respect to general TBoxes, and thus also without TBoxes.

Being closely related to the two-variable fragment of first-order logic extended with counting quantifiers, $\mathcal{ALCOIQ}$ has a more subtle model theory than the description logics that we have been concerned with so far. In particular, $\mathcal{ALCOIQ}$ concepts can enforce interpretations that are not tree-shaped. This is exploited in the subsequent proof, which is by a reduction of a NExpTime-complete version of the tiling problem and involves enforcing interpretations that have the shape of a torus. On an intuitive level, this tiling problem can be framed as follows. A *tile* is of square shape and has coloured edges. A *tile type* is a way to colour the edges of a tile. We are given a finite number of tile types, and have an unlimited supply of tiles of each type available. Additionally, we are given an initial sequence of tiles $t_0, \dots, t_{n-1}$. The problem is whether

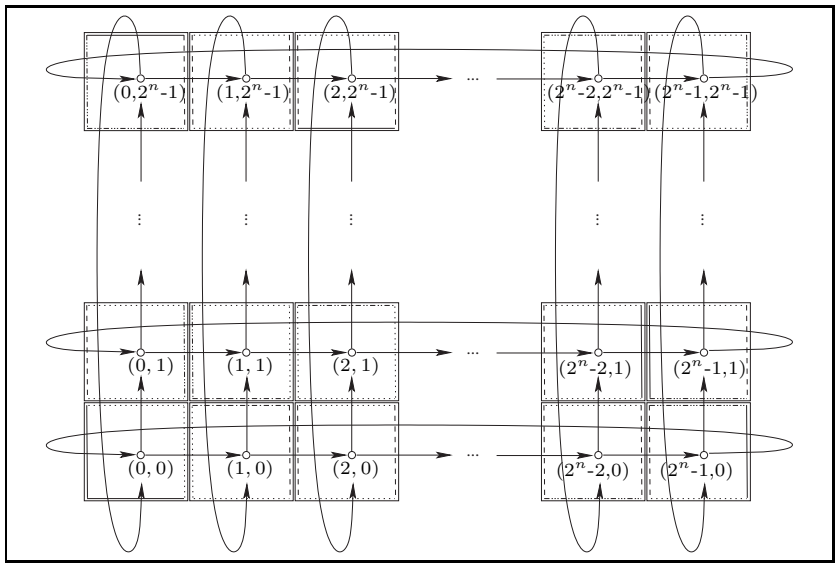

Fig. 5.5. An illustration of a tiling, where we use line styles instead of colours.

we can produce a tiling of the torus of size $2^{n+1} \times 2^{n+1}$ such that (i) all horizontally or vertically adjacent tiles are coloured identically at their touching edges and (ii) the position $(i, 0)$ is covered with t_i, for all $i < n$. See Figure 5.5 for an illustration.

Definition 5.16. A *torus tiling problem* P is a triple (T, H, V), where T is a finite set of *tile types* and $H, V \subseteq T \times T$ represent the horizontal and vertical matching conditions. Let P be a torus tiling problem and $c = t_0 \cdots t_{n-1} \in T^n$ an *initial condition*. A mapping

$$\tau : \{0, \dots, 2^n - 1\} \times \{0, \dots, 2^n - 1\} \to T$$

is a *solution* for P and c if and only if, for all $i, j < 2^n$, the following hold:

- if $\tau(i, j) = t$ and $\tau(i \oplus_{2^n} 1, j) = t'$, then $(t, t') \in H$;
- if $\tau(i, j) = t$ and $\tau(i, j \oplus_{2^n} 1) = t'$, then $(t, t') \in V$;
- $\tau(i, 0) = c_i$ for $i < n$,

where $\oplus_i$ denotes addition modulo i.

We want to reduce the torus tiling problem to satisfiability of $\mathcal{ALCOIQ}$ concepts with respect to general TBoxes. To this end, we show how to convert a torus tiling problem P and initial condition c

into a TBox $\mathcal{T}_{P,c}$ such that P and c have a solution if and only if $\top$ is satisfiable with respect to $\mathcal{T}_{P,c}$ (that is, if $\mathcal{T}_{P,c}$ has any model at all). The construction is such that models of $\mathcal{T}_{P,c}$ take the form of a torus that encodes solutions for P and c, and conversely, each such solution gives rise to a torus-shaped model of $\mathcal{T}_{P,c}$.

Let $P = (T, H, V)$ and $c = t_0 \cdots t_{n-1}$. We now show how to assemble the TBox $\mathcal{T}_{P,c}$. With every domain element d from a model $\mathcal{I}$ of $\mathcal{T}_{P,c}$, we associate a position $(x_d, y_d) \in \{0, \dots, 2^n - 1\} \times \{0, \dots, 2^n - 1\}$ in the torus. To this end, we introduce concept names $X_0, \dots, X_{n-1}$ and $Y_0, \dots, Y_{n-1}$. For any domain element d, we identify x_d with the number whose binary representation has a one in the ith position if $d \in X_i^{\mathcal{I}}$ and a zero otherwise (for all $i < n$); y_d is defined in the same way using the concept names $Y_0, \dots, Y_{n-1}$, where in both cases we assume that the least significant bit is at position 0. To represent neighbourhood in the torus, we use role names r_x (horizontal neighbours to the right) and r_y (vertical neighbours to the top).

We now describe the desired behaviour of the concept and role names just introduced, starting by saying that every node in the torus has a (right) horizontal neighbour and an (upper) vertical neighbour:

$$\top \sqsubseteq \exists r_x.\top \sqcap \exists r_y.\top.$$

Next, we synchronise the positions represented by the concept names $X_0, \dots, X_{n-1}, Y_0, \dots, Y_{n-1}$ with the neighbourhoods represented by the role names r_x, r_y. When travelling along r_x the vertical position should not change, and likewise for r_y and the horizontal position:

$$\begin{array}{llll} Y_i \sqsubseteq \forall r_x.Y_i & \text{and} & \neg Y_i \sqsubseteq \forall r_x.\neg Y_i & \text{for all } i \leq n, \\ X_i \sqsubseteq \forall r_y.X_i & \text{and} & \neg X_i \sqsubseteq \forall r_y.\neg X_i & \text{for all } i \leq n. \end{array}$$

It is slightly more complicated to ensure that the horizontal position is incremented when travelling along r_x and likewise for the vertical position and r_y. We start with r_x:

$$\begin{array}{ll} \bigsqcap_{j<i} X_j \sqsubseteq (X_i \to \forall r_x.\neg X_i) \sqcap (\neg X_i \to \forall r_x.X_i) & \text{for all } i < n, \\ \bigsqcup_{j<i} \neg X_j \sqsubseteq (X_i \to \forall r_x.X_i) \sqcap (\neg X_i \to \forall r_x.\neg X_i) & \text{for all } i < n. \end{array}$$

These GCIs capture binary incrementation in a straightforward way: if bits 0 to $i-1$ are all one, then bit i is flipped; if bits 0 to $i-1$ include at least one 0-bit, then bit i retains its value. Note that in the case that $i = 0$, the conjunction on the left-hand side of the top-most GCI is

empty (thus equivalent to $\top$) and so is the disjunction on the left-hand side of the second GCI (which is thus equivalent to $\bot$). We also add the corresponding statements for incrementation of the vertical position along r_y, which are analogous:

$$\bigsqcap_{j<i} Y_j \sqsubseteq (Y_i \to \forall r_y.\neg Y_i) \sqcap (\neg Y_i \to \forall r_y.Y_i) \quad \text{for all } i < n,$$

$$\bigsqcup_{j<i} \neg Y_j \sqsubseteq (Y_i \to \forall r_y.Y_i) \sqcap (\neg Y_i \to \forall r_y.\neg Y_i) \quad \text{for all } i < n.$$

To represent tile types, we introduce a concept name A_t for each $t \in T$. Every node in the torus carries exactly one tile type and the initial condition $c = t_0 \cdots t_{n-1}$ is satisfied by this tiling:

$$\begin{aligned} \top &\sqsubseteq \bigsqcup_{t\in T} A_t \sqcap \bigsqcup_{t,t'\in T, t\neq t'} \neg(A_t \sqcap A_{t'}), \\ A_{t_i} &\sqsupseteq \bigsqcap_{j<n, \mathsf{bit}_j(i)=0} \neg X_j \sqcap \bigsqcap_{j<n, \mathsf{bit}_j(i)=1} X_j \sqcap \bigsqcap_{j<n} \neg Y_j \quad \text{for all } i < n, \end{aligned}$$

where $\mathsf{bit}_j(i)$ denotes the jth bit of the binary representation of i. We must also ensure that the horizontal matching conditions H and vertical matching conditions V of our torus tiling problem P are satisfied:

$$\top \sqsubseteq \bigsqcup_{(t,t')\in H} (A_t \sqcap \forall r_x.A_{t'}) \sqcap \bigsqcup_{(t,t')\in V} (A_t \sqcap \forall r_y.A_{t'}).$$

With what we have added to the TBox $\mathcal{T}_{P,c}$ so far, have we captured the torus tiling problem sufficiently well to make the reduction work? It is easy to see that this is not the case; that is, there are models of $\mathcal{T}_{P,c}$ which do not take the shape of a torus: there may be multiple nodes that represent the same torus position, there can be nodes with an $r_x r_y$-successor (first follow an r_x-edge, then an r_y-edge) that is not an $r_y r_x$-successor and so on. In fact, this is not surprising since so far we have only used the description logic $\mathcal{ALC}$, but no inverse roles, no qualified number restrictions and no nominals. It might thus seem that we have quite a bit of coding effort still ahead of us. Interestingly, this is not the case and it is very easy to finish the reduction at this point. We simply have to say that the inverses of the roles r_x and r_y are functional and that the grid position $(2^n - 1, 2^n - 1)$ occurs at most once, for which we use a single nominal a:

$$\begin{aligned} \top &\sqsubseteq (\leqslant 1\, r_x^-.\top) \sqcap (\leqslant 1\, r_y^-.\top), \\ \{a\} &\sqsupseteq X_0 \sqcap \cdots \sqcap X_{n-1} \sqcap Y_0 \sqcap \cdots \sqcap Y_{n-1}. \end{aligned}$$

Why is this sufficient? Let $\mathcal{I}$ be a model of $\mathcal{T}_{P,c}$. The crucial point to observe is that

(∗) for each torus position (i, j), there is a unique element $d \in \Delta^{\mathcal{I}}$ with $(x_d, y_d) = (i, j)$.

In fact, because of the GCI $\top \sqsubseteq \exists r_x.\top \sqcap \exists r_y.\top$ and the synchronisation of the roles r_x, r_y with the concept names $X_0, \dots, X_{n-1}, Y_0, \dots, Y_{n-1}$, our TBox $\mathcal{T}_{P,c}$ ensures that there is *at least* one such d; the reader might like to attempt a formal proof. Additionally, we can prove by induction on $2^n - (i + j)$ that, for every torus position (i, j), there is also *at most* one $d \in \Delta^{\mathcal{I}}$ with $(x_d, y_d) = (i, j)$. The induction start is easy because $d = a^{\mathcal{I}}$ is the only element with $(x_d, y_d) = (2^n - 1, 2^n - 1)$. For the induction step, assume that $(x_d, y_d) = (x_e, y_e) = (i, j)$. We have to show that $d = e$. First assume that $i < 2^n - 1$. Then there is a d' with $(d, d') \in r_x^{\mathcal{I}}$ and $(x_{d'}, y_{d'}) = (i + 1, j)$ and an e' with $(e, e') \in r_x^{\mathcal{I}}$ and $(x_{e'}, y_{e'}) = (i + 1, j)$. By induction hypothesis, $d' = e'$. Because r_x^- is functional, we have $d = e$ as required. If $i = 2^n - 1$, then we must have $j < 2^n - 1$ and can argue analogously using the functionality of r_y^-.

We have thus established (∗). It is now possible to show that $\mathcal{I}$ is isomorphic to the $(2^n - 1, 2^n - 1)$-torus in the expected sense, but in fact this is not necessary since (∗) is essentially all that is needed to establish correctness of the reduction.

Lemma 5.17. *$\mathcal{T}_{P,c}$ has a model if and only if there exists a solution for P and c.*

Proof. (only if) Let $\mathcal{I}$ be a model of $\mathcal{T}_{P,c}$. For all $(i, j) \in \{0, \dots, 2^n - 1\} \times \{0, \dots, 2^n - 1\}$, set $\tau(i, j) = t$ if and only if there is a $d \in \Delta^{\mathcal{I}}$ with $(x_d, y_d) = (i, j)$ and $d \in A_t^{\mathcal{I}}$. By (∗) and since $\mathcal{T}_{P,c}$ ensures that every $d \in \Delta^{\mathcal{I}}$ is in the extension of exactly one concept A_t, τ is a well-defined and total function. It remains to argue that τ is a solution for P and c. Let us first show satisfaction of the horizontal matching condition H of P and consider a torus position (i, j) with $i < 2^n - 1$. By (∗), there are unique $d, e \in \Delta^{\mathcal{I}}$ with $(x_d, y_d) = (i, j)$ and $(x_e, y_e) = (i+1, j)$. Moreover, we must have $(d, e) \in r_x^{\mathcal{I}}$ because $d \in (\exists r_x.\top)^{\mathcal{I}}$, by (∗), and since $\mathcal{T}_{P,c}$ enforces that any r_x-successor e' of d must satisfy $(d', e') = (i + 1, j)$. Since H is satisfied in $\mathcal{I}$ along the role r_x, it is thus also satisfied by τ. The vertical matching condition V can be treated similarly. Moreover, by definition of τ and because of the GCI in $\mathcal{T}_{P,c}$ that deals with the initial condition c, it is clear that τ satisfies c.

(if) Let τ be a solution for P and c. Define an interpretation $\mathcal{I}$ as

follows:

$$\begin{aligned}
\Delta^{\mathcal{I}} &= \{0,\dots,2^n-1\}\times\{0,\dots,2^n-1\},\\
r_x^{\mathcal{I}} &= \{((i,j),(i+1,j)) \mid i<2^n-1, j<2^n\},\\
r_y^{\mathcal{I}} &= \{((i,j),(i,j+1)) \mid i<2^n, j<2^n-1\},\\
A_t^{\mathcal{I}} &= \{(i,j)\in\Delta^{\mathcal{I}} \mid \tau(i,j)=t\} \text{ for all } t\in T.
\end{aligned}$$

By going through the GCIs contained in it, it can be verified that $\mathcal{I}$ is a model of $\mathcal{T}_{P,c}$. □

The size of $\mathcal{T}_{P,c}$ is polynomial in n and satisfiability of $\mathcal{ALCOIQ}$ concepts with respect to TBoxes can be reduced to satisfiability without TBoxes, so we obtain the following theorem.

Theorem 5.18. *In $\mathcal{ALCOIQ}$, concept satisfiability and subsumption (without TBoxes) are* NExpTime*-hard.*

It is interesting to note that, to obtain this result, we do not need the full expressive power of qualified number restrictions. Indeed, all we need to say is that two roles are functional.

5.3 Undecidable extensions of $\mathcal{ALC}$

We consider two extensions of $\mathcal{ALC}$ in which satisfiability and subsumption are undecidable. Since Description Logic research aims at sound, complete and terminating algorithms, it is a commonly held opinion that constructors which lead to undecidability should not be included in a description logic, or only in a weakened form that is computationally better behaved.

5.3.1 Role value maps

Suppose that we are constructing a TBox about universities, which includes the statements

$$\begin{aligned}
\mathsf{Course} &\sqsubseteq \exists \mathsf{held\text{-}at}.\mathsf{University},\\
\mathsf{Lecturer} &\sqsubseteq \exists \mathsf{teaches}.\mathsf{Course} \sqcap \exists \mathsf{employed\text{-}by}.\mathsf{University}.
\end{aligned}$$

To improve our knowledge base, we may want to express that if someone teaches a course held at a university, then he is employed by that specific university. This is not possible in $\mathcal{ALC}$ (which can be proved using the tree model property; see Section 3.5), but it can easily be done in the extension of $\mathcal{ALC}$ with so-called *role value maps* (RVMs), which come in

two flavours: local and global. A *local* role value map is a new concept constructor with the syntax $(r_1 \circ \cdots \circ r_k \sqsubseteq s_1 \circ \cdots \circ s_\ell)$, where $r_1, \ldots, r_k$ and $s_1, \ldots, s_\ell$ are role names. To define the semantics, let

$$(r_1 \cdots r_k)^{\mathcal{I}}(d_0) = \{d_k \in \Delta^{\mathcal{I}} \mid \exists d_1, \ldots, d_k : (d_i, d_{i+1}) \in r_i^{\mathcal{I}} \text{ for } 0 \leq i < k\}.$$

Then we define

$$(r_1 \circ \cdots \circ r_k \sqsubseteq s_1 \circ \cdots \circ s_\ell)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid (r_1 \cdots r_k)^{\mathcal{I}}(d) \subseteq (s_1 \cdots s_\ell)^{\mathcal{I}}(d)\}.$$

In the *global* version of role value maps, $(r_1 \circ \cdots \circ r_k \sqsubseteq s_1 \circ \cdots \circ s_\ell)$ is not a concept constructor, but an expression that may occur in the TBox. The semantics of such an expression can be defined in terms of local RVMs: an interpretation $\mathcal{I}$ *satisfies* a global RVM $(r_1 \circ \cdots \circ r_k \sqsubseteq s_1 \circ \cdots \circ s_\ell)$ if it satisfies the TBox statement $\top \sqsubseteq (r_1 \circ \cdots \circ r_k \sqsubseteq s_1 \circ \cdots \circ s_\ell)$. In the initial example, we could now express the desired property using the global RVM

$$(\mathsf{teaches} \circ \mathsf{held\text{-}at} \sqsubseteq \mathsf{employed\text{-}by}).$$

Local role value maps were already present in the very first description logic system, KL-ONE. Several years after the invention of KL-ONE, it was proved that satisfiability in the underlying DL is undecidable, and that the reason for this is the presence of role value maps. In the following, we prove that satisfiability in $\mathcal{ALC}$ extended with RVMs is undecidable, whether or not the local or global version is used, and whether or not TBoxes are admitted.

We first show undecidability of satisfiability in $\mathcal{ALC}$ extended with global RVMs, and in the presence of general TBoxes. The proof is by a reduction of an undecidable version of the tiling problem: compare Section 5.2.2. The main differences are that (i) we tile $\mathbb{N} \times \mathbb{N}$, the first quadrant of the plane, instead of an exponentially sized torus; and (ii) there is no initial condition.

Definition 5.19. A *tiling problem* P is a triple (T, H, V), where T is a finite set of *tile types* and $H, V \subseteq T \times T$ represent the *horizontal and vertical matching conditions*. A mapping $\tau : \mathbb{N} \times \mathbb{N} \to T$ is a *solution* for P if and only if for all $i, j \geq 0$, the following hold:

- if $\tau(i, j) = t$ and $\tau(i + 1, j) = t'$, then $(t, t') \in H$;
- if $\tau(i, j) = t$ and $\tau(i, j + 1) = t'$, then $(t, t') \in V$.

Let $P = (T, H, V)$ be a tiling problem. We construct a general TBox $\mathcal{T}_P$ with global RVMs such that models of $\mathcal{T}_P$ represent solutions

to P. In $\mathcal{T}_P$, we use concept names A_t, $t \in T$, to represent the tiling, and role names r_x and r_y to represent the horizontal and vertical successor relations between positions in the plane. The TBox $\mathcal{T}_P$ consists of the following parts:

(i) Every position has a horizontal and a vertical successor:

$$\top \sqsubseteq \exists r_x.\top \sqcap \exists r_y.\top.$$

(ii) Every position is labelled with exactly one tile type:

$$\top \sqsubseteq \bigsqcup_{t \in T} A_t \sqcap \bigsqcap_{t,t' \in T, t \neq t'} \neg(A_t \sqcap A_{t'}).$$

(iii) Adjacent tiles satisfy the matching conditions:

$$\top \sqsubseteq \bigsqcup_{(t,t') \in H} (A_t \sqcap \forall r_x.A_{t'}) \ \sqcap \ \bigsqcup_{(t,t') \in V} (A_t \sqcap \forall r_y.A_{t'}).$$

(iv) Every $r_x r_y$-successor is also a $r_y r_x$-successor and vice versa:

$$(r_x \circ r_y \sqsubseteq r_y \circ r_x),$$
$$(r_y \circ r_x \sqsubseteq r_x \circ r_y).$$

This finishes the construction of $\mathcal{T}_P$. Note that we have not enforced that the horizontal and vertical successors are unique (which they are in the first quadrant of the plane). Interestingly, this is not necessary for the reduction to be correct.

Lemma 5.20. *$\top$ is satisfiable with respect to $\mathcal{T}_P$ if and only if P has a solution.*

Proof. (only if) Let $\mathcal{I}$ be a model of $\mathcal{T}_P$. We construct a mapping $f : \mathbb{N} \times \mathbb{N} \to \Delta^{\mathcal{I}}$ such that, for all $i, j \geq 0$, $(f(i,j), f(i+1,j)) \in r_x^{\mathcal{I}}$ and $(f(i,j), f(i,j+1)) \in r_y^{\mathcal{I}}$, proceeding in two steps. First, we cut out a "staircase", i.e., define $f(i,j)$ for all $i, j \in \mathbb{N}$ such that $j \in \{i, i-1\}$:

- set $f(0,0)$ to an arbitrary element of $\Delta^{\mathcal{I}}$;
- if $f(i,i)$ was defined last, select a $d \in \Delta^{\mathcal{I}}$ with $(f(i,i), d) \in r_x^{\mathcal{I}}$, and set $f(i+1,i) = d$;
- if $f(i,i-1)$ was defined last, select a $d \in \Delta^{\mathcal{I}}$ with $(f(i,i-1), d) \in r_y^{\mathcal{I}}$, and set $f(i,i) = d$.

The required elements d exist since $\mathcal{I}$ is a model of $\mathcal{T}_P$. In the second step, we complete the construction of f as follows:

- if $f(i,j)$, $f(i+1,j)$ and $f(i+1,j+1)$ are defined and $f(i,j+1)$ is undefined, select a $d \in \Delta^{\mathcal{I}}$ with $(f(i,j),d) \in r_y^{\mathcal{I}}$ and $(d, f(i+1,j+1)) \in r_x^{\mathcal{I}}$, and set $f(i,j+1) = d$;
- if $f(i,j)$, $f(i,j+1)$ and $f(i+1,j+1)$ are defined and $f(i+1,j)$ is undefined, select a $d \in \Delta^{\mathcal{I}}$ with $(f(i,j),d) \in r_x^{\mathcal{I}}$ and $(d, f(i+1,j+1)) \in r_y^{\mathcal{I}}$, and set $f(i+1,j) = d$.

The required elements d exist due to the role value maps in $\mathcal{T}$. Intuitively, the mapping f "cuts out of $\mathcal{I}$" a representation of the first quadrant in which horizontal and vertical successors are unique. Now define a mapping $\tau : \mathbb{N} \times \mathbb{N} \to T$ by setting $\tau(i,j) = t$ if $f(i,j) \in A_t$. It is easily verified that this mapping is well defined and a solution for P.

(if) Let τ be a solution for P. Define an interpretation $\mathcal{I}$ as follows:

$$\begin{aligned} \Delta^{\mathcal{I}} &= \mathbb{N} \times \mathbb{N}, \\ r_x^{\mathcal{I}} &= \{((i,j),(i+1,j)) \mid i,j \geq 0\}, \\ r_y^{\mathcal{I}} &= \{((i,j),(i,j+1)) \mid i,j \geq 0\}, \\ A_t^{\mathcal{I}} &= \{(i,j) \mid \tau(i,j) = t\} \text{ for all } t \in T. \end{aligned}$$

Clearly, $\mathcal{I}$ is a model of $\mathcal{T}_P$ and we are done. □

We have thus shown the following.

Theorem 5.21. *In $\mathcal{ALC}$ with global role value maps, concept satisfiability and subsumption with respect to general TBoxes are undecidable.*

Next, we strengthen this result by showing that satisfiability in $\mathcal{ALC}$ is undecidable even if general TBoxes are not admitted and global RVMs are replaced with local ones.

Theorem 5.22. *In $\mathcal{ALC}$ with local role value maps, concept satisfiability and subsumption (without TBoxes) are undecidable.*

Proof. The proof is by reduction from the satisfiability of $\mathcal{ALC}$ concepts with respect to general TBoxes and global role value maps. Let C be an $\mathcal{ALC}$ concept and $\mathcal{T}$ a general TBox with global role value maps. Let Γ be the set of all role names used in C and $\mathcal{T}$. Introduce a fresh role name u and define the concept D as the conjunction of the following:

- the concept $\exists u.C$ to generate an instance of C;
- the concept $\bigsqcap_{r \in \Gamma} (u \circ r \sqsubseteq u)$ to ensure that the element that satisfies D reaches all other "relevant" elements of the model by travelling a single step along u;

- the concept

$$\forall u.\Big(\bigsqcap_{C \sqsubseteq D \in \mathcal{T}} C \to D \sqcap \bigsqcap_{(r_1 \circ \cdots \circ r_k \sqsubseteq s_1 \circ \cdots \circ s_\ell) \in \mathcal{T}} (r_1 \circ \cdots \circ r_k \sqsubseteq s_1 \circ \cdots \circ s_\ell) \Big),$$

  which guarantees that all concept inclusions and global RVMs from $\mathcal{T}$ are satisfied.

The proof that D is satisfiable if and only if C is satisfiable with respect to $\mathcal{T}$, which we leave as an exercise, bears some similarity to the proof of Theorem 5.15. □

There are not many ways to regain decidability in the presence of role value maps. Note, though, that role hierarchies are a special case of role value maps where sequences of roles are restricted to length one.

5.3.2 Concrete domains

In many applications of DLs, it is also necessary to describe concrete qualities of objects, such as the age of people, which is most appropriately represented by a non-negative integer value. Enabling such representations is the purpose of an extension of Description Logic known as concrete domains. In fact, concrete domains give rise to a class of extensions of a given DL rather than only to a single extension, depending on which concrete qualities we allow to be represented and how we can compare them using predicates. In this section, we show that satisfiability in $\mathcal{ALC}$ with general TBoxes becomes undecidable when we add a seemingly simple concrete domain based on the non-negative numbers, with a unary predicate for equality to zero and a binary predicate for incrementation.

A *concrete domain* is a pair $\mathsf{D} = (\Delta^{\mathsf{D}}, \Phi^{\mathsf{D}})$, where Δ^{D} is a non-empty set and Φ^{D} is a finite set of predicates. Each *predicate* in Φ^{D} has a name P, an arity k_P and an extension $P^{\mathsf{D}} \subseteq (\Delta^{\mathsf{D}})^{k_P}$. The concrete domain used in this section is called D_{+1}. It is defined as $\mathsf{D}_{+1} = (\mathbb{N}, \Phi^{\mathsf{D}_{+1}})$, where $\Phi^{\mathsf{D}_{+1}}$ consists of the unary predicate $=_0$ associated with the extension $(=_0)^{\mathsf{D}_{+1}} = \{0\}$ and the binary predicate $+1$ associated with the extension $(+1)^{\mathsf{D}_{+1}} = \{(i, j) \in \mathbb{N} \times \mathbb{N} \mid j = i + 1\}$.

To integrate a concrete domain D into a description logic (in this case $\mathcal{ALC}$), we introduce *abstract features* and *concrete features*, as additional sorts. Every interpretation $\mathcal{I}$ assigns to each abstract feature g a partial function $g^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}}$ and to each concrete feature h a partial function

$h^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathsf{D}}$. Note that an abstract feature is nothing but a role name whose interpretation is restricted to be a partial function. We then add a new concept constructor called a *predicate restriction*, taking the form $\exists c_1, \ldots, c_k.P$, where P is the name of a predicate from Φ^{D} of arity k and each c_i is a *feature chain*, that is, a sequence $g_1 \cdots g_n h$ of $n \geq 0$ abstract features g_i and one concrete feature h. For example, the following GCI expresses that every human has an age and a father who has a larger age:

$$\mathsf{Human} \sqsubseteq \exists \mathsf{age}, \mathsf{father}\,\mathsf{age}.\ >,$$

where age is a concrete feature, father is an abstract feature and we assume a concrete domain based on the non-negative integers that includes the predicate $>$. We use $\mathcal{ALC}(\mathsf{D})$ to express the extension of $\mathcal{ALC}$ with the concrete domain D.

To prove undecidability of satisfiability in $\mathcal{ALC}(\mathsf{D}_{+1})$ with respect to general TBoxes, we use a reduction from the halting problem of two-register machines. A two-register machine M is similar to a Turing machine. It also has states, but instead of a tape, it has two registers which contain non-negative integers. In one step, the machine can increment the content of one of the registers or test whether the content of the given register is zero and, if not, then decrement it. In the second case, the successor state depends on whether the tested register was zero or not. There is a designated halting state, and M halts if it encounters that state.

Definition 5.23. A (deterministic) *two-register machine* (2RM) is a pair $M = (Q, P)$ with $Q = \{q_0, \ldots, q_\ell\}$ a set of *states* and $P = I_0, \ldots, I_{\ell-1}$ a sequence of *instructions*. By definition, q_0 is the *initial state* and q_ℓ the *halting state*. For all $i < \ell$,

- either $I_i = +(p, q_j)$ is an *incrementation instruction* with $p \in \{1, 2\}$ a register and q_j the subsequent state; or
- $I_i = -(p, q_j, q_k)$ is a *decrementation instruction* with $p \in \{1, 2\}$ a register, q_j the subsequent state if register p contains 0, and q_k the subsequent state otherwise.

A *configuration* of M is a triple (q, m, n), with q the current state and $m, n \in \mathbb{N}$ the register contents. We write $(q_i, n_1, n_2) \Rightarrow_M (q_j, m_1, m_2)$ if one of the following holds:

- $I_i = +(p, q_j)$, $m_p = n_p + 1$ and $m_{\overline{p}} = n_{\overline{p}}$, where $\overline{1} = 2$ and $\overline{2} = 1$;
- $I_i = -(p, q_j, q_k)$, $n_p = m_p = 0$ and $m_{\overline{p}} = n_{\overline{p}}$;

- $I_i = -(p, q_k, q_j)$, $n_p > 0$, $m_p = n_p - 1$ and $m_{\overline{p}} = n_{\overline{p}}$.

The *computation* of M on input $(n, m) \in \mathbb{N}^2$ is the unique longest configuration sequence $(p_0, n_0, m_0) \Rightarrow_M (p_1, n_1, m_1) \Rightarrow_M \cdots$ such that $p_0 = q_0$, $n_0 = n$ and $m_0 = m$.

The halting problem for 2RMs is to decide, given a 2RM M, whether its computation on input $(0, 0)$ is finite (which implies that its last state is q_ℓ). We reduce this problem to the *un*satisfiability of $\mathcal{ALC}(\mathsf{D}_{+1})$ concepts with respect to general TBoxes by transforming a 2RM $M = (Q, P)$ into a TBox $\mathcal{T}_M$ and selecting a concept name I such that I is unsatisfiable with respect to $\mathcal{T}_M$ if and only if M halts. More precisely, every model of I and $\mathcal{T}_M$ describes an infinite computation of M on $(0, 0)$ and, conversely, every such computation gives rise to a model of I and $\mathcal{T}_M$. We use a single abstract feature g to describe the relation $\Rightarrow_M$, concrete features h_1 and h_2 to describe the register content, and concept names $Q_0, \ldots, Q_\ell$ for the states. For convenience, we first use an additional binary equality predicate $=$ (with the obvious extension), which is not actually contained in $\Phi^{\mathsf{D}_{+1}}$, and later show how to replace it. We define the TBox $\mathcal{T}_M$ step by step, along with explanations:

- We start in state q_0 and with the registers containing zero:

$$I \sqsubseteq Q_0 \sqcap \exists h_1.{=_0} \sqcap \exists h_2.{=_0}.$$

- Incrementation is executed correctly; that is, for all $I_i = +(p, q_j)$,

$$Q_i \sqsubseteq \exists g.Q_j \sqcap \exists h_p, gh_p.{+_1} \sqcap \exists h_{\overline{p}}, gh_{\overline{p}}.{=}.$$

  Observe that all the existential restrictions talk about the same g-filler since g is functional.
- Decrementation is executed correctly; that is, for all $I_i = -(p, q_j, q_k)$

$$\begin{array}{rcl} Q_i \sqcap \exists h_p.{=_0} & \sqsubseteq & \exists g.Q_j \sqcap \exists h_p, gh_p.{=} \sqcap \exists h_{\overline{p}}, gh_{\overline{p}}.{=}, \\ Q_i \sqcap \neg\exists h_p.{=_0} & \sqsubseteq & \exists g.Q_j \sqcap \exists gh_p, h_p.{+_1} \sqcap \exists h_{\overline{p}}, gh_{\overline{p}}.{=}. \end{array}$$

  Observe that we have swapped the arguments to $+_1$ to simulate a predicate $-_1$.
- The halting state q_ℓ is never reached, and thus the computation is infinite:

$$\top \sqsubseteq \neg Q_\ell.$$

It is not difficult to prove the following result.

Lemma 5.24. *The computation of M on $(0,0)$ is finite if and only if I is unsatisfiable with respect to $\mathcal{T}_M$.*

To establish undecidability of $\mathcal{ALC}(\mathsf{D}_{+1})$, it thus remains to show how to eliminate the binary equality predicate. The idea is to replace an equality test with repeated decrementations and tests for zero. We only treat the example concept $\exists h_1, h_2.=$. To eliminate it, we can replace it with a new concept name M, and then add the following, where g' is a new abstract feature:

$$M \quad \sqsubseteq \quad (\exists h_1.=_0 \sqcap \exists h_2.=_0) \sqcup (\exists g'.M \sqcap \exists g' h_1, h_1.+_1 \sqcap \exists g' h_2, h_2.+_1).$$

Note that we again use $+_1$ to describe *de*crementation. It is now easy to remove binary equality from the above reduction, and we obtain the following result.

Theorem 5.25. *In $\mathcal{ALC}(\mathsf{D}_{+1})$, concept satisfiability and subsumption with respect to general TBoxes are undecidable.*

There are a number of ways to overcome undecidability of $\mathcal{ALC}$ extended with concrete domains. First, we can move to acyclic TBoxes, which results in decidability for a large number of concrete domains, including D_{+1}. Second, we may select concrete domains more carefully to achieve decidability even in the presence of general TBoxes. An example of such a more well-behaved concrete domain is based on the real numbers, with binary predicates for the comparisons $<$, $\leq$, $=$, $\neq$, $\geq$, and $>$, and unary predicates for the same comparisons with any fixed rational number. Third, we can stipulate that the concrete domain operator may contain only concrete features, but no sequences composed of abstract and concrete features. Then we obtain decidability even with general TBoxes and expressive concrete domains. The drawback is that it is no longer possible to relate the data values of *different* domain elements.

5.4 Historical context and literature review

PSPACE-completeness of satisfiability in $\mathcal{ALC}$ was first observed in the seminal paper by Schmidt-Schauß and Smolka, [SS91], which studied concept satisfiability and subsumption without TBoxes. Independently and previously, Ladner and others had proved that satisfiability in the modal logic K is PSPACE-complete [Lad77, HM92]. Later, Schild observed that $\mathcal{ALC}$ is a notational variant of K [Sch91], and thus the two mentioned results are identical. It has long been common knowledge in

the DL community that the PSPACE upper bound can be extended to acyclic TBoxes; a published proof can be found in [Lut99]. Schmidt-Schauß and Smolka proved the PSPACE upper bound using a tableau-style algorithm. Our $\mathcal{ALC}$-worlds algorithm is an adaptation of the K-worlds algorithm for deciding satisfiability in the modal logic K, which goes back to Ladner [Lad77]. In both [SS91] and [Lad77], the PSPACE lower bound is proved using a reduction of the validity problem for quantified Boolean formulas (QBFs). This problem is very closely related to our finite Boolean games, taken from [SM73, Sch78] (the game is called G_ω(CNF) in the latter). We have preferred to use finite Boolean games because they allow us to use a very similar problem, namely infinite Boolean games, for proving EXPTIME-hardness for the case with general TBoxes.

The EXPTIME upper bound for $\mathcal{ALC}$ with general TBoxes is a consequence of Schild's observation, mentioned above, and containment of propositional dynamic logic in EXPTIME, which was established by Fischer and Ladner [FL79]. The lower bound also follows from the corresponding bound for PDL, although in this case it is necessary to carefully analyze the proof, as has been done by Schild [Sch91]. The type elimination algorithm that we use for the upper bound was first used in the context of PDL, namely by Pratt [Pra79]. The original and most common way to establish the lower bound is by a reduction of the word problem for exponentially space-bounded alternating Turing machines [FL79]. Our infinite Boolean games are from [SC79] (where they are called G_5).

EXPTIME-completeness of $\mathcal{ALCOI}$ was first observed in modal logic. More precisely, description logics with nominals correspond to (a simple version of) so-called hybrid logics, and $\mathcal{ALCOI}$ is a fragment of hybrid logic with backwards modalities. It was shown by Areces *et al.* [ABM99] that this fragment is EXPTIME-complete, using for the hardness part the same approach that is followed in this chapter. The NEXPTIME-hardness of $\mathcal{ALCOIQ}$ was first established by Tobies [Tob99], who also gives a matching upper bound. As has been mentioned, $\mathcal{ALCOIQ}$ is closely related to the two-variable fragment of first-order logic extended with counting quantifiers in which satisfiability is also NEXPTIME-complete; see [GOR97, PST97, Pra09].

The undecidability of $\mathcal{ALC}$ with role value maps was first shown by Schmidt-Schauß [Sch89] using a reduction of the word problem for groups. This actually yields a stronger result than the one presented in this chapter because only the following constructors are needed: con-

junction, value restriction and role value maps based on equality (instead of inclusion). The undecidability of $\mathcal{ALC}(\mathsf{D}_{+1})$ with general TBoxes was proved in [Lut02] using a reduction of the post correspondence problem (PCP), building on a result by Baader and Hanschke [BH91].

All of the upper complexity bounds established in this section extend to KB consistency. Technically, this is typically not a big challenge. An interesting exception is presented in [DLNS94], where it is shown that, on the fragment $\mathcal{ALE}$ of $\mathcal{ALC}$, KB consistency is PSPACE-complete while satisfiability is only coNP-complete.