TECHNISCHE UNIVERSITÄT DRESDEN

THEORETISCHE INFORMATIK UND LOGIK

21. Vorlesung: Datalog

Markus Krötzsch

Lehrstuhl Wissensbasierte Systeme

TU Dresden, 6. Juli 2018

Rückblick: Formeln als Anfragen

linien:

Linie	Typ
85	Bus
3	Tram
F1	Fähre
...	...

haltestellen:

SID	Name	Rollstuhl
17	Hauptbahnhof	true
42	Helmholtzstr.	true
57	Stadtgutstr.	true
123	Gustav-Freytag-Str.	false
...	...	...

verbindung:

Von	Zu	Linie
57	42	85
17	789	3
...	...	...

Die einfache Arität der Prädikatenlogik wird durch ein Schema mit Namen (und oft auch Datentypen) ersetzt:

- linien[Linie:string, Typ:string]
- haltestellen[SID:int, Halt:string, Rollstuhl:bool]
- verbindung[Von:int, Zu:int, Linie:string]

Relationale Algebra: Parameter (Spalten) durch Namen adressiert
Prädikatenlogik: Parameter durch Reihenfolge adressiert

Die Anfrage $\exists z_{\text{Linie}}.(\text{verbindung}(x_{\text{Von}}, x_{\text{Zu}}, z_{\text{Linie}}) \wedge \text{linien}(z_{\text{Linie}}, x_{\text{Typ}}))$ entspricht einer (natürlichen) Join-Operation ($\wedge$) mit anschließender Projektion ($\exists$):
$\pi_{\text{Von,Zu,Linie}}(\text{verbindung} \bowtie \text{linien})$.

Rekursive Anfragen

Rückblick: Mit Prädikatenlogik können nur lokale Eigenschaften getestet werden.

Nichtlokale Eigenschaften wie die Erreichbarkeit in Graphen sind praktisch relevant (speziell in Graphdatenbanken)

Wie kann man solche Anfragen logisch ausdrücken?

Rekursive Anfragen

Rückblick: Mit Prädikatenlogik können nur lokale Eigenschaften getestet werden.

Nichtlokale Eigenschaften wie die Erreichbarkeit in Graphen sind praktisch relevant (speziell in Graphdatenbanken)

Wie kann man solche Anfragen logisch ausdrücken?

Idee: Um beliebig weit zu schauen muss man Rekursion einführen.

Beispiel: Eine Haltestelle ist von Helmholtzstr. aus erreichbar, wenn

(1) sie die Haltestelle Helmholtzstr. ist, oder

(2) sie neben einer Haltestelle liegt, die von Helmholtzstr. aus erreichbar ist.

Rekursion in Logik

Beispiel: Eine Haltestelle ist von Helmholtzstr. aus erreichbar, wenn

(1) sie die Haltestelle Helmholtzstr. ist, oder

(2) sie neben einer Haltestelle liegt, die von Helmholtzstr. aus erreichbar ist.

Wie kann man das in Logik ausdrücken?

Rekursion in Logik

Beispiel: Eine Haltestelle ist von Helmholtzstr. aus erreichbar, wenn

(1) sie die Haltestelle Helmholtzstr. ist, oder

(2) sie neben einer Haltestelle liegt, die von Helmholtzstr. aus erreichbar ist.

Wie kann man das in Logik ausdrücken?

Beispiel: Das Prädikat Erreichbar enthält alle von Helmholtzstr. erreichbaren Haltestellen, wenn die folgenden Formeln erfüllt sind:

(1) Erreichbar(`helmholtzstr`)

(2) $\forall x, y, z.\text{Erreichbar}(x) \wedge \text{verbindung}(x, y, z) \rightarrow \text{Erreichbar}(y)$

Model Checking?

Beispiel: Das Prädikat Erreichbar enthält alle von Helmholtzstr. erreichbaren Haltestellen, wenn die folgenden Formeln erfüllt sind:

(1) Erreichbar(`helmholtzstr`)

(2) $\forall x, y, z.\text{Erreichbar}(x) \wedge \text{verbindung}(x, y, z) \rightarrow \text{Erreichbar}(y)$

Sei Q die Menge der Formeln (1) und (2).

Model Checking?

Beispiel: Das Prädikat Erreichbar enthält alle von Helmholtzstr. erreichbaren Haltestellen, wenn die folgenden Formeln erfüllt sind:

(1) Erreichbar(`helmholtzstr`)

(2) $\forall x, y, z.\text{Erreichbar}(x) \wedge \text{verbindung}(x, y, z) \rightarrow \text{Erreichbar}(y)$

Sei Q die Menge der Formeln (1) und (2).

- Die Modelle von Q sind alle Interpretationen, in denen Erreichbar mindestens die von Helmholtzstr. aus erreichbaren Haltestellen enthält.
- Eine gegebene Datenbankinstanz, betrachtet als Interpretation, ist normalerweise kein Modell von Q, sofern es nicht schon eine entsprechende Tabelle für Erreichbar gibt.

$\rightsquigarrow$ Model Checking führt hier nicht zum gewünschten Ergebnis!

Datenbanken als logische Theorien

Wir nehmen daher eine andere Perspektive ein:

- Datenbankinstanzen $\mathcal{I}$ werden als endliche Menge von Fakten $\mathcal{F}_\mathcal{I}$ dargestellt:

$$\mathcal{F}_\mathcal{I} = \{p(c_1, \ldots, c_n) \mid c_1, \ldots, c_n \in \mathbf{C} \text{ und } \langle c_1^\mathcal{I}, \ldots, c_n^\mathcal{I} \rangle \in p^\mathcal{I}\}$$

  (wie zuvor nimmt man vereinfachend oft an, dass $c^\mathcal{I} = c$ gilt)

- Rekursive Anfragen können wie im Beispiel als prädikatenlogische Formelmenge Q dargestellt werden
- Ein Fakt $p(c_1, \ldots, c_n)$ ist eine Antwort auf die Anfrage Q auf der Datenbank $\mathcal{I}$ wenn gilt: $\mathcal{F}_\mathcal{I}, Q \models p(c_1, \ldots, c_n)$.

Das heißt, wir fassen Anfragebeantwortung als prädikatenlogisches Schließen auf (Entailment, nicht Model Checking).

Beispiel

Beispiel: Für die Anfrage

$$Q = \{\text{Erreichbar}(\texttt{helmholtzstr}),$$
$$\forall x, y, z.\, \text{Erreichbar}(x) \wedge \text{verbindung}(x, y, z) \rightarrow \text{Erreichbar}(y)\}$$

und die Datenbankinstanz $\mathcal{I}$ mit

$$\mathcal{F}_{\mathcal{I}} = \{\text{verbindung}(\texttt{helmholtzstr}, \texttt{stadtgutstr}, 85)$$
$$\text{verbindung}(\texttt{stadtgutstr}, \texttt{räcknitzhöhe}, 85)$$
$$\text{verbindung}(\texttt{räcknitzhöhe}, \texttt{zellescher\_weg}, 11)$$
$$\text{verbindung}(\texttt{schillerplatz}, \texttt{körnerplatz}, 61)\}$$

folgen genau die Antworten Erreichbar(helmholtzstr), Erreichbar(stadtgutstr), Erreichbar(räcknitzhöhe) und Erreichbar(zellescher_weg).

Sind rekursive Anfragen praktikabel?

Wir wissen: prädikatenlogisches Schließen ist unentscheidbar.

Ist die Beantwortung rekursiver Anfragen dann überhaupt möglich?

Sind rekursive Anfragen praktikabel?

Wir wissen: prädikatenlogisches Schließen ist unentscheidbar.

Ist die Beantwortung rekursiver Anfragen dann überhaupt möglich?

Ja, wenn wir uns auf bestimmte Formen von Anfragen beschränken.

Eine Datalog-Regel ist eine Formel der folgenden Form

$$\forall x_1, \ldots, x_\ell. \quad H \leftarrow B_1 \wedge \ldots \wedge B_n$$

wobei H und B_i prädikatenlogische Atome sind und $x_1, \ldots, x_\ell$ eine Liste aller in den Atomen vorkommenden Variablen ist. H heißt Kopf und $B_1 \wedge \ldots \wedge B_n$ Rumpf der Regel. Wir verlangen, dass jede Variable im Kopf auch im Rumpf vorkommt. Ein Fakt ist eine variablenfreie Regel mit $n = 0$.

Ein Datalog-Programm P ist eine Menge von Datalog-Regeln (einschl. Fakten).

Es ist üblich, die Allquantoren nicht explizit zu schreiben. Fakten schreibt man ohne $\leftarrow$.

Beispiele

Das vorige Beispiel war bereits ein Datalog-Programm:

$$\text{Erreichbar}(\texttt{helmholtzstr})$$

$$\text{Erreichbar}(y) \leftarrow \text{Erreichbar}(x) \wedge \text{verbindung}(x, y, z)$$

Beispiele

Das vorige Beispiel war bereits ein Datalog-Programm:

$$\text{Erreichbar}(\texttt{helmholtzstr})$$
$$\text{Erreichbar}(y) \leftarrow \text{Erreichbar}(x) \wedge \text{verbindung}(x, y, z)$$

Ein umfangreicheres Beispiel:

$$\text{vater}(\text{alice}, \text{bob}) \quad \text{mutter}(\text{alice}, \text{carla}) \quad \text{mutter}(\text{evan}, \text{carla}) \quad \text{vater}(\text{carla}, \text{david})$$
$$\text{Elter}(x, y) \leftarrow \text{vater}(x, y) \qquad \text{Elter}(x, y) \leftarrow \text{mutter}(x, y)$$
$$\text{Vorfahre}(x, y) \leftarrow \text{Elter}(x, y)$$
$$\text{Vorfahre}(x, z) \leftarrow \text{Elter}(x, y) \wedge \text{Vorfahre}(y, z)$$
$$\text{GleicheGeneration}(x, x) \leftarrow \text{Vorfahre}(x, y)$$
$$\text{GleicheGeneration}(y, y) \leftarrow \text{Vorfahre}(x, y)$$
$$\text{GleicheGeneration}(x, y) \leftarrow \text{Elter}(x, v) \wedge \text{Elter}(y, w) \wedge \text{GleicheGeneration}(v, w)$$

Auswertung von Datalog

Wie kann man logische Schlüsse aus Datalog ziehen?

Idee: Wende Regeln iterativ auf gegebene Fakten an, um neue Fakten abzuleiten

- Wir betrachten hier Datenbanken (Interpretationen) als logische Theorie, d.h., Menge von Grundfakten
- Die gegebenen Fakten kann man sich als Teil eines Datalog-Programms vorstellen

Auswertung von Datalog

Wie kann man logische Schlüsse aus Datalog ziehen?

Idee: Wende Regeln iterativ auf gegebene Fakten an, um neue Fakten abzuleiten

- Wir betrachten hier Datenbanken (Interpretationen) als logische Theorie, d.h., Menge von Grundfakten
- Die gegebenen Fakten kann man sich als Teil eines Datalog-Programms vorstellen

Der Konsequenzoperator T_P für ein Datalog-Programm P bildet endliche Mengen von Fakten $\mathcal{I}$ auf Mengen von Fakten ab:

$$T_P(\mathcal{I}) = \{H\theta \mid H \leftarrow B_1 \wedge \ldots \wedge B_n \in P \text{ und es gibt Substitution } \theta \text{ mit } B_1\theta, \ldots, B_n\theta \in \mathcal{I}\}$$

Auswertung von Datalog

Wie kann man logische Schlüsse aus Datalog ziehen?

Idee: Wende Regeln iterativ auf gegebene Fakten an, um neue Fakten abzuleiten

- Wir betrachten hier Datenbanken (Interpretationen) als logische Theorie, d.h., Menge von Grundfakten
- Die gegebenen Fakten kann man sich als Teil eines Datalog-Programms vorstellen

Der Konsequenzoperator T_P für ein Datalog-Programm P bildet endliche Mengen von Fakten $\mathcal{I}$ auf Mengen von Fakten ab:

$$T_P(\mathcal{I}) = \{H\theta \mid H \leftarrow B_1 \wedge \ldots \wedge B_n \in P \text{ und es gibt Substitution } \theta \text{ mit } B_1\theta, \ldots, B_n\theta \in \mathcal{I}\}$$

Beobachtungen:

- Substitutionen θ mit $B_1\theta, \ldots, B_n\theta \in \mathcal{I}$ sind einfach Antworten auf die Datenbankanfrage $B_1 \wedge \ldots \wedge B_n$ über Datenbank $\mathcal{I} \rightsquigarrow$ praktisch berechenbar
- θ muss alle Variablen in der angewendeten Regel auf Konstanten (Domänenelemente) aus $\mathcal{I}$ abbilden

T_P iterativ anwenden

Zur Ermittlung aller Schlüsse muss man T_P iterativ anwenden:

Für ein Datalog-Programm P definieren wir rekursiv:

- $T_P^0 = \emptyset$
- $T_P^{i+1} = T_P(T_P^i)$ für alle $i \geq 0$

T_P iterativ anwenden

Zur Ermittlung aller Schlüsse muss man T_P iterativ anwenden:

Für ein Datalog-Programm P definieren wir rekursiv:

- $T_P^0 = \emptyset$
- $T_P^{i+1} = T_P(T_P^i)$ für alle $i \geq 0$

Beobachtungen:

- $T_P^1 = T_P(\emptyset)$ ist die Menge aller Fakten in P
- T_P^i enthält nur Fakten, die man bilden kann, indem man einen Regelkopf aus P mit Konstanten aus P instanziiert $\rightsquigarrow$ endlich viele mögliche Schlüsse

T_P iterativ anwenden

Zur Ermittlung aller Schlüsse muss man T_P iterativ anwenden:

Für ein Datalog-Programm P definieren wir rekursiv:

- $T_P^0 = \emptyset$
- $T_P^{i+1} = T_P(T_P^i)$ für alle $i \geq 0$

Beobachtungen:

- $T_P^1 = T_P(\emptyset)$ ist die Menge aller Fakten in P
- T_P^i enthält nur Fakten, die man bilden kann, indem man einen Regelkopf aus P mit Konstanten aus P instanziiert $\rightsquigarrow$ endlich viele mögliche Schlüsse

T_P erreicht also nach endlich vielen Schritten einen Grenzwert, definiert wie folgt:

$$T_P^\infty = \bigcup_{i \geq 0} T_P^i$$

Beispiel

Für das Programm P mit den Regeln

$$\text{vater}(\text{alice}, \text{bob}) \quad \text{mutter}(\text{alice}, \text{carla}) \quad \text{mutter}(\text{evan}, \text{carla}) \quad \text{vater}(\text{carla}, \text{david})$$

$$\text{Elter}(x, y) \leftarrow \text{vater}(x, y) \qquad \text{Elter}(x, y) \leftarrow \text{mutter}(x, y)$$

$$\text{GG}(x, x) \leftarrow \text{Elter}(x, y)$$

$$\text{GG}(y, y) \leftarrow \text{Elter}(x, y)$$

$$\text{GG}(x, y) \leftarrow \text{Elter}(x, v) \wedge \text{Elter}(y, w) \wedge \text{GG}(v, w)$$

(GG = GleicheGeneration) ergibt sich:

- $T_P^0 = \emptyset$
- $T_P^1 = \{\text{vater}(\text{alice}, \text{bob}), \text{mutter}(\text{alice}, \text{carla}), \text{mutter}(\text{evan}, \text{carla}), \text{vater}(\text{carla}, \text{david})\}$
- $T_P^2 = T_P^1 \cup \{\text{Elter}(\text{alice}, \text{bob}), \text{Elter}(\text{alice}, \text{carla}), \text{Elter}(\text{evan}, \text{carla}), \text{Elter}(\text{carla}, \text{david})\}$
- $T_P^3 = T_P^2 \cup \{\text{GG}(\text{alice}, \text{alice}), \text{GG}(\text{bob}, \text{bob}), \text{GG}(\text{carla}, \text{carla}), \text{GG}(\text{evan}, \text{evan})\}$
- $T_P^4 = T_P^3 \cup \{\text{GG}(\text{alice}, \text{evan}), \text{GG}(\text{evan}, \text{alice})\}$
- $T_P^5 = T_P^4 = T_P^\infty$

Semantische Bedeutung von T_P

Beobachtung 1: Jede Datalog-Regel $H \leftarrow B_1 \wedge \ldots \wedge B_n$ entspricht einer Klausel $H \vee \neg B_1 \vee \ldots \vee \neg B_n$, wobei jeweils alle Variablen allquantifiziert sind.

Semantische Bedeutung von T_P

Beobachtung 1: Jede Datalog-Regel $H \leftarrow B_1 \wedge \ldots \wedge B_n$ entspricht einer Klausel $H \vee \neg B_1 \vee \ldots \vee \neg B_n$, wobei jeweils alle Variablen allquantifiziert sind.

$\rightsquigarrow$ Datalog-Programme sind syntaktische Varianten skolemisierter Klauseln

$\rightsquigarrow$ Für die Inferenz von Fakten kann man sich auf Herbrand-Modelle beschränken

Semantische Bedeutung von T_P

Beobachtung 1: Jede Datalog-Regel $H \leftarrow B_1 \wedge \ldots \wedge B_n$ entspricht einer Klausel $H \vee \neg B_1 \vee \ldots \vee \neg B_n$, wobei jeweils alle Variablen allquantifiziert sind.

$\rightsquigarrow$ Datalog-Programme sind syntaktische Varianten skolemisierter Klauseln
$\rightsquigarrow$ Für die Inferenz von Fakten kann man sich auf Herbrand-Modelle beschränken

Beobachtung 2: Die Berechnung eines Fakts $H\theta$ durch Anwendung einer solchen Regel entspricht einer (Hyper)-Resolution der Klausel $H \vee \neg B_1 \vee \ldots \vee \neg B_n$ mit den Fakten $B_1\theta, \ldots, B_n\theta$, wobei θ der kleinste Unifikator ist.

Semantische Bedeutung von T_P

Beobachtung 1: Jede Datalog-Regel $H \leftarrow B_1 \wedge \ldots \wedge B_n$ entspricht einer Klausel $H \vee \neg B_1 \vee \ldots \vee \neg B_n$, wobei jeweils alle Variablen allquantifiziert sind.

$\rightsquigarrow$ Datalog-Programme sind syntaktische Varianten skolemisierter Klauseln
$\rightsquigarrow$ Für die Inferenz von Fakten kann man sich auf Herbrand-Modelle beschränken

Beobachtung 2: Die Berechnung eines Fakts $H\theta$ durch Anwendung einer solchen Regel entspricht einer (Hyper)-Resolution der Klausel $H \vee \neg B_1 \vee \ldots \vee \neg B_n$ mit den Fakten $B_1\theta, \ldots, B_n\theta$, wobei θ der kleinste Unifikator ist.

$\rightsquigarrow$ Abgeleitete Fakten sind logische Konsequenzen (Korrektheit)

Semantische Bedeutung von T_P

Beobachtung 1: Jede Datalog-Regel $H \leftarrow B_1 \wedge \ldots \wedge B_n$ entspricht einer Klausel $H \vee \neg B_1 \vee \ldots \vee \neg B_n$, wobei jeweils alle Variablen allquantifiziert sind.

$\rightsquigarrow$ Datalog-Programme sind syntaktische Varianten skolemisierter Klauseln
$\rightsquigarrow$ Für die Inferenz von Fakten kann man sich auf Herbrand-Modelle beschränken

Beobachtung 2: Die Berechnung eines Fakts $H\theta$ durch Anwendung einer solchen Regel entspricht einer (Hyper)-Resolution der Klausel $H \vee \neg B_1 \vee \ldots \vee \neg B_n$ mit den Fakten $B_1\theta, \ldots, B_n\theta$, wobei θ der kleinste Unifikator ist.

$\rightsquigarrow$ Abgeleitete Fakten sind logische Konsequenzen (Korrektheit)

Mithilfe dieser Einsichten lässt sich zeigen, dass T_P^∞ zur Berechnung der logischen Schlussfolgerung geeignet ist:

Satz: Für ein Datalog-Programm P ist T_P^∞ das kleinste Herbrand-Modell. Das heißt, für einen beliebigen Fakt F gilt $F \in T_P^\infty$ gdw. F in jedem Herbrand-Modell vorkommt gdw. $P \models F$.

Ableitungsbäume

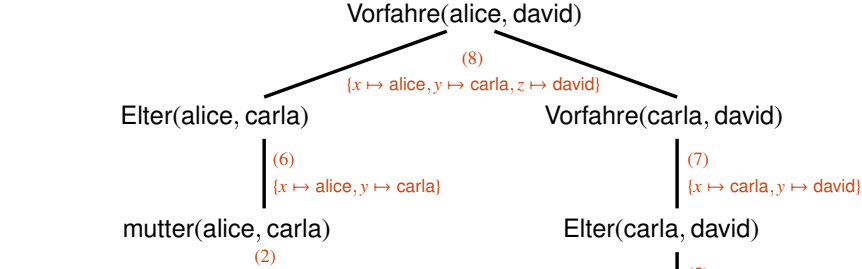
Die Folgerung $P \models F$ lässt sich als endlicher Baum darstellen:

- Jeder Knoten ist ein variablenfreies Atom
- Jeder Elternknoten entsteht durch Anwendung einer Regel aus P auf seine Kindknoten
- Jeder Blattknoten ist ein gegebener Fakt aus P
- Jeder Knoten wird zusätzlich mit der Regel und Substitution θ beschriftet, die zur Ableitung angewendet wurden

$\rightsquigarrow$ Ableitungsbaum stellt die Resolutionsableitung des Fakts and der Wurzel des Baums grafisch dar

Beobachtung: Für jeden Fakt $F \in T_P^\infty$ gibt es mindestens einen Ableitungsbaum mit Wurzel F.

Beispiel



(1)	vater(alice, bob)
(2)	mutter(alice, carla)
(3)	mutter(evan, carla)
(4)	vater(carla, david)
(5)	$\text{Elter}(x, y) \leftarrow \text{vater}(x, y)$
(6)	$\text{Elter}(x, y) \leftarrow \text{mutter}(x, y)$
(7)	$\text{Vorfahre}(x, y) \leftarrow \text{Elter}(x, y)$
(8)	$\text{Vorfahre}(x, z) \leftarrow \text{Elter}(x, y) \wedge \text{Vorfahre}(y, z)$

Rückblick: Kontextfreie Grammatiken

In der Vorlesung Formale Systeme haben wir auch eine Art von Ableitungsbäumen kennengelernt: Ableitungsbäume für kontextfreie Grammatiken

Rückblick: Kontextfreie Grammatiken

In der Vorlesung Formale Systeme haben wir auch eine Art von Ableitungsbäumen kennengelernt: Ableitungsbäume für kontextfreie Grammatiken

Diese sind eng mit denen aus Datalog verwandt, weil man kontextfreie Grammatiken in Datalog darstellen kann:

Gegeben: Kontextfreie Grammatik $G = \langle V, \Sigma, P, S \rangle$

- Für jedes Nichtterminal A $\in V$, sei A ein zweistelliges Prädikat
- Für jedes Terminal b $\in V$, sei b ein zweistelliges Prädikat
- Für jede Produktionsregel $A \to v$ mit $v = v_1 \dots v_n \in (\Sigma \cup V)^*$ sei $A(x_0, x_n) \leftarrow v_1(x_0, x_1) \wedge \dots \wedge v_n(x_{n-1}, x_n)$ eine Datalogregel
- Die Menge aller entsprechenden Regel bezeichnen wir mit P_G

Wir betrachten Datenbanken, die nur Prädikate aus Σ enthalten

Rückblick: Kontextfreie Grammatiken

In der Vorlesung Formale Systeme haben wir auch eine Art von Ableitungsbäumen kennengelernt: Ableitungsbäume für kontextfreie Grammatiken

Diese sind eng mit denen aus Datalog verwandt, weil man kontextfreie Grammatiken in Datalog darstellen kann:

Gegeben: Kontextfreie Grammatik $G = \langle V, \Sigma, P, S \rangle$

- Für jedes Nichtterminal $\mathtt{A} \in V$, sei A ein zweistelliges Prädikat
- Für jedes Terminal $\mathtt{b} \in V$, sei b ein zweistelliges Prädikat
- Für jede Produktionsregel $A \to v$ mit $v = v_1 \dots v_n \in (\Sigma \cup V)^*$ sei $A(x_0, x_n) \leftarrow v_1(x_0, x_1) \wedge \dots \wedge v_n(x_{n-1}, x_n)$ eine Datalogregel
- Die Menge aller entsprechenden Regel bezeichnen wir mit P_G

Wir betrachten Datenbanken, die nur Prädikate aus Σ enthalten

Dann gilt: $P_G \models S(a, b)$ genau dann wenn es zwischen a und b einen Pfad $w_1(a, c_1), w_2(c_1, c_2), \dots, w_\ell(c_{\ell-1}, b)$ gibt, wobei $\mathtt{w_1} \cdots \mathtt{w_\ell} \in \mathbf{L}(G)$

Wozu Ableitungsbäume?

Baumartige Darstellungen von logischen Ableitungen sind in verschiedenen Zusammenhängen hilfreich:

- Intuitive Darstellung der Ableitungsschritte
- Umformungsoperationen auf Bäumen erlauben die Erzeugung neuer Ableitungen
  $\leadsto$ hilfreiche Beweistechnik

  Beispiel: Wir haben Ableitungsbäume kontextfreier Grammatiken verwendet um das Pumpinglemma für kontextfrei Sprachen zu zeigen. Damit konnte man zeigen, dass manche Sprachen nicht kontextfrei sind.

  Analoge Techniken kann man auch für Datalog anwenden, zum Beispiel um zu zeigen, dass kein Datalog-Programm Paare von Elementen finden kann, zwischen denen es einen Pfad aus einem binären Prädikat p gibt, dessen Länge eine Primzahl ist.

- Zur Analyse von Datalogprogrammen werden oft (allgemeinere) endliche Automaten verwendet, welche auf Ableitungsbäumen arbeiten (Verallgemeinerung regulärer Sprachen von Wörtern auf Bäumen).

Komplexität

Mithilfe von T_P kann man logische Konsequenzen berechnen

Wie aufwändig ist das?

Komplexität

Mithilfe von T_P kann man logische Konsequenzen berechnen
Wie aufwändig ist das?

Worst Case?

- Sei p die Anzahl der Prädikatensmbole, a deren maximale Arität, x die maximale Zahl an Variablen pro Regel und n die Zahl der Konstanten
- Dann gibt es insgesamt $\leq p \cdot n^a$ variablenfreie Fakten, die abgeleitet werden könnten
- Für eine Regel gibt es maximal n^x Substitutionen, die bei der Ableitung eine Rolle spielen könnten

Komplexität

Mithilfe von T_P kann man logische Konsequenzen berechnen
Wie aufwändig ist das?

Worst Case?

- Sei p die Anzahl der Prädikatensmbole, a deren maximale Arität, x die maximale Zahl an Variablen pro Regel und n die Zahl der Konstanten
- Dann gibt es insgesamt $\leq p \cdot n^a$ variablenfreie Fakten, die abgeleitet werden könnten
- Für eine Regel gibt es maximal n^x Substitutionen, die bei der Ableitung eine Rolle spielen könnten

$\rightsquigarrow$ Berechnung von T_P^∞ in exponentieller Zeit möglich

Komplexität

Mithilfe von T_P kann man logische Konsequenzen berechnen
Wie aufwändig ist das?

Worst Case?

- Sei p die Anzahl der Prädikatensmbole, a deren maximale Arität, x die maximale Zahl an Variablen pro Regel und n die Zahl der Konstanten
- Dann gibt es insgesamt $\leq p \cdot n^a$ variablenfreie Fakten, die abgeleitet werden könnten
- Für eine Regel gibt es maximal n^x Substitutionen, die bei der Ableitung eine Rolle spielen könnten

$\rightsquigarrow$ Berechnung von T_P^∞ in exponentieller Zeit möglich

Man kann zeigen, dass dies worst-case-optimal ist:

Satz: Das Problem der Schlussfolgerung von Fakten ("$P \models F$?") für Datalog ist Exp-Time-vollständig.

Ist Datalog praktisch?

ExpTime ist eine ziemlich hohe Komplexität – Ist Datalog praktisch implementierbar?

Ist Datalog praktisch?

ExpTime ist eine ziemlich hohe Komplexität – Ist Datalog praktisch implementierbar?

Ja!

- Die Worst-Case-Komplexität erfordert, dass die Stelligkeit von Prädikaten unbeschränkt wachsen kann $\leadsto$ in Anwendungen sind sehr große Stelligkeiten untypisch
- In Abhängigkeit von der Größe der Datenbank (der Zahl der Fakten) wächst die Laufzeit lediglich polynomiell $\leadsto$ gutes Skalierungsverhalten für große Datenmengen
- Es gibt inzwischen eine Reihe sehr effizienter Implementierungen, z.B.
  - hochskalierbare speicherbasierte Systeme: z.B. VLog (VU Amsterdam, TU Dresden), RDFox (Oxford)
  - Systeme basierend auf relationalen Datenbanken: z.B. Llunatic
  - Systeme für komplexere Logiken, die Datalog als Sonderfall unterstützen: z.B. DLV (Answer Set Programming), E (Theorembeweiser)

Logik höherer Ordnung

Prädikatenlogik ist genau genommen Prädikatenlogik erster Stufe.

Hintergrund:

- Erste Stufe: Quantoren beziehen sich auf Domänenelemente

  Beispiel: „Jede natürliche Zahl n hat einen Nachfolger $s(n)$“

- Zweite Stufe: Quantoren beziehen sich auf Prädikate

  Beispiel: „Für jede Menge M gilt: Enthält M die Zahl 0 und mit jeder natürlichen Zahl n auch stets deren Nachfolger $s(n)$, so enthält M alle natürlichen Zahlen.“

Logik zweiter Ordnung:

- Ausdrucksstärker; kann z.B. die natürlichen Zahlen exakt charakterisieren
- Schwieriger: kein vollständiges und korrektes Beweisverfahren

$\rightsquigarrow$ siehe Vorlesung Advanced Logic

Logik höherer Ordnung: Syntax und Semantik

Syntax: Wie in Prädikatenlogik, aber mit quantifizierten Prädikaten-Variablen.
(Die Stelligkeit einer Prädikaten-Variablen muss jeweils klar festgelegt werden)

Beispiel: “Für jede Menge M gilt: Enthält M die Zahl 0 und mit jeder natürlichen Zahl n auch stets deren Nachfolger $s(n)$, so enthält M alle natürlichen Zahlen.”

$$\forall M.(M(0) \wedge \forall x.(M(x) \rightarrow M(s(x))) \rightarrow \forall y.M(y))$$

Wir verwenden hier ein Funktionssymbol s zur Darstellung von Nachfolgern.

Logik höherer Ordnung: Syntax und Semantik

Syntax: Wie in Prädikatenlogik, aber mit quantifizierten Prädikaten-Variablen.
(Die Stelligkeit einer Prädikaten-Variablen muss jeweils klar festgelegt werden)

Beispiel: "Für jede Menge M gilt: Enthält M die Zahl 0 und mit jeder natürlichen Zahl n auch stets deren Nachfolger $s(n)$, so enthält M alle natürlichen Zahlen."

$$\forall M.(M(0) \wedge \forall x.(M(x) \rightarrow M(s(x))) \rightarrow \forall y.M(y))$$

Wir verwenden hier ein Funktionssymbol s zur Darstellung von Nachfolgern.

Semantik: wie erwartet (gleiche Interpretationen wie in erster Stufen; Interprätation von Prädikaten-Variablen mit Zuweisungen wie bei Objektvariablen in erster Stufe)

(Intuition: zweite Stufe/erste Stufe = QBF/Aussagenlogik)

Logik höherer Ordnung: logisches Schließen

Offenbar ist Schließen in Logik zweiter Stufe mindestens genauso schwer wie in Logik zweiter Stufe. Tatsächlich ist es noch deutlich schwerer:

Fakt: Logisches Schließen in Logik höherer Ordnung ist nicht semi-entscheidbar und insbesondere gibt es kein korrektes und vollständiges Ableitungsverfahren für diese Logik.

Logik höherer Ordnung und Datalog

Idee: Die Fakten, die in allen Modellen eines (Datalog-Programms) gefolgert werden können, sind genau diejenigen, die in jeder erfüllenden Interpretation der Datalog-Prädikate gelten.

Beispiel: Das Datalog-Programm

$$\text{Erreichbar}(\texttt{helmholtzstr})$$

$$\text{Erreichbar}(y) \leftarrow \text{Erreichbar}(x) \wedge \text{verbindung}(x, y, z)$$

kann als Formel der Logik zweiter Stufe wie folgt dargestellt werden:

$$\forall \text{Erreichbar}.\Big((\text{Erreichbar}(\texttt{helmholtzstr}) \wedge (\forall x, y, z.\text{Erreichbar}(x) \wedge \text{verbindung}(x, y, z) \rightarrow \text{Erreichbar}(y))) \rightarrow \text{Erreichbar}(v)\Big)$$

Die Formel hat eine freie Variable v und stellt Erreichbar als Prädikaten-Variable dar. Ein Fakt $\text{Erreichbar}(a)$ folgt aus dem ursprünglichen Programm über einer Datenbank $\mathcal{I}$ genau dann wenn $\mathcal{I}$ die Formel mit Variablenbelegung $v \mapsto a$ erfüllt.

Model Checking!

Das vorige Beispiel zeigt:

> Die Beantwortung von Anfragen in Datalog entspricht einem Auswertungsproblem (Model Checking) für spezielle Formeln zweiter Ordnung über endlichen Modellen.

Diese Sicht wird gegenüber der Betrachtung als Logik erster Stufe bevorzugt, weil dadurch abgeleitete Prädikate zu lokalen Variablen des Programms werden, anstatt globaler Teil von Interpretationen (Datenbanken) zu sein

Zusammenfassung und Ausblick

Datalog erlaubt die Darstellung rekursiver Anfragen in Logik

Anfragebeantwortung in Datalog
= logisches Schließen in Prädikatenlogik = Auswertungsproblem in Logik zweiter Stufe
(ExpTime-vollständig, aber polynomiell bzgl. Datenbankgröße)

Ableitungen in Datalog können berechnet und dargestellt werden:

- mit dem T_P-Operator
- durch Ableitungsbäume

Was erwartet uns als nächstes?

- Gödel
- Probeklausur und 2. Repetitorium