Beschreibungslogik
Kapitel 5: Komplexität

Sommersemester 2017 Thomas Schneider

AG Theorie der künstlichen Intelligenz (TdKI)

http://tinyurl.com/ss17-bl

Vorlesungsübersicht

Ziel des Kapitels

Automatisches Schlussfolgern spielt zentrale Rolle für BLen:

- ermöglicht die Entwicklung intelligenter Anwendungen
- die Ausdrucksstärke von BLen ist stark darauf zugeschnitten

Wichtig für automatisches Schlussfolgern:

1. Entscheidbarkeit der relevanten Schlussfolgerungsprobleme
2. möglichst geringe Komplexität
3. Algorithmen, die sich in der Praxis performant verhalten

Dieses Kapitel: **Punkt 2**

Zur Erinnerung

2ExpTime	2ExpTime-vollständige Probleme sind **erwiesenermaßen** nicht in exp. Zeit lösbar.
$\cup$	
ExpTime	ExpTime-vollständige Probleme sind **erwiesenermaßen** nicht in Polyzeit lösbar.
$\cup$	
PSpace	NP-/PSpace-vollständige Probleme sind **wahrscheinlich** nicht in Polyzeit lösbar.
$\cup$	
NP	
$\cup$	
P	$\approx$ effizient lösbar

Obere Schranke: Enthaltensein in der jeweiligen Klasse

Untere Schranke: Härte für die jeweilige Klasse (mittels Polyzeit-Reduktionen)

Siehe auch Skript zur VL „Theoretische Informatik 1/2“, §18–20:
`http://tinyurl.com/ss16-theoinf`

Kapitel 5: Komplexität

Kapitel 5: Komplexität

Obere Schranke

Wir wollen zeigen:

Theorem 5.1

In $\mathcal{ALC}$ ist die Erfüllbarkeit von Konzepten bzgl. TBoxen ExpTime*-vollständig.*

$\rightsquigarrow$ Mit Lemma 2.9: Subsumtion und Äquivalenz ExpTime-vollständig

Wir beginnen mit oberer Schranke (Enthaltensein in ExpTime):

- Wir verwenden ein Verfahren aus der Modallogik: Typelimination [Pratt78]
- Basiert auf **syntaktischem** Typ-Begriff

Syntaktische Typen

Wir nehmen an, dass

- das Eingabe-Konzept C_0 in NNF ist und
- die Eingabe-TBox die Form $\{\top \sqsubseteq C_{\mathcal{T}}\}$ hat mit $C_{\mathcal{T}}$ in NNF.

Definition 5.2 (Typ)

Ein **Typ für C_0 und $\mathcal{T}$** ist eine Teilmenge $t \subseteq \mathsf{sub}(C_0, \mathcal{T})$, so dass

1. $A \in t$ gdw. $\neg A \notin t$ für alle $\neg A \in \mathsf{sub}(C_0, \mathcal{T})$
2. $C \sqcap D \in t$ gdw. $C \in t$ und $D \in t$ f. alle $C \sqcap D \in \mathsf{sub}(C_0, \mathcal{T})$
3. $C \sqcup D \in t$ gdw. $C \in t$ oder $D \in t$ f. alle $C \sqcup D \in \mathsf{sub}(C_0, \mathcal{T})$
4. $C_{\mathcal{T}} \in t$

T 5.1

Typelimination

Generelle Idee der Typelimination bei Eingabe C_0, $\mathcal{T}$:

- Generiere alle Typen für C_0 und $\mathcal{T}$ (exponentiell viele).
- Eliminiere wiederholt Typen, die in keinem Modell von $\mathcal{T}$ vorkommen können.
- Überprüfe, ob ein Typ überlebt hat, der C_0 enthält.
- Wenn ja, antworte „erfüllbar“, sonst „unerfüllbar“.

Schlechte Typen

Präzisierung von „Typen, die in keinem Modell vorkommen können“:

Definition 5.3 (schlechter Typ)

Sei Γ Typenmenge und $t \in \Gamma$.

Dann ist t **schlecht in Γ**, wenn für ein $\exists r.C \in t$ gilt:

$$\text{Es gibt kein } t' \in \Gamma \text{ mit } \{C\} \cup \{D \mid \forall r.D \in t\} \subseteq t'.$$

Intuitiv:

Typ ist schlecht, wenn er eine existentielle Restriktion enthält, für die es keinen „Zeugen“ gibt.

T 5.1 Forts.

Typelimination

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

input : $\mathcal{ALC}$-Konzept C_0, $\mathcal{ALC}$-TBox $\mathcal{T}$
output : „erfüllbar“ / „unerfüllbar“

Berechne Γ_0: Menge aller Typen für C_0 und $\mathcal{T}$

$i := 0$
repeat
  $i := i + 1$
  $\Gamma_i := \{t \in \Gamma_{i-1} \mid t \text{ nicht schlecht in } \Gamma_{i-1}\}$
until $\Gamma_i = \Gamma_{i-1}$

if *es gibt* $t \in \Gamma_i$ *mit* $C_0 \in t$ **then return** „erfüllbar“
else return „unerfüllbar“

T 5.1 Forts.

Analyse des Algorithmus

Wir müssen nun zeigen, dass der Algorithmus . . .

1. **in exponentieller Zeit terminiert,**
2. **korrekt** ist,
3. **vollständig** ist.

Wir beginnen wieder mit Terminierung (+ Komplexitätsanalyse).

Terminierung

Lemma 5.4

$\mathcal{ALC}$-$\texttt{Elim}(C_0, \mathcal{T})$ terminiert nach $2^{\text{poly}(|C_0|+|\mathcal{T}|)}$ Schritten.

Beweis. Sei $n = |C_0| + |\mathcal{T}|$. Dann gilt:

- Es gibt nur 2^n viele Typen (Lem. 3.13: $|\text{sub}(C, \mathcal{T})| \leq |C| + |\mathcal{T}|$)
- In jedem Schritt der **repeat**-Schleife wird mindestens ein Typ eliminiert $\rightsquigarrow$ Schleife terminiert nach max. 2^n Durchläufen.
- Die übrigen Operationen (z. B. Prüfen, ob ein Typ schlecht ist) können leicht in Zeit $2^{\text{poly}(n)}$ implementiert werden.

□

Korrektheit

Lemma 5.5

Wenn $\mathcal{ALC}$-Elim$(C_0, \mathcal{T})$ „erfüllbar“ zurückgibt, dann ist C_0 bezüglich $\mathcal{T}$ erfüllbar.

Beweis. Wenn der Algorithmus „erfüllbar“ zurückgibt, dann gibt es in der resultierenden Typmenge Γ_i einen Typen $t_0 \in \Gamma_i$ mit $C_0 \in t_0$.

Aus Γ_i konstruieren wir Interpretation $\mathcal{I}$ wie folgt.

$$\Delta^{\mathcal{I}} = \Gamma_i$$

$$A^{\mathcal{I}} = \{t \mid A \in t\} \quad \text{für alle Konzeptnamen } A$$

$$r^{\mathcal{I}} = \{(t, t') \mid \forall r.C \in t \text{ impliziert } C \in t'\} \quad \text{f. a. Rollennamen } r$$

Die Erfüllbarkeit von C_0 bzgl. $\mathcal{T}$ folgt dann aus: T 5.2

Behauptung. Für alle $C \in \text{sub}(C_0, \mathcal{T})$ und alle $t \in \Gamma_i$ gilt:

$$C \in t \quad \text{impliziert} \quad t \in C^{\mathcal{I}}$$ T 5.2 Forts.

Vollständigkeit

Lemma 5.6

Wenn C_0 bzgl. $\mathcal{T}$ erfüllbar ist,
dann gibt $\mathcal{ALC}$-$\texttt{Elim}(C_0, \mathcal{T})$ „erfüllbar“ zurück.

Beweis. Sei C_0 erfüllbar bzgl. $\mathcal{T}$
und $\mathcal{I}$ ein Modell von C_0 und $\mathcal{T}$ mit $d_0 \in C_0^{\mathcal{I}}$. Sei

$$\Gamma = \{t_{\mathcal{I}}(d) \mid d \in \Delta^{\mathcal{I}}\}$$

und $\Gamma_0, \Gamma_1, \ldots, \Gamma_k$ die von $\mathcal{ALC}$-$\texttt{Elim}(C_0, \mathcal{T})$ erzeugte Sequenz.

Wir zeigen:

Behauptung. $\Gamma \subseteq \Gamma_i$ für alle $i \leq k$ **T 5.3**

Wegen $d_0 \in C_0^{\mathcal{I}}$ ist also $C_0 \in t_{\mathcal{I}}(d_0) \in \Gamma \subseteq \Gamma_k$.

Also gibt der Algorithmus „erfüllbar“ zurück. □

Komplexität

Aus Lemmas 5.4–5.6
(Terminierung in exp. Zeit, Korrektheit, Vollständigkeit)

folgt nun:

Theorem 5.7

In $\mathcal{ALC}$ ist die Erfüllbarkeit von Konzepten bzgl. TBoxen entscheidbar in ExpTime.

Typelimination vs. Tableau-Algorithmen

Offensichtliche Entsprechungen:

- $\sqcap$-Regel, $\sqcup$-Regel, TBox-Regel finden sich wieder in der Definition eines Typs.
- $\exists$-Regel und $\forall$-Regel finden sich wieder in Def. von „schlecht“.
- Freiheit von offensichtlichen Widersprüchen findet sich wieder in der Definition eines Typs.

Unterschiede:

- Tableau-Algorithmus benötigt im Worst Case dreifach exponentielle Laufzeit.
- Typelimination benötigt im Best Case exponentielle Laufzeit.

Kapitel 5: Komplexität

Untere Schranke

Standard-Ansatz zum Beweis von ExpTime-Härte:
Reduktion des Wortproblems für polynomiell platzbeschränkte alternierende Turingmaschinen.

Unser Ansatz:
Wir reduzieren stattdessen ein **spieltheoretisches Problem** (das ist intuitiver).

ExpTime-Spiele

Zwei Spielerinnen spielen auf gegebener aussagenlogischer Formel φ.

Jede Variable in φ gehört entweder Spielerin 1 oder Spielerin 2.

Das Spiel beginnt auf einer gegebenen Anfangsbelegung π_0 der Variablen in φ.

Spielerin 1 beginnt; die Spielerinnen wechseln sich ab.

In jedem Zug ändert Spielerin Wahrheitswert einer ihrer Variablen; es ist erlaubt, zu passen.

Spielerin 1 gewinnt, wenn φ jemals wahr wird (egal, welche Spielerin gezogen hat).

Spielerin 2 gewinnt, wenn das Spiel unendlich weitergeht, ohne dass φ wahr wird. **T 5.4**

ExpTime-Spiele

Definition 5.8

Ein **Spiel** ist ein Tupel $(\varphi, \Gamma_1, \Gamma_2, \pi_0)$
mit Γ_1, Γ_2 Partitionierung der Variablen in φ, π_0 Anfangsbelegung.

Eine **Konfiguration** ist ein Paar (i, π)
mit $i \in \{1, 2\}$ aktive Spielerin und π Belegung.

Belegung π ist ***j*-Variation** von π' ($j \in \{1, 2\}$), wenn $\pi = \pi'$
oder π und π' unterscheiden sich nur in einer Variablen $p \in \Gamma_j$.

Bedeutung von „π ist j-Variation von π'“:
Spielerin j kann π in π' transformieren (oder umgekehrt).

Das hier relevante Entscheidungsproblem bezieht sich auf
Gewinn**strategien** für Spielerin **2.**

Gewinnstrategien

Intuitiv:

- Eine Gewinnstrategie sagt Spielerin 2 **nach jedem möglichen Spielverlauf,** wie sie spielen muss um zu gewinnen.
- Wenn Spielerin 2 eine Gewinnstrategie hat,
  kann sie das Spiel **immer** gewinnen – egal, was Spielerin 1 tut.

Gewinnstrategien

Definition 5.9 (Gewinnstrategie)

Eine **Gewinnstrategie** für Spielerin 2 im Spiel $(\varphi, \Gamma_1, \Gamma_2, \pi_0)$ ist ein unendlicher knotenbeschrifteter Baum (V, E, ℓ), wobei ℓ jedem Knoten $v \in V$ Konfiguration $\ell(v)$ zuweist, so dass:

(a) Die Wurzel ist beschriftet mit $(1, \pi_0)$.

(b) Wenn $\ell(v) = (2, \pi)$, dann hat v Nachfolger v' mit $\ell(v') = (1, \pi')$ und π' 2-Variation von π.

(c) Wenn $\ell(v) = (1, \pi)$, dann hat v Nachfolger $v_0, \ldots, v_{|\Gamma_1|}$ mit $\ell(v_i) = (2, \pi_i)$, wobei $\pi_0, \ldots, \pi_{|\Gamma_1|}$ **alle** existierenden 1-Variationen von π sind.

(d) Wenn $\ell(v) = (i, \pi)$, dann $\pi \not\models \varphi$.

T 5.5

ExpTime-Spiele als Entscheidungsproblem

Definition 5.10

Spiel$_1$ ist das folgende Problem:

Gegeben Spiel $(\varphi, \Gamma_1, \Gamma_2, \pi_0)$,
entscheide, ob Spielerin 2 eine Gewinnstrategie hat.

Theorem 5.11 (Stockmeyer, Chandra 1979)

Spiel$_1$ *ist* ExpTime-*vollständig.*

ExpTime-Härte von Erfüllbarkeit in $\mathcal{ALC}$ bzgl. TBoxen:
Beweis **per Reduktion von Spiel$_1$**

ExpTime-Härte, Überblick

Ziel:
Gegeben Spiel $S = (\varphi, \Gamma_1, \Gamma_2, \pi_0)$, konstruiere (in Polynomialzeit) Konzept C_S und TBox $\mathcal{T}_S$, so dass:

Spielerin 2 hat Gewinnstrategie in S **gdw.** C_S erfüllbar bzgl. $\mathcal{T}_S$

Idee:
(Baum)-Modelle von C_S und $\mathcal{T}_S$ kodieren Gewinnstrategien.

Details der Reduktion

Sei $\Gamma_1 = \{p_0, \ldots, p_{k-1}\}$ und $\Gamma_2 = \{p_k, \ldots, p_{n-1}\}$.

Signatur von C_S und $\mathcal{T}_S$:

- Rollenname r für Kanten im Baum
- Konzeptname W für die Wurzel
- Konzeptnamen $P_0, \ldots, P_{n-1}$ für die Variablen
- Konzeptnamen S_1, S_2 für die aktive Spielerin
- Konzeptnamen $V_0, \ldots, V_{n-1}$ für die Variable, deren Wert zum Erreichen der aktuellen Konfiguration geändert wurde

Details der Reduktion

(1) Die Anfangskonfiguration ist korrekt:

$$W \sqsubseteq S_1 \sqcap \bigsqcap_{\substack{i<n \\ \pi_0(p_i)=0}} \neg P_i \sqcap \bigsqcap_{\substack{i<n \\ \pi_0(p_i)=1}} P_i$$

(2) Wenn Spielerin 1 am Zug ist, gibt es $k+1$ Nachfolger:

$$S_1 \sqsubseteq \exists r.(\neg V_0 \sqcap \cdots \sqcap \neg V_{n-1}) \sqcap \bigsqcap_{i<k} \exists r.V_i$$

(3) Wenn Spielerin 2 am Zug ist, gibt es **einen** Nachfolger:

$$S_2 \sqsubseteq \exists r.(\neg V_0 \sqcap \cdots \sqcap \neg V_{n-1}) \sqcup \bigsqcup_{k\leq i<n} \exists r.V_i$$

(4) Es ändert sich höchstens eine Variable pro Zug:

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

Details der Reduktion

(5) Die ausgewählte Variable ändert ihren Wahrheitswert:[1]

$$\top \sqsubseteq \bigsqcap_{i<n} \Big(\big(P_i \to \forall r.(V_i \to \neg P_i) \big) \sqcap \big(\neg P_i \to \forall r.(V_i \to P_i) \big) \Big)$$

(6) Alle anderen Variablen behalten ihren Wert:[1]

$$\top \sqsubseteq \bigsqcap_{i<n} \Big(\big(P_i \to \forall r.(\neg V_i \to P_i) \big) \sqcap \big(\neg P_i \to \forall r.(\neg V_i \to \neg P_i) \big) \Big)$$

(7) Die Spielerinnen wechseln sich ab:

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

(8) Die Formel φ ist immer falsch: $\top \sqsubseteq \neg\varphi$

Setzen außerdem $\boldsymbol{C_S = W}$.

[1] $C \to D$ ist Abkürzung für $\neg C \sqcup D$.

Korrektheit der Reduktion

Lemma 5.12

Spielerin 2 hat Gewinnstrategie in S **gdw.** *C_S erfüllbar bzgl. $\mathcal{T}_S$.*

Beweis. „⇒“ Sei (V, E, ℓ) eine Gewinnstrategie für Spielerin 2 mit Wurzel $w \in V$. Wir konstruieren Interpretation $\mathcal{I}$ wie folgt.

$$\Delta^{\mathcal{I}} = V \qquad r^{\mathcal{I}} = E \qquad W^{\mathcal{I}} = \{w\}$$

$$P_i^{\mathcal{I}} = \{v \in V \mid \ell(v) = (t, \pi) \text{ mit } \pi(p_i) = 1\} \qquad \text{für alle } i < n$$

$$S_i^{\mathcal{I}} = \{v \in V \mid \ell(v) = (i, \pi)\} \qquad \text{für alle } i \in \{1, 2\}$$

$$V_i^{\mathcal{I}} = \{v \in V \mid (v', v) \in E \text{ mit } \ell(v') = (t', \pi'), \ \ell(v) = (t, \pi), \ \pi(p_i) \neq \pi'(p_i) \ \} \text{ f. alle } i < n$$

Die Erfüllbarkeit von C_S bzgl. $\mathcal{T}_S$ folgt dann aus:

Behauptung. $\mathcal{I}$ ist ein Modell von W und $\mathcal{T}_S$. **T 5.6**

□

Korrektheit der Reduktion

Lemma 5.12

Spielerin 2 hat Gewinnstrategie in S **gdw.** *C_S erfüllbar bzgl. $\mathcal{T}_S$.*

Beweis. „$\Leftarrow$“ Sei $\mathcal{I}$ ein Modell von W und $\mathcal{T}_S$.
O. B. d. A. ist $\mathcal{I}$ ein **Baummodell** (Thm. 3.6). Setze

$$V = \Delta^{\mathcal{I}} \qquad E = r^{\mathcal{I}} \qquad \ell(v) = (i, \pi) \text{ für alle } v \in V,$$

$$\text{wobei } i = \begin{cases} 1 & \text{wenn } v \in S_1^{\mathcal{I}} \\ 2 & \text{wenn } v \in S_2^{\mathcal{I}} \end{cases} \quad \text{und} \quad \pi(p_i) = \begin{cases} 1 & \text{wenn } v \in P_i^{\mathcal{I}} \\ 0 & \text{wenn } v \notin P_i^{\mathcal{I}} \end{cases}$$

Wegen Konzeptinklusionen **(1)** und **(7)** ist $S_1^{\mathcal{I}}, S_2^{\mathcal{I}}$ eine **Partitionierung** von $\Delta^{\mathcal{I}}$; also ist ℓ **wohldefiniert.**

Noch zu zeigen:

Behauptung. (V, E, ℓ) ist eine Gewinnstrategie für Spielerin 2. **T 5.7**

„Ernte“

Lemma 5.12

Spielerin 2 hat Gewinnstrategie in S **gdw.** *C_S erfüllbar bzgl. $\mathcal{T}_S$.*

Daraus folgt:

Theorem 5.13

In $\mathcal{ALC}$ ist die Erfüllbarkeit von Konzepten bzgl. TBoxen ExpTime*-hart.*

Daraus und aus Thm. 5.7 (obere Schranke) folgt:

Theorem 5.1

In $\mathcal{ALC}$ ist die Erfüllbarkeit von Konzepten bzgl. TBoxen ExpTime*-vollständig.*

Kapitel 5: Komplexität

Obere Schranke

Wir wollen zeigen:

Theorem 5.14

In $\mathcal{ALC}$ ist die Erfüllbarkeit von Konzepten (ohne TBoxen) PSPACE*-vollständig.*

Mit Lemma 2.9 sind dann auch Subsumtion und Äquivalenz PSPACE-vollständig.

Wir beginnen mit **oberer Schranke** (Enthaltensein in PSPACE), benutzen ein Verfahren aus der Modallogik: **K-Worlds**

Baummodelle

Zur Erinnerung:
Wenn Konzept C erfüllbar ist, dann hat C ein Baummodell (Theorem 3.6).

Mit TBox $\mathcal{T}$ kann es sein, dass **alle Baummodelle unendlich** sind:

$$\text{z. B.} \quad A \text{ erfüllbar bzgl. } \mathcal{T} = \{A \sqsubseteq \exists r.A\}$$

Ohne TBox gibt es stets ein Baummodell, dessen **Tiefe durch $|C|$ beschränkt** ist.

Es genügt, die Existenz solcher Modelle zu überprüfen.

$\mathcal{ALC}$-Worlds: Grundidee

Gesucht: Algorithmus, der in PSPACE läuft – **aber:**

- Ein linear tiefer Baum ist exponentiell groß.
- Wenn wir gesamtes Modell im Speicher hielten, dann würde das **exponentiellen Platz** erfordern.
- **Stattdessen:** prüfe Existenz des Baumes mittels Tiefensuche; halte **zu jeder Zeit nur einen Pfad** des Baumes im Speicher

Wir entwickeln nichtdeterministischen Algorithmus, verwenden:

Theorem 5.15 (Savitch 1970)

PSPACE = NPSPACE

Vorbereitungen

Wir nehmen an, dass Eingabe C_0 in **NNF** ist.

Wir verwenden wieder **Typen,** definieren diese jedoch **differenzierter.**

Zur Erinnerung:

Die **Rollentiefe rd(C)** von Konzepten $C \in \mathrm{sub}(C_0)$ ist induktiv wie folgt definiert.

$$\begin{aligned} \mathrm{rd}(A) &= \mathrm{rd}(\neg A) &&= 0 \\ \mathrm{rd}(C \sqcap D) &= \mathrm{rd}(C \sqcup D) &&= \max(\mathrm{rd}(C), \mathrm{rd}(D)) \\ \mathrm{rd}(\exists r.C) &= \mathrm{rd}(\forall r.C) &&= 1 + \mathrm{rd}(C) \end{aligned}$$

Lemma 4.4

Für alle $C \in \mathrm{sub}(C_0)$ gilt: $\mathrm{rd}(C) \leq |C|$

i-Typen

Definition 5.16 (i-Konzepte)

Für $i \geq 0$ ist die Menge der **i-Konzepte** definiert als:

$$\mathrm{sub}_i(C_0) = \{C \in \mathrm{sub}(C_0) \mid \mathbf{rd}(C) \leq i\}$$

Definition 5.17 (i-Typ)

Sei $i \geq 0$.
Ein **Typ für C_0** ist eine Teilmenge $t \subseteq \mathrm{sub}_i(C_0)$, so dass

1. $A \in t$ gdw. $\neg A \notin t$ für alle $\neg A \in \mathrm{sub}_i(C_0)$
2. $C \sqcap D \in t$ gdw. $C \in t$ und $D \in t$ für alle $C \sqcap D \in \mathrm{sub}_i(C_0)$
3. $C \sqcup D \in t$ gdw. $C \in t$ oder $D \in t$ für alle $C \sqcup D \in \mathrm{sub}_i(C_0)$

T 5.8

$\mathcal{ALC}$-Worlds

Procedure $\mathcal{ALC}$-`Worlds`(C_0):

- $i \leftarrow \mathrm{rd}(C_0)$
- Rate $t \subseteq \mathrm{sub}_i(C_0)$ mit $C_0 \in t$
- `recurse` (t, i, C_0)

Procedure `recurse`(t, i, C_0):

- **if** *t ist kein i-Typ für C_0* **then return** `false`
- **for all** $\exists r.C \in t$ **do**
  - $S := \{C\} \cup \{D \mid \forall r.D \in t\}$
  - Rate $t' \subseteq \mathrm{sub}_{i-1}(C_0)$ mit $S \subseteq t'$
  - **if** *recurse*$(t', i-1, C_0) =$ `false` **then return** `false`
- **return** `true`

T 5.8 Forts.

Analyse des Algorithmus

Wir müssen zeigen, dass der Algorithmus . . .

1. **terminiert** und nur **polynomiellen Platz** benötigt,
2. **korrekt** ist,
3. **vollständig** ist.

Wir beginnen wieder mit Terminierung + Komplexitätsanalyse.

Terminierung

Lemma 5.18

$\mathcal{ALC}$-Worlds(C_0) terminiert & benötigt polynomiellen Platz (in $|C_0|$).

Beweis.

Jeder Lauf von $\mathcal{ALC}$-Worlds(C_0) kann als **Rekursionsbaum** $B = (V, E, \ell)$ dargestellt werden:

- **Wurzel:** initialer Aufruf von `recurse`
  übrige Knoten: rekursive Aufrufe
- $(v, v') \in E$ wenn Aufruf v' während Aufruf v stattfindet
- $\ell(v) = (p_1(v), p_2(v), p_3(v))$ sind Parameter bei Aufruf v

Terminierung

Nun ist leicht zu sehen:

- **Verzweigungsgrad** von B ist beschränkt durch Anzahl Subkonzepte $\exists r.C$ von C_0, also durch $|C_0|$.
- **Tiefe** von B ist beschränkt durch $\mathrm{rd}(C_0)$, also durch $|C_0|$.

Also terminiert $\mathcal{ALC}$-`Worlds`(C_0) und

- Rekursionsstapel hat Tiefe $\leq |C_0|$ und
- Speicherbedarf pro Aufruf polynomiell in $|C_0|$.

Also wird nur polynomiell viel Platz benötigt. □

Korrektheit

Lemma 5.19

Wenn $\mathcal{ALC}$-Worlds$(C_0) = \texttt{true}$, dann ist C_0 erfüllbar.

Beweis. Sei $\mathcal{ALC}$-Worlds$(C_0) = \texttt{true}$ und $\boldsymbol{T = (V, E, \ell)}$ der Rekursionsbaum eines erfolgreichen Laufes, mit Wurzel $\boldsymbol{v_0}$.

Für jeden Knoten $v \in V \setminus \{v_0\}$ sei $\boldsymbol{\sigma(v)}$ der Rollenname r des Konzeptes $\exists r.C$, für das der Aufruf v gemacht wurde.

Aus B konstruieren wir Interpretation $\mathcal{I}$ wie folgt.

$$\Delta^{\mathcal{I}} = V$$
$$r^{\mathcal{I}} = \{(v, v') \in E \mid \sigma(v') = r\} \quad \text{für alle Rollennamen } r$$
$$A^{\mathcal{I}} = \{v \mid A \in p_1(v)\} \quad \text{f. a. Konzeptn. } A \quad p_1(v)\text{: 1. Param. in } \ell(v)$$

Beh. Für alle $C \in \mathrm{sub}(C_0)$ und $v \in V$: $C \in p_1(v) \Rightarrow v \in C^{\mathcal{I}}$ **Übg.**

Da $C_0 \in p_1(v_0)$, ist auch $v_0 \in C_0^{\mathcal{I}} \Rightarrow \mathcal{I}$ ist Modell von C_0. □

Vollständigkeit

Lemma 5.20

Wenn C_0 erfüllbar ist, dann gibt es einen Lauf von $\mathcal{ALC}$-`Worlds`(C_0), der `true` *zurückgibt.*

Beweis. Sei C_0 erfüllbar und $\mathcal{I}$ ein Modell von C_0 mit $d_0 \in C_0^{\mathcal{I}}$.

Für jedes $d \in \Delta^{\mathcal{I}}$ und $i \geq 0$ definiere:

$$t_i(d) = \{C \in \mathsf{sub}_i(C_0) \mid d \in C^{\mathcal{I}}\}$$

Idee: Verwenden $\mathcal{I}$, um die nichtdeterministischen Entscheidungen von $\mathcal{ALC}$-`Worlds`(C_0) zu einem erfolgreichen Lauf zu „lenken“.

Zu diesem Zweck übergeben wir ein Element $d \in \Delta^{\mathcal{I}}$ als virtuelles **viertes Argument p_4** an `recurse`, so dass für alle $v \in V$:

$$C \in p_1(v) \quad \text{impliziert} \quad p_4(v) \in C^{\mathcal{I}} \qquad (*)$$

T 5.9 □

Komplexität

Aus Lemmas 5.18–5.20
(Terminierung in Polyplatz, Korrektheit, Vollständigkeit)

folgt nun:

Theorem 5.21

In $\mathcal{ALC}$ ist die Erfüllbarkeit von Konzepten (ohne TBoxen) entscheidbar in PSPACE.

$\mathcal{ALC}$-Worlds vs. Tableau-Algorithmen

Offensichtliche Entsprechungen:

- $\sqcap$-Regel, $\sqcup$-Regel finden sich wieder in der Definition i-Typ.
- $\exists$-Regel und $\forall$-Regel finden sich wieder im rekursiven Aufruf.
- Freiheit von offensichtlichen Widersprüchen findet sich wieder in der Definition i-Typ.
- Korrektheitsbeweise sind recht ähnlich.

Unterschiede:

- Tableau-Algorithmus ist deterministisch;
  hat dafür „teure“ $\sqcup$-Regel.
- Tableau-Algorithmus ist nicht platzoptimiert.

Erweiterungen von $\mathcal{ALC}$

$\mathcal{ALC}$-Worlds kann auf $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$ erweitert werden.

Auch in diesen Logiken ist Erfüllbarkeit ohne TBoxen also in PSPACE.

Kapitel 5: Komplexität

Untere Schranke

Standard-Ansatz zum Beweis von PSPACE-Härte:
Reduktion des Gültigkeitsproblems für QBFs
(quantifizierte Boolesche Formeln)

Unser Ansatz:
Wir reduzieren stattdessen wieder ein **spieltheoretisches Problem**
(mit obigem verwandt, aber intuitiver).

PSPACE-Spiele

Zwei Spielerinnen spielen auf gegebener aussagenlogischer Formel φ.

Jede Variable in φ gehört entweder Spielerin 1 oder Spielerin 2.
Jeder Spielerin gehören **gleich viele Variablen.**
Die Variablen der Spielerinnen sind **linear geordnet.**

Spielerin 1 beginnt; die Spielerinnen wechseln sich ab.

In jedem Zug wählt Spielerin Wahrheitswert ihrer **nächsten** Variable.

Spielerin 1 gewinnt, wenn φ am Ende wahr ist;
sonst gewinnt Spielerin 2. **T 5.10**

PSPACE-Spiele

Unterschiede zu EXPTIME-Spielen:

- Das Spiel **endet immer;** die Anzahl der Schritte ist **vorbestimmt.**
- Die Spielerin hat **keine Freiheit** in der **Wahl** ihrer Variablen.
- Jede Variable bekommt nur **einmal** einen Wahrheitswert zugewiesen.
- Man darf **nicht passen.**
- Es wird **keine Anfangsbelegung** benötigt.

PSPACE-Spiele

Definition 5.22

Ein **Spiel** ist eine aussagenlogische Formel φ mit Variablen $p_1, \ldots, p_n$, n geradzahlig.

Eine **Konfiguration** ist ein Wort $w \in \{0,1\}^*$.

Intuition:

- Variablen p_i mit i ungerade gehören Sp. 1, die anderen Sp. 2.
- Konfiguration w ist partielle Belegung:
  i-tes Symbol in w ist Wahrheitswert von p_i .

Das hier relevante Entscheidungsproblem bezieht sich auf Gewinn**strategien** für **Spielerin 1.**

Gewinnstrategien

Definition 5.23 (Gewinnstrategie)

Eine **Gewinnstrategie** für Spielerin 1 im Spiel φ
ist **endlicher** knotenbeschrifteter Baum (V, E, ℓ),
wobei ℓ jedem Knoten $v \in V$ Konfiguration $\ell(v)$ zuweist, so dass:

(a) Die Wurzel ist beschriftet mit ε (leere Konfiguration).

(b) Wenn $\ell(v) = w$ mit $|w|$ gerade und $|w| < n$, (d.h. Sp. 1 am Zug)
dann hat v Nachfolger v' mit $\ell(v') \in \{w0, w1\}$.

(c) Wenn $\ell(v) = w$ mit $|w|$ ungerade, (d. h. Sp. 2 am Zug)
dann hat v Nachfolger v', v'' mit $\ell(v') = w0$ und $\ell(v'') = w1$.

(d) Wenn $\ell(v) = w$ mit $|w| = n$, dann $w \models \varphi$.

T 5.11

ExpTime-Spiele als Entscheidungsproblem

Definition 5.24

Spiel$_2$ ist das folgende Problem:

Gegeben Spiel φ,
entscheide, ob Spielerin 1 eine Gewinnstrategie hat.

Theorem 5.25 (Schaefer 1978)

Spiel$_2$ *ist* PSpace*-vollständig.*

PSpace-Härte von Erfüllbarkeit in $\mathcal{ALC}$ ohne TBoxen:
Beweis **per Reduktion von Spiel$_2$**

PSPACE-Härte, Überblick

Ziel:
Gegeben Spiel φ, konstruiere (in Polynomialzeit) Konzept C_φ, so dass:

Spielerin 1 hat Gewinnstrategie in φ **gdw.** C_φ erfüllbar

Idee:
(Baum)-Modelle von C_φ kodieren Gewinnstrategien.

Details der Reduktion

Die Variablen in φ seien $p_1, \ldots, p_n$, n geradzahlig.

Signatur von C_φ:

- Rollenname r für Kanten im Baum
- Konzeptnamen $P_1, \ldots, P_n$ für die Wahrheitswerte der Variablen in partiellen Belegungen

Wir schreiben wieder $\forall r^i.C$ für $\underbrace{\forall r. \cdots \forall r.}_{i\text{-mal}} C$.

C_φ ist eine Konjunktion mit folgenden Konjunkten.

Details der Reduktion

(1) $|w|$ gerade **gdw.** Sp. 1 am Zug **gdw.** Knoten auf Tiefe i, $2 \mid i$.
Dann gibt es einen Nachfolger, der Wert für P_{i+1} auswählt.

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

(2) $|w|$ ungerade **gdw.** Sp. 2 am Zug **gdw.** Knoten auf Tiefe i, $2 \nmid i$.
Dann gibt es zwei Nachfolger für beide Werte von P_{i+1}.

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

Details der Reduktion

(3) Einmal gewählte Wahrheitswerte bleiben erhalten:

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

(4) An den Blättern ist φ wahr:

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

Korrektheit der Reduktion

Setze $C_\varphi = C_1 \sqcap C_2 \sqcap C_3 \sqcap C_4$.

Lemma 5.26

Spielerin 1 hat Gewinnstrategie in φ **gdw.** *C_φ erfüllbar.*

Daraus folgt:

Theorem 5.27

In $\mathcal{ALC}$ ist die Erfüllbarkeit von Konzepten (ohne TBoxen) PSPACE*-hart.*

Daraus und aus Thm. 5.21 (obere Schranke) folgt:

Theorem 5.14

In $\mathcal{ALC}$ ist die Erfüllbarkeit von Konzepten (ohne TBoxen) PSPACE*-vollständig.*

Zusammenfassung

- Erfüllbarkeit in $\mathcal{ALC}$ **mit** TBoxen ist **ExpTime**-vollständig.
- Erfüllbarkeit in $\mathcal{ALC}$ **ohne** TBoxen ist **PSpace**-vollständig.
- Dasselbe gilt für $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$.
- **Baummodelle** spielen in allen Fällen eine wichtige Rolle.
- PSpace vs. ExpTime: polynomiell tiefe vs. unendliche Bäume
- **Typen** sind wichtiger Begriff zum Entwickeln von Algorithmen.

Gründe für Entscheidbarkeit?

Es gab eine Zeitlang Diskussionen darüber, was die beste Erklärung für die Entscheidbarkeit von Modal- und Beschreibungslogiken ist.

- Existenz von **Baummodellen**
- Einbettbarkeit in das **2-Variablen-Fragment** der Prädikatenlogik
- Einbettbarkeit in das **Guarded Fragment** der Prädikatenlogik

Siehe z. B.:

Erich Grädel: *Why are Modal Logics So Robustly Decidable?*
(Literaturverzeichnis am Ende der Folien.)

Kapitel 5: Komplexität

Konkrete Bereiche (concrete domains)

Einige Erweiterungen von $\mathcal{ALC}$, die zunächst vielleicht harmlos erscheinen, können zu **Unentscheidbarkeit** führen.

Wir betrachten hier beispielhaft **konkrete Bereiche**, die es erlauben, Zahlen, Strings und andere Datentypen zu verwenden.

Definition 5.28 (Konkreter Bereich)

Ein **konkreter Bereich** ist ein Paar $\mathcal{B} = (\Delta^{\mathcal{B}}, \Phi^{\mathcal{B}})$, wobei

- $\Delta^{\mathcal{B}}$ eine Menge von **Werten** ist und
- $\Phi^{\mathcal{B}}$ eine Menge von **Prädikaten**,

so dass jedes $P \in \Phi^{\mathcal{B}}$ mit einer Stelligkeit $n \geq 0$ ausgestattet ist und mit einer Extension $P^{\mathcal{B}} \subseteq (\Delta^{\mathcal{B}})^n$.

T 5.12

$\mathcal{ALC}(\mathcal{B})$

Definition 5.29 ($\mathcal{ALC}(\mathcal{B})$ Syntax)

Sei $\mathcal{B}$ ein konkreter Bereich.

$\boldsymbol{\mathcal{ALC}(\mathcal{B})}$ ist die Erweiterung von $\mathcal{ALC}$ um $\mathcal{B}$, d. h. um

- **Featurenamen** (eine zusätzliche Art von Rolle) und
- die Konstruktoren $\exists R_1, \ldots, R_n.P$ und $\forall R_1, \ldots, R_n.P$,

wobei $P \in \Phi^{\mathcal{B}}$ n-stellig ist und die R_i **Rollenkompositionen** der Form

$$r_1; r_2; \cdots ; r_k; f$$

sind mit $r_1, \ldots, r_k$ Rollennamen und f Featurename.

T 5.13

$\mathcal{ALC}(\mathcal{B})$

Definition 5.30 ($\mathcal{ALC}(\mathcal{B})$ Semantik)

Eine Interpretation $\mathcal{I}$ ordnet nun zusätzlich jedem Featurenamen f eine **Funktion** $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{B}}$ zu. Für jede Rollenkomposition

$$R = r_1; r_2; \cdots ; r_k; f$$

bezeichnet $R^{\mathcal{I}}$ die Komposition der Interpretationen:

$$R^{\mathcal{I}} = r_1^{\mathcal{I}} \circ r_2^{\mathcal{I}} \circ \cdots \circ r_k^{\mathcal{I}} \circ f^{\mathcal{I}}$$

Die Semantik der zusätzlichen Konstruktoren ist nun:

$$(\exists R_1, \ldots, R_k.P)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists d_1, \ldots, d_k : (d, d_i) \in R_i^{\mathcal{I}} \text{ für } 1 \leq i \leq k \text{ und } (d_1, \ldots, d_k) \in P^{\mathcal{B}}\}$$

$$(\forall R_1, \ldots, R_k.P)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \forall d_1, \ldots, d_k : (d, d_i) \in R_i^{\mathcal{I}} \text{ für } 1 \leq i \leq k \text{ impliziert } (d_1, \ldots, d_k) \in P^{\mathcal{B}}\}$$

T 5.14

Unentscheidbarkeit mit konkreten Bereichen

Wir zeigen, dass bereits scheinbar einfache konkrete Bereiche zu Unentscheidbarkeit führen können. Betrachten kB $\mathcal{B}_1$ mit:

$$\Delta^{\mathcal{B}_1} = \mathbb{N}$$
$$\Phi^{\mathcal{B}_1} = \{=_0, =, +_1\}$$

wobei $=_0$ einstellig ist und $=$, $+_1$ zweistellig, mit den Extensionen:

$$=_0^{\mathcal{B}_1} = \{0\}$$
$$=^{\mathcal{B}_1} = \{(0,0),\ (1,1),\ (2,2),\ \dots\}$$
$$+_1^{\mathcal{B}_1} = \{(0,1),\ (1,2),\ (2,3),\ \dots\} \quad \leftarrow \textbf{Inkrementierung!}$$

Theorem 5.31

*Erfüllbarkeit von $\mathcal{ALC}(\mathcal{B}_1)$-Konzepten bzgl. TBoxen ist **unentscheidbar.***

Beweis: per Reduktion des Halteproblems für **2-Registermaschinen**

2-Registermaschinen

2-Registermaschinen sind ähnlich zu Turingmaschinen:

- Es gibt endlich viele Zustände.
- Statt eines Arbeitsbandes gibt es **zwei Register** mit Werten $\in \mathbb{N}$.
- Statt einer Übergangsfunktion gibt es **Instruktionen.**

Die Instruktionen erlauben es, den Wert eines Registers

- zu inkrementieren oder
- auf 0 zu testen und bei Wert $\neq 0$ zu dekrementieren.

Bei der zweiten Art Instruktion hängt der Folgezustand davon ab, ob der Registerwert 0 war.

2-Registermaschinen

Definition 5.32

(Deterministische) **2-Registermaschine (2RM)** ist Paar $M = (Q, P)$ mit $Q = \{q_0, \ldots, q_\ell\}$ Menge von **Zuständen** und $P = I_0, \ldots, I_{\ell-1}$ **Instruktionsfolge**.

Per Definition ist q_0 Startzustand, q_ℓ Stoppzustand.

Jede Instruktion I_i hat eine der folgenden Formen:

- $I_i = +(p, q_j)$ mit $p \in \{1, 2\}$ Register und q_j Folgezustand: **Inkrementierungsanweisung**
- $I_i = -(p, q_j, q_k)$ mit $p \in \{1, 2\}$ Register und q_j, q_k Folgezustände:
  Dekrementierungsanweisung mit Folgezustand q_j , wenn Register p den Wert 0 enthält, und q_k sonst

2-Registermaschinen

Definition 5.33

Konfiguration ist Tripel (q, m, n) mit q aktueller Zustand und $m, n \in \mathbb{N}$ Registerinhalte.

Konfigurationsübergänge $(q, m, n) \vdash_M (q', m', n')$ sind so definiert:

- Wenn $I_i = +(1, q_j)$, dann $(q_i, m, n) \vdash_M (q_j, \mathbf{m+1}, n)$
- Wenn $I_i = +(2, q_j)$, dann $(q_i, m, n) \vdash_M (q_j, m, \mathbf{n+1})$
- Wenn $I_i = -(1, q_j, q_k)$, dann $(q_i, 0, n) \vdash_M (\mathbf{q_j}, \mathbf{0}, n)$
  und $(q_i, m, n) \vdash_M (\mathbf{q_k}, \mathbf{m-1}, n)$ für $m > 0$
- Wenn $I_i = -(2, q_j, q_k)$, dann $(q_i, m, 0) \vdash_M (\mathbf{q_j}, m, \mathbf{0})$
  und $(q_i, m, n) \vdash_M (\mathbf{q_k}, m, \mathbf{n-1})$ für $n > 0$

Berechnung von M auf Eingabe $(m, n) \in \mathbb{N}^2$ ist die eindeutige (längste) Konfigurationsfolge

$$(q_0, m, n) = (p_0, m_0, n_0) \vdash_M (p_1, m_1, n_1) \vdash_M \cdots . \quad \text{T 5.15}$$

Die Reduktion

Definition 5.34

Das **Halteproblem für 2RMs** ist das folgende Problem:

Gegeben 2RM M,
entscheide, ob M gestartet auf Eingabe $(0, 0)$ hält (also q_ℓ erreicht).

Theorem 5.35 (Minsky 1967)

Das Halteproblem für 2RMs ist unentscheidbar.

Für Beweis von Theorem 5.31:

Gegeben 2RM M, konstruiere $\mathcal{ALC}(\mathcal{B}_1)$-TBox $\mathcal{T}_M$ und wähle einen Konzeptnamen A, so dass gilt:

$$M \text{ hält auf } (0,0) \quad \textbf{gdw.} \quad A \text{ unerfüllbar bzgl. } \mathcal{T}_M$$

Die Reduktion

Wir verwenden die folgenden Symbole:

- Konzeptnamen $Q_0, \ldots, Q_\ell$ für die Zustände $q_0, \ldots, q_\ell$
- Featurenamen f_1, f_2, um die Registerinhalte zu speichern
- Rollennamen r, um Nachfolgekonfigurationen zu verbinden
- Konzeptnamen A, der die Anfangskonfiguration anzeigt

Wir nehmen o. B. d. A. an: $q_0 \neq q_\ell$

Die Reduktion

Die TBox $\mathcal{T}_M$ enthält folgende Konzeptinklusionen:

(1) Start in Zustand q_0 und mit Registerwerten 0 :

$$A \sqsubseteq Q_0 \sqcap \exists f_1.{=_0} \sqcap \exists f_2.{=_0}$$

(2) Inkrementierung. Für alle $I_i = +(p, q_j)$:

$$Q_i \sqsubseteq \exists r.Q_j \sqcap \forall f_p\,, (r; f_p).{+_1} \sqcap \forall f_{\overline{p}}\,, (r; f_{\overline{p}}).{=}$$

(wobei $\overline{1} = 2$ und $\overline{2} = 1$)

(3) Dekrementierung. Für alle $I_i = -(p, q_j, q_k)$:

$$Q_i \sqcap \exists f_p.{=_0} \sqsubseteq \exists r.Q_j \sqcap \forall f_p\,, (r; f_p).{=} \sqcap \forall f_{\overline{p}}\,, (r; f_{\overline{p}}).{=}$$

$$Q_i \sqcap \neg\exists f_p.{=_0} \sqsubseteq \exists r.Q_k \sqcap \forall (r; f_p)\,, f_p.{+_1} \sqcap \forall f_{\overline{p}}\,, (r; f_{\overline{p}}).{=}$$

(4) Haltezustand wird nie erreicht:

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

Die Reduktion

Lemma 5.36

M hält auf $(0,0)$ ***gdw.*** *A unerfüllbar bzgl.* $\mathcal{T}_M$.

T 5.16

Andere konkrete Bereiche

Man kann zeigen, dass „=“ für die Reduktion nicht gebraucht wird.

Es führen also schon sehr schwache arithmetische konkrete Bereiche zu Unentscheidbarkeit.

Auch mit Wörtern und Konkatenation ist man sehr schnell unentscheidbar.

Entscheidbar bleibt man aber, wenn man sich auf Vergleichsoperatoren konzentriert wie etwa $\mathbb{N}$ mit $=$, $>$, $<$, $\geq$, $\leq$.

Literatur für dieses Kapitel (Basis)

Franz Baader, Ian Horrocks, Carsten Lutz, Uli Sattler.

An Introduction to Description Logic.

Cambridge University Press, 2007.

Kapitel 5: Complexity

Demnächst in SUUB verfügbar: `http://tinyurl.com/suub-intro-dl`

Bestellung beim Verlag: `http://www.cambridge.org/9780521695428`

Ich habe ein Exemplar und lasse Euch auf Nachfrage gern reinschauen.

Literatur für dieses Kapitel (weiterführend 1)

Vaughan R. Pratt.

A Practical Decision Method for Propositional Dynamic Logic: Preliminary Report.

In STOC 1978, S. 326–337. http://doi.acm.org/10.1145/800133.804362

Erstes Auftreten von Typelimination, allerdings für eine andere Logik (die jedoch mit $\mathcal{ALC}$ verwandt ist)

Larry J. Stockmeyer, Ashok K. Chandra.

Provably Difficult Combinatorial Games.

SIAM J. Comput. 8(2):151–174, 1979. http://dx.doi.org/10.1137/0208013

Unser Spiel$_1$ und Thm. 5.11 sind dort G_5 und Thm. 3.1 (S. 159f.).

Thomas J. Schaefer.

On the Complexity of Some Two-Person Perfect-Information Games.

J. Comput. Syst. Sci. 16(2):185–225, 1978.
https://doi.org/10.1016/0022-0000(78)90045-4

Unser Spiel$_2$ und Thm. ?? sind dort G_ω(CNF) und Lem. 2.3 (S. 192f.).

Literatur für dieses Kapitel (weiterführend 2)

Erich Grädel.

Why are Modal Logics So Robustly Decidable?

Current Trends in Theoretical Computer Science 393–408, 2001.

`http://www.logic.rwth-aachen.de/pub/graedel/Gr-trends01.ps`

Sehr verständlich geschrieben; gibt einen unterhaltsamen Einblick in Modallogik und relevante Fragmente der Prädikatenlogik und beleuchtet Gründe für deren Entscheidbarkeit.

Marvin L. Minsky.

Computation: Finite and Infinite Machines.

Prentice-Hall, 1967.

Dort wird gezeigt, dass das Halteproblem für 2RM unentscheidbar ist.

Literatur für dieses Kapitel (weiterführend 3)

Carsten Lutz.

Description Logics with Concrete Domains – A Survey.

Advances in Modal Logic Vol. 4: 265-296. King's College Publ., 2003.

`http://www.informatik.uni-bremen.de/tdki/research/papers/2003/Lutz-AiML4.ps.gz`

Umfassende Überblicksarbeit über DLs mit konkreten Bereichen.

Franz Baader, Sebastian Brandt, Carsten Lutz.

Pushing the $\mathcal{EL}$ Envelope.

IJCAI 2005: 364–369.

`http://ijcai.org/Proceedings/05/Papers/0372.pdf`

Untersucht systematisch Erweiterungen von $\mathcal{EL}$ (Kapitel 6). Die Erweiterung $\mathcal{EL}^{++}$ enthält eine abgeschwächte Form von konkreten Bereichen und ist immer noch in Polyzeit.

Technischer Report mit Beweisdetails: `http://lat.inf.tu-dresden.de/research/reports/2005/BaaderBrandtLutz-LTCS-05-01.ps.gz`