Description Logics: a Nice Family of Logics — Complexity, Part 2 —

Uli Sattler[1] *Thomas Schneider*[2]

[1]School of Computer Science, University of Manchester, UK

[2]Department of Computer Science, University of Bremen, Germany

ESSLLI, 18 August 2016

Goal for today

Automated reasoning plays an **important role** for DLs.

- It allows the development of intelligent applications.
- The expressivity of DLs is strongly tailored towards this goal.

Requirements for automated reasoning:

- Decidability of the relevant decision problems
- Low complexity if possible
- Algorithms that perform well in practice

Yesterday & today: 1 & 3

Now: 2

Plan for today

1. EXPTIME-membership
2. EXPTIME-hardness
3. PSPACE-hardness (not covered in the lecture)
4. Undecidability and a NEXPTIME lower bound $\rightsquigarrow$ Uli

Thanks to Carsten Lutz for much of the material on these slides.

And now . . .

1. EXPTIME-membership
2. EXPTIME-hardness
3. PSPACE-hardness (not covered in the lecture)
4. Undecidability and a NEXPTIME lower bound $\rightsquigarrow$ Uli

EXPTIME-membership

We start with an ExpTime upper bound
for concept satisfiability in $\mathcal{ALC}$ relative to TBoxes.

Theorem

The following problem is in ExpTime.

Input: an $\mathcal{ALC}$ concept C_0 and an $\mathcal{ALC}$ TBox $\mathcal{T}$
Question: is there a model $\mathcal{I} \models \mathcal{T}$ with $C_0^{\mathcal{I}} \neq \emptyset$?

We'll use a technique known from modal logic:
type elimination [Pratt 1978]

The basis is a *syntactic* notion of a *type*.

Syntactic types

We assume that

- the input concept C_0 is in NNF
- the input TBox is $\mathcal{T} = \{\top \sqsubseteq C_{\mathcal{T}}\}$ with $C_{\mathcal{T}}$ in NNF

Let $\mathsf{sub}(C_0, \mathcal{T})$ be the set of subconcepts of C_0 and $C_{\mathcal{T}}$.
A **type for C_0 and $\mathcal{T}$** is a subset $t \subseteq \mathsf{sub}(C_0, \mathcal{T})$ such that

1. $A \in t$ iff $\neg A \notin t$ for all $\neg A \in \mathsf{sub}(C_0, \mathcal{T})$
2. $C \sqcap D \in t$ iff $C \in t$ and $D \in t$ for all $C \sqcap D \in \mathsf{sub}(C_0, \mathcal{T})$
3. $C \sqcup D \in t$ iff $C \in t$ or $D \in t$ for all $C \sqcup D \in \mathsf{sub}(C_0, \mathcal{T})$
4. $C_{\mathcal{T}} \in t$

Intuition:
Types describe domain elements completely, up to $\mathsf{sub}(C_0, \mathcal{T})$. •

General idea

General idea of type elimination for input C_0, $\mathcal{T}$:

- Generate all types for C_0 and $\mathcal{T}$ (exponentially many).
- Repeatedly eliminate types that cannot occur in any model of C_0 and $\mathcal{T}$.
- Check whether some type containing C_0 has survived.
- If yes, return "satisfiable"; otherwise "unsatisfiable".

Bad types

The precise notion of "types that cannot occur in a model":

Let Γ be a set of types and $t \in \Gamma$.
Then t is **bad in Γ** if, for some $\exists r.C \in t$,

there is no $t' \in \Gamma$ with $\{C\} \cup \{D \mid \forall r.D \in t\} \subseteq t'$.

Intuition:
A type is bad if it contains an existential restriction
for which there is no "witness" in Γ. •

The type elimination algorithm

procedure $\mathcal{ALC}$-Elim$(C_0, \mathcal{T})$

compute Γ_0: set of all types for C_0 and $\mathcal{T}$

$i := 0$

repeat

$i := i + 1$

$\Gamma_i := \{t \in \Gamma_{i-1} \mid t \text{ not bad in } \Gamma_{i-1}\}$

until $\Gamma_i = \Gamma_{i-1}$

if there is $t \in \Gamma_i$ with $C_0 \in t$ **then**

return "satisfiable"

else return "unsatisfiable"

Termination

. . . is easy (cf. the tableau algorithm):

Lemma

$\mathcal{ALC}$-Elim$(C_0, \mathcal{T})$ terminates after $2^{\mathcal{O}(|C_0|+|\mathcal{T}|)}$ steps.

Proof.

- There are $2^{|C_0|+|\mathcal{T}|}$ subsets of $\text{sub}(C_0, \mathcal{T})$.
- Thus Γ_0 has at most $2^{|C_0|+|\mathcal{T}|}$ elements, and all of them can be generated in time $2^{\mathcal{O}(|C_0|+|\mathcal{T}|)}$.
- Determining whether a type t is bad in Γ takes time polynomial in $|t| \cdot |\Gamma|$.
- Since $\Gamma_0 \supseteq \cdots \supseteq \Gamma_i \supseteq \Gamma_{i+1}$, there are at most $2^{|C_0|+|\mathcal{T}|}$ iterations. □

Soundness and completeness

Lemma

$\mathcal{ALC}$-Elim$(C_0, \mathcal{T})$ returns "satisfiable" **iff** C_0 is satisfiable w.r.t. $\mathcal{T}$.

Proof of "⇒"

- Construct model $\mathcal{I}$ from the set of remaining types Γ:

$$\Delta^{\mathcal{I}} = \Gamma \qquad A^{\mathcal{I}} = \{t \mid A \in t\}$$
$$r^{\mathcal{I}} = \{(t, t') \mid \forall r.D \in t \text{ implies } D \in t'\}$$

- The following claim can be proven inductively, exploiting the fact that no type in Γ is bad:

  For all $t \in \Gamma$ and $C \in \text{sub}(C_0, \mathcal{T})$: $C \in t$ **iff** $t \in C^{\mathcal{I}}$

- $\mathcal{I} \models \mathcal{T}$ because all types contain $C_{\mathcal{T}}$ and using the claim
- $C_0^{\mathcal{I}} \neq \emptyset$ because some type containing C_0 has survived

Soundness and completeness

Lemma

$\mathcal{ALC}$-Elim$(C_0, \mathcal{T})$ returns "satisfiable" **iff** C_0 is satisfiable w.r.t. $\mathcal{T}$.

Proof of "⇐"

- Let $\mathcal{I} \models \mathcal{T}$ with $d_0 \in C_0^{\mathcal{I}}$.
- For every $d \in \Delta^{\mathcal{I}}$, let $\mathbf{tp(d)} = \{C \in \text{sub}(C_0, \mathcal{T}) \mid d \in C^{\mathcal{I}}\}$, i.e., the type of d in $\mathcal{I}$ relative to C_0 and $\mathcal{T}$.
- Let $\boldsymbol{\Gamma} = \{\text{tp}(d) \mid d \in \Delta^{\mathcal{I}}\}$, i.e., the set of all types occurring in $\mathcal{I}$.
- Since $\mathcal{I}$ is a model, no type in Γ can be bad.
- Now by easy induction: $\Gamma \subseteq \Gamma_i$ for every i; hence all types in Γ survive.
- Since $C_0 \in \text{tp}(d_0) \in \Gamma$, the algo. returns "satisfiable". □

Harvest

Direct consequence from the previous lemmas:

Theorem

Satisfiability of and subsumption between $\mathcal{ALC}$ concepts w.r.t. TBoxes is decidable in ExpTime.

Type elimination for more expressive DLs

Type elimination can be extended to give the same ExpTime upper bound for

- $\mathcal{ALCI}$

  Exercise: Extend the definition of "bad" to inverse roles.
  Adapt the soundness and completeness proof.
  Verify the new algorithm on the following inputs.

  - $C_0 = \top,\ \mathcal{T} = \{\top \sqsubseteq \exists r.A \sqcap \forall r^-.\neg A\}$
  - $C_0 = \top,\ \mathcal{T} = \{\top \sqsubseteq \exists r^-.A \sqcap \forall r.\neg A\}$

- $\mathcal{ALCQI}$

Comparison with tableau algorithms

Similarities

- $\sqcap$-, $\sqcup$-, and GCI-rule are captured by the definition of a type.
- $\exists$- and $\forall$-rule are captured by the definition of "bad".
- Clash-freeness is captured by the definition of a type.

Difference

- In the worst case, type elimination runs in exponential time; the tableau algorithm in exponential space (and 2-exp. time).

Should we thus prefer type elimination over tableaux for reasoning over real-world ontologies? **No!**

- Type elimination requires exp. time in the **best case** (for every input: initial generation of all types).
- Tableau algorithms lend themselves to numerous **optimisations** avoiding the worst-case behaviour for many real-world inputs.

Tight complexity bounds for $\mathcal{ALC}$

Can we do better than ExpTime?

No, concept satisfiability and subsumption w.r.t. $\mathcal{ALC}$ TBoxes is **ExpTime-complete**.

We'll next show the lower bound (hardness).

And now ...

Lower bound

Standard approach to showing **ExpTime-hardness**:

Reduction from the word problem for polynomially space-bounded alternating Turing machines

More convenient approach:

Reduction from a related, but much more intuitive problem: a game-theoretic problem whose **ExpTime-completeness** is known

EXPTIME games

Two players $1, 2$ play on a given propositional formula φ.

Every propositional variable in φ belongs to exactly one player.

The game begins with a given initial assignment π_0 of the vars.

Player 1 begins; the players take turns.

In every move the player changes the truth value of **one** of her variables **or passes.**

Player 1 wins as soon as φ becomes true (no matter which player's turn it is).

Player 2 wins if the game continues **infinitely** without φ becoming true.

Example

Consider $\varphi = (p \leftrightarrow \neg q) \wedge (p' \leftrightarrow \neg q')$

where P1 owns p, p' and P2 owns q, q', and $\pi_0 = \begin{smallmatrix} p & q & p' & q' \\ 0 & 0 & 0 & 0 \end{smallmatrix}$.

A run of the game where Player 1 wins:

- P1 sets p to 1 $\rightsquigarrow$ $\pi = 1000$
- P2 passes (obviously he hasn't understood the game yet! :-))
- P1 sets p' to 1 $\rightsquigarrow$ $\pi = 1010$ WIN!

A run of the game where Player 2 wins:

- Whenever P1 changes the value of p (or p'), in the subsequent move P2 changes the value of q (or q').
- Whenever P1 passes, so does P2 in the subsequent move.

This ensures that, always, either p and q or p' and q' have the same truth value $\rightsquigarrow$ P1 cannot win!

EXPTIME games: basics

Basic notions

- We represent a **game** as a tuple $(\varphi, \Gamma_1, \Gamma_2, \pi_0)$ with
  - Γ_1, Γ_2 a partitioning of the variables in φ
  - π_0 the initial assignment
- A **configuration** is a pair (i, π)
  with $i \in \{1, 2\}$ the active player and π an assignment.
  (a "snapshot" of the running game)
- π is a ***j*-variation** of π' for $j \in \{1, 2\}$ if
  $\pi = \pi'$ or π and π' differ in exactly one variable $p \in \Gamma_j$.
  (i.e., Player j can transform π into π' or vice versa)

The decision problem from which we will reduce refers to **winning strategies.**

Winning strategies

Intuitively:

- A winning strategy specifies a next move for Player 2 in every possible situation of the game.
  (situation = a finite sequence of the players' previous moves)
- If Player 2 has a winning strategy, then she can win the game regardless of Player 1's moves.

To take into account the "history" of game situations, winning strategies are represented using trees.

Winning strategies

A **winning strategy** for Player **2** in the game $(\varphi, \Gamma_1, \Gamma_2, \pi_0)$
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 $(1, \pi_0)$.
- If $\ell(v) = (2, \pi)$, then v has a successor v' with
  $\ell(v') = (1, \pi')$, where π' is **some** 2-variation of π.
- If $\ell(v) = (1, \pi)$, then v has successors $v_0, \ldots, v_{|\Gamma_1|}$ with
  $\ell(v_i) = (2, \pi_i)$, where the π_i are **all** 1-variations of π.
- If $\ell(v) = (i, \pi)$, then $\pi \not\models \varphi$.

Example

Consider again $\varphi = (p \leftrightarrow \neg q) \wedge (p' \leftrightarrow \neg q')$

with $\Gamma_1 = \{p, p'\}$ and $\Gamma_2 = \{q, q'\}$ and $\pi_0 = \begin{smallmatrix} p & q & p' & q' \\ 0 & 0 & 0 & 0 \end{smallmatrix}$.

...

EXPTIME games as a decision problem

Game$_1$ is the following decision problem:

Input: a game $(\varphi, \Gamma_1, \Gamma_2, \pi_0)$
Question: does Player 2 have a winning strategy?

Theorem (Stockmeyer & Chandra 1979)

Game$_1$ is **ExpTime-complete**.

EXPTIME-hardness

We want to show via a reduction from Game$_1$:

Theorem

Satisfiability of concepts w.r.t. $\mathcal{ALC}$-TBoxes is **ExpTime-hard**.

That is, given a game $G = (\varphi, \Gamma_1, \Gamma_2, \pi_0)$, we want to construct (in polytime) a concept C_G and TBox $\mathcal{T}_G$ such that:

Player 2 has a winning strategy in G **iff** C_G satisfiable w.r.t. $\mathcal{T}_G$

Main idea: (tree) models of C_G and $\mathcal{T}_G$ encode winning strategies!

- $\mathcal{T}_G$ restricts (tree) models such that they represent only winning strategies.
- C_G simply says that there is a root.

Details of the reduction

Let the variable partitioning be
$\Gamma_1 = \{p_1, \ldots, p_m\}$ and $\Gamma_2 = \{p_{m+1}, \ldots, p_n\}$.

We build C_G and $\mathcal{T}_G$ from the following concept and role names.

- role name r for the edges in the strategy tree
- concept name R for the root
- concept names $P_1, \ldots, P_n$ for the variables
- concept names S_1, S_2 for the active player
- concept names $V_1, \ldots, V_n$ for the variable whose value was changed to reach the current configuration

$\mathcal{T}_G$ consists of the following GCIs

1. The initial configuration is correctly represented.

$$R \sqsubseteq S_1 \sqcap \prod_{i \le n,\ \pi_0(p_i)=0} \neg P_i \sqcap \prod_{i \le n,\ \pi_0(p_i)=1} P_i$$

2. Whenever it is Player 1's turn, there are $m+1$ successors representing all her possible moves:

$$S_1 \sqsubseteq \exists r.(\neg V_1 \sqcap \cdots \sqcap \neg V_n) \sqcap \prod_{i \le m} \exists r.V_i$$

3. Whenever it is Player 2's turn, there is one successor:

$$S_2 \sqsubseteq \exists r.(\neg V_0 \sqcap \cdots \sqcap \neg V_n) \sqcup \bigsqcup_{m < i \le n} \exists r.V_i$$

4. In every move, at most one variable is changed:

$$\top \sqsubseteq \prod_{i < j \le n} \neg(V_i \sqcap V_j)$$

$\mathcal{T}_G$ consists of the following GCIs

5. The selected variable changes its truth value:

$$\top \sqsubseteq \prod_{i \le n} \Big((P_i \to \forall r.(V_i \to \neg P_i)) \sqcap (\neg P_i \to \forall r.(V_i \to P_i)) \Big)$$

6. All other variables keep their truth value:

$$\top \sqsubseteq \prod_{i \le n} \Big((P_i \to \forall r.(\neg V_i \to P_i)) \sqcap (\neg P_i \to \forall r.(\neg V_i \to \neg P_i)) \Big)$$

7. The players take turns:

$$S_1 \sqsubseteq \forall r.S_2, \quad S_2 \sqsubseteq \forall r.S_1, \quad S_1 \sqsubseteq \neg S_2$$

8. The formula φ never becomes true:

$$\top \sqsubseteq \neg\varphi$$

Additionally, set $C_G = R$.

Correctness of the reduction

Lemma

Player 2 has a winning strategy in G **iff** C_G satisfiable w.r.t. $\mathcal{T}_G$

To prove the lemma, we have to transform

"$\Rightarrow$" ... a winning strategy (V, E, ℓ) into a model $\mathcal{I}$ of C_G and $\mathcal{T}_G$:

- use V as the domain and interpret r as E;
- interpret r as the root;
- read off interpretation of the P_i, S_i, V_i from the labels $\ell(v)$.

It is then easy to check that $\mathcal{I}$ satisfies all GCIs 1–8.

"$\Leftarrow$" ... a **tree model** $\mathcal{I}$ of C_G & $\mathcal{T}_G$ into a winning strat. (V, E, ℓ):

- analogously read off the $\ell(v)$ from the interpretations of the P_i.

GCIs 1–8 ensure that the resulting tree is a winning strategy.

Correctness, proof of "$\Rightarrow$"

Player 2 has a winning strategy in G $\Rightarrow$ C_G satisfiable w.r.t. $\mathcal{T}_G$

- Construct model $\mathcal{I}$ from winning strat. (V, E, ℓ) with root d_0:

$$\Delta^{\mathcal{I}} = V \qquad r^{\mathcal{I}} = E \qquad R^{\mathcal{I}} = \{d_0\}$$

$$P_i^{\mathcal{I}} = \{v \in V \mid \ell(v) = (j, \pi) \text{ and } \pi(p_i) = 1\}$$

$$S_j^{\mathcal{I}} = \{v \in V \mid \ell(v) = (j, \pi)\}$$

$$V_i^{\mathcal{I}} = \{v \in V \setminus \{d_0\} \mid \ell(v) = (j, \pi) \text{ and } \ell(v') = (j, \pi') \text{ for the parent node } v' \text{ of } v \text{ and } \pi(p_i) \neq \pi'(p_i)\}$$

- $\mathcal{I} \models \mathcal{T}_G$: check each GCI 1–8 separately. **(Exercise!)**
- $C_G^{\mathcal{I}} \neq \emptyset$ because $R^{\mathcal{I}} = \{d_0\}$.

Correctness, proof of "$\Leftarrow$"

Player 2 has a winning strategy in G $\Leftarrow$ C_G satisfiable w.r.t. $\mathcal{T}_G$

Exercise. (use the hint on Slide 30)

Harvest

Theorem

Satisfiability of concepts w.r.t. $\mathcal{ALC}$-TBoxes is **ExpTime-complete**.

The lower bound follows from the previous lemma.

For the upper bound see Slide 13 (type elimination).

Let's do this again!

We've used **ExpTime** games successfully to prove a matching lower bound for satisfiability of $\mathcal{ALC}$ concepts w.r.t. TBoxes.

For satisfiability/subsumption **without** TBoxes,
we already have a **PSpace** upper bound (tableau procedure).

We'll now show a matching lower bound via a reduction from a very similar type of game.

And now . . .

1. EXPTIME-membership
2. EXPTIME-hardness
3. PSPACE-hardness (not covered in the lecture)
4. Undecidability and a NEXPTIME lower bound $\rightsquigarrow$ Uli

Lower bound

We've seen on Wednesday:
Concept satisfiability and subsumption in $\mathcal{ALC}$ is **PSpace-complete**.

Standard approach to showing **PSpace-hardness**:

> Reduction from the validity problem of quantified boolean formulas (QBF)

More convenient approach:

> Reduction from a related, but much more intuitive problem:
> a game-theoretic problem whose **PSpace-completeness** is known

PSPACE games

Two players $1, 2$ play on a given propositional formula φ.

Every propositional variable in φ belongs to exactly one player.

Both players own **the same number of variables.**

Each player's variables are **linearly ordered.**

Player 1 begins; the players take turns.

In every move the player **chooses** the truth value of **her next variable.**

Player 1 wins if φ is true **at the end;** otherwise Player 2 wins.

Example

Consider the same formula $\varphi = (p_1 \leftrightarrow \neg p_2) \wedge (p_3 \leftrightarrow \neg p_4)$ where P1 owns p_1, p_3 and P2 owns p_2, p_4.

A run of the game where Player 2 wins:

- P1 sets p_1 to 1 $\rightsquigarrow$ $\pi = 1???$
- P2 sets p_2 to 1 $\rightsquigarrow$ $\pi = 11??$

$\rightsquigarrow$ No matter how p_3, p_4 are set, P2 will win (since $(p_1 \leftrightarrow \neg p_2)$ is false).

Differences in comparison with EXPTIME games

- The game always ends; the number of moves is predetermined.
- The players have no freedom in choosing which variables to assign next.
- The players cannot pass.
- Each variable is assigned a truth value exactly once.
- No initial assignment is needed.

PSPACE games: basics

A **game** is now just a propositional formula φ with an **even number** of variables $p_1, \ldots, p_n$.

A **configuration** is just a word $\pi \in \{0, 1\}^*$.

Intuition:

- Variables with odd index belong to P1, with even index to P2.
- Configuration π is a partial assignment: i-th symbol of π is the truth value of p_i .

Winning strategies

We're interested in winning strategies of **Player 1:**

A **winning strategy** for Player 1 in the game φ
is a **finite** 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 ε (empty configuration).
- If $\ell(v) = \pi$ with $|\pi|$ even and $|\pi| < n$ (P1's move),
  then v has a successor v' with $\ell(v') = \pi 0$ or $\ell(v') = \pi 1$.
- If $\ell(v) = \pi$ with $|\pi|$ odd (P2's move),
  then v has successors v', v'' with $\ell(v') = \pi 0$ & $\ell(v'') = \pi 1$.
- If $\ell(v) = \pi$ with $|\pi| = n$, then $\pi \models \varphi$.

PSPACE games as a decision problem

Game$_2$ is the following decision problem:

Input: a game φ
Question: does Player 1 have a winning strategy?

Theorem (Stockmeyer & Chandra 1979)

Game$_2$ is **PSPACE-complete**.

PSPACE-hardness

We want to show via a reduction from Game$_2$:

Theorem

Satisfiability of $\mathcal{ALC}$ concepts is **PSPACE-hard**.

That is, given a game φ, we want to construct (in polytime) a concept C_φ such that:

Player 1 has a winning strategy in game φ **iff** C_φ satisfiable

Main idea, again: (tree) models of C_φ encode winning strategies!

Now C_φ restricts (tree) models such that they represent only winning strategies.

Details of the reduction

Let the variables in φ be $p_1, \ldots, p_n$, n even.

We build C_φ from the following concept and role names.

- role name r for the edges in the strategy tree
- concept names $P_1, \ldots, P_n$ for truth values of the variables in the partial assignments

We write $\forall r^i.C$ for $\underbrace{\forall r. \cdots \forall r.C}_{i \text{ times}}$

Details of the reduction

C_φ consists of the following conjuncts.

1. If $|\pi|$ is even (i.e., it is P1's move),
   then there is a successor that selects the truth value of P_{i+1}:

$$C_1 \;:=\; \prod_{i\in\{0,2,\ldots,n-2\}} \forall r^i.\Big(\exists r.\neg P_{i+1} \sqcup \exists r.P_{i+1}\Big)$$

2. If $|\pi|$ is odd (i.e., it is P2's move),
   then there are two successors for both truth values of P_{i+1}:

$$C_2 \;:=\; \prod_{i\in\{1,3,\ldots,n-1\}} \forall r^i.\Big(\exists r.\neg P_{i+1} \sqcap \exists r.P_{i+1}\Big)$$

Details of the reduction

C_φ consists of the following conjuncts.

1. The chosen truth values do not change:

$$C_3 \;:=\; \prod_{1\le i\le j<n} \forall r^j.\Big((P_i \to \forall r.P_i) \sqcap (\neg P_i \to \forall r.\neg P_i)\Big)$$

2. At the leaves φ is true:

$$C_4 \;:=\; \forall r^n.\varphi$$

Set $C_\varphi \;=\; C_1 \sqcap C_2 \sqcap C_3 \sqcap C_4$.

Correctness of the reduction

Lemma

Player 1 has a winning strategy in φ **iff** C_φ satisfiable.

Proof: very similar to the corresponding EXPTIME lemma.

As a consequence we get:

Theorem

Satisfiability of $\mathcal{ALC}$ concepts is PSPACE-complete.

Summary

Satisfiability of and subsumption for $\mathcal{ALC}$ concepts w.r.t. TBoxes is ExpTime-complete.

Satisfiability of and subsumption for $\mathcal{ALC}$ concepts without TBoxes is PSpace-complete.

The same holds for $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$.

Tree models play an important rôle in all cases.

PSpace versus ExpTime: polynomially deep versus infinite trees

Types are an important notion for developing worst-case optimal algorithms.

And now . . .

1. EXPTIME-membership
2. EXPTIME-hardness
3. PSPACE-hardness (not covered in the lecture)
4. Undecidability and a NEXPTIME lower bound $\leadsto$ Uli