Gedächnisprotokoll
Datenbanken, KI, KR
Prüfer: Prof.Lakemeyer
Prüfling: Markus de Brün
Note: 1,0
Dauer: 50 min

Erstmal eine Bemerkung vorab: Ich habe in der Prüfung viel gestammelt und konnte nicht alle Fragen beantworten. Das hat dann manchmal zu laaaangen Gesprächspausen geführt. Das Protokoll ließt sich zwar sehr flüssig, aber dem war nicht so ;-)
Trotzdem muss ich sagen, dass die Atmosphäre sehr angenehm war. Herr Lakemeyer hat mir Zeit zum Nachdenken gegeben und war bei Problemen sehr hilfsbereit.
Fragen oder Korrekturvorschläge bitte an markus.bruen@rwth-aachen.de

Lakemeyer: Ich würde vorschlagen, dass wir Datenbanken, danach KI und zum Schluss KR machen. Okay?

Ich: Okay.

Gut. Was kann man denn mit der Entity-Relationship Modell machen?

Das bietet eine Möglichkeit zur graphischen Darstellung des Weltmodells, dass wir abbilden wollen. Es gibt da Entities, die über Relationen miteinander verbunden sind und Attribute. Den Relationen können auch Kardinalitäten zugeordnet werden. Da gibt es zum einen die funktionale Notation und die (min,max)-Notation.

Aber da gibt es doch noch mehr. Was ist denn mit Vererbung?

Um Spezialisierung bzw Generalisierung abzubilden gibt es *is-a* und für Aggregation das *part-of*.

Und warum wollen wir Vererbung haben?

Also zum einen ist das ja viel übersichtlicher. Viele Dinge sind ja hierarchisch organisiert, so dass die Attribute einer Oberklasse an die Unterklasse weitergegeben werden.

Hmm. Wir wollen damit Redundanzen vermeiden, damit die Attribute nicht mehrfach gespeichert werden müssen. Wenn wir das ins relationale Modell übertragen wollen, wie geht dass denn?

Also eine direkte Möglichkeit gibt es da nicht. Das müssten wir dann über Sichten realisieren.

Naja. Sichten sind eine Möglichkeit. Wie würden sie das denn machen, wenn sie Personen haben, mit Name und Adresse, und Studenten mit Matrikelnummer und Fach. Wie kann man denn jetzt sagen, dass Studenten auch Personen sind? Sie können das auch direkt in SQL angeben.

Also... Wir haben Relationen für Personen mit den Attributen "Name" und "Adresse" und Studenten mit "MatrikelNr" und "Fach". Dann brauchen wir aber noch ein Attribut, mit dem man die beiden verknüpfen kann, z.B. "PersonenNr", die dann in beiden Relationen vorkommen muss. Da kann man dann einen join drüber machen. Dann würde man eine Sicht anlegen mit... *create view X ...* (Die Syntax war auf jeden Fall falsch, aber das machte wohl nichts. Ich weiß noch, dass ich was von "Sichten stellen virtuelle Relationen zur Verfügung" erzählt habe, aber nicht mehr wie ich darauf gekommen bin.)

Und wie ist das mit der Updatefähigkeit von Sichten? Gibt es da Probleme?

Ja, weil wenn wir etwas in die Sicht statt in eine Relation einfügen wollen, dann müssen wir sicherstellen, dass wir das auch der Basisrelation noch zuordnen können. Da gibt es verschiedene Kriterien: Es muss ein Schlüssel der Basisrelation erhalten sein; in der Sicht darf keine Aggregatfunktion oder so etwas wie *having* oder *distinct* verwendet werden; Und die Sicht darf sich nur auf eine Tabelle (also Relation oder Sicht) beziehen, die natürlich selbst wieder updatefähig sein muss.

Aber hier haben sie ja zwei Relationen verwendet. Ist das nicht updatefähig?

Hmm... Also ich sehe nichts, was dagegen sprechen würde... Aber die in SQL updatefähigen Sichten sind ja echte Untermenge der theoretisch updatefähigen Sichten.

Ja. Ich sehe hier auch nichts. Das könnte sogar gehen. Warum stellt man denn diese Kriterien auf und lässt das nicht bei Bedarf überprüfen, ob die Sicht updatefähig ist?

Ich vermute mal, dass das zu kompliziert ist.

Unentscheidbar um genau zu sein. Dabei

sieht das so einfach aus. Kommen wir mal zur KI. Wir hatten da verschiedene Agenten Typen kennen gelernt. Z.B. den Simple-Reflex-Agent und dann noch... Ja. Was denn noch?

Den Model-Based-Agent, der eine interne Beschreibung über den Weltzustand beinhaltet. Danach einen Goal-Based-Agent, der zusätlich noch eine Ziel Beschreibung enthält, weil ein Weltmodell nicht immer ausreichend ist, um die richtige Aktion auszuwählen. Also die, die das Leistungsmaß maximieren soll. Zum Schluss hatten wir noch den Utility-Based-Agent, bei dem auch die Möglichkeit besteht Präfenenzen für verschiedene Ziele anzugeben.

Und wir hatten den lernenden Agenten. Was ändert sich denn da?

Da wir der Agent noch um ein Lernelement, einen Kritiker und einen Problemgenerator erweitert. Das Lernelement ist dafür verantwortlich das Performance-Element, also das was vorher dem Agenten entsprach, zu verbessern. Dafür stellt der Kritiker Rückmeldungen zur Verfügung. Da kann man denn zwischen supervised, reinforcement und unsupervised lerning unterscheiden.

Wir haben da ja im wesentlichen nur das supervised lerning betrachtet. Was ist denn da die Rückmeldung vom Kritiker?

Also beim induktiven Lernen haben wir ja eine Menge an Beispielen, also Tupeln der Form (x,f(x)). Und wir wollen ja abstrakt gesehen diese Funktion f annähren. Das kann zum Beispiel eine Abbildung von Beobachtungen auf Aktionen sein. Diese Beispielmenge teilt man dann auf in eine Trainings- und eine Testmenge. Mit Ersterer trainiert man nun den Agenten und mit der Testmenge überprüft man dann wie akkurat dir Funktion angenährt wurde.

Ja. Also gibt der Kritiker diese Beispiele vor. Warum gibt es denn verschiedene Lernverfahren? Ist doch viel schöner nur eins zu haben, oder?

Nicht alle Lernverfahren sind für alle Probleme gleichermaßen gut geeignet. Wir hatten zum Beispiel die Mehrheitsfunktion, die sich mit Entscheidungsbäumen nur sehr schlecht lernen lässt.

Woran liegt das denn?

Beim Lernen mit Entscheidungsbäumen versucht man ja Attribute zu finden, welche die Beispielmenge möglicht in Mengen aufteilen, die nur Beispiele einer Klassifikation enthalten. So etwas lässt sich aber hier nicht finden, da alle Attribute gleich schlecht sind.

Dann kann man doch Entscheidungslisten nehmen, oder?

Nein. Die E-Listen sind ausdrucksschwächer als die E-Bäume. Also es sei denn wir wählen $k = n$.

Okay. Aber mit neuronalen Netzen geht das doch?

Ja. Da sogar sehr einfach. Mit Perceptronen. Wir brauchen nur n Eingabeeinheiten, belegen alle W_j mit 1 und wählen $t = n/2$.

Aber die Mehrheitsfunktion will ja keiner lernen. Was ist denn der Nachteil an diesen Perceprtonen?

Man kann damit nur linear separierbare Funktionen darstellen.

Okay, dann nehmen wir jetzt halt noch versteckte Ebenen hinzu. Jetzt können wir viel mehr darstellen.

Ja, aber es gibt leider immer noch Nachteile. Man kann nicht sagen, wie man die Struktur des Netzes wählen muss, um etwas gut darstellen zu können. Außerdem können wir hier nicht mehr garantieren, dass das Lernen konvergiert. Das ist nur bei Perceptronen gegeben.

Was bedeutet denn Lernen in diesem Zusammenhang?

Wir suchen nach geeigneten Parametern, also den Gewichtungen der Links, mit denen sich dann die Funktion gut annähren lässt.

Und warum konvergiert das nicht?

Wir wollen ja den Fehler minimieren. Wenn wir das als gradient descent search auffassen, dann kann es hier auch lokale Minima geben, wo wir dann festhängen.

Das reicht jetzt aber auch zum Lernen. Wie ist denn die Formel von Bayes? Und erklären sie mal wozu man die braucht.

$P(A|B) = P(B|A) * P(A) * P(B)^{-1} = \alpha * P(B|A) * P(A)$
Die kann man zum Beispiel in der medizinischen Diagnose verwenden. Also wenn A eine Krankheit ist und B ein Symptom, dann wollen wir wissen, wie hoch die Wahrscheinlichkeit ist, dass man A hat wenn man unter B leidet. Das könnte man zum einen natürlich statistisch erfassen, aber dann würde man nicht erkennen wie sich $P(A|B)$ ändert, wenn sich $P(A)$ ändert. Also zum Beispiel im Falle einer A-Epidemie. Da empfiehlt es sich die Formel zu benutzen, weil sich der kausale Zusammenhang $P(B|A)$ ja in der Regel nicht ändert.

Und wir kommt man an α?

Das kann man im Nachhinein wählen, so dass sich die möglichen Werte alle zu Eins addieren.

Okay. Wenn A und B boolesche Werte sind: Wie berechnet man dann α?

???
(Das war übel. Ich habe angefangen was hin zuschreiben, bin aber nicht weiter gekommen. Nach ein oder zwei Minuten Stille habe ich dann gesagt, dass ich da nicht weiterkomme. Prof. Lakemeyer hat dann Herrn Beck (den Beisitzer) gefragt, ob das gut aussähe. Herr Beck meinte, dass da irgendwas nicht stimmt. Daraufhin hat sich Herr Lakemeyer ein Buch geschnappt und noch zwei/drei Minuten gelesen um dann festzustellen, dass das in diesem Buch nicht drinsteht...)

Okay, dann lassen wir das mal und kommen zu KR. Bla... Wissensbasis... Bla... logische Folgerungen... Bla... Resolution... Bla. Warum ist denn Resolution für die KI nicht so gut geeignet?

Weil die Erfüllbarkeit in der Prädikatenlogik nur semi-entscheidbar ist. Also wenn etwas unerfüllbar ist, dann können wir auch irgendwann die leere Klausel herleiten. Sonst kann das aber auch nicht terminieren und wir wissen dann nicht, wie lange wir warten müssen. Das ist schlecht, denn wir wollen ja relativ schnell Entscheidungen haben.

Woran liegt das denn intuitiv, dass das so schwer ist?

Ein Problem ist, dass Variablen ja stellvertretend für alle Instantiierungen stehen.

Okay, aber in der Aussagenlogik haben wir die nicht und trotzdem ist das beste Verfahren das man kennt exponentiell.

Joah...

In der Vorlesung hatten wir das Beispiel mit den Blöcken: Ist ein grüner Block... und so weiter. Was sollte das denn verdeutlichen?

Pff...

Also wie würden sie denn daran gehen. Was muss man machen?

Man muss alle Möglichkeiten ausprobieren.

Genau. Durch Fallunterscheidungen wird das so schwer. Warum klappt das denn in Datenbanken so effizient?

Weil Datenbanken davon ausgehen über vollständiges Wissen zu verfügen. Also closed world assumption und domain closure. Dadurch lassen sich, wenn die Datenbasis konsistent ist, Anfragen auf Atome reduzieren. Das entspricht dann einem einfachen Nachschlagen.

Und was kann man in der KR da besser machen?

Wir können Horn-Formeln verwenden. Da ist das in linearer Zeit entscheidbar.

Wie? In was für Horn-Formeln?

In der Aussagenlogik. In der Prädikatenlogik ist das immer noch nur semi-entscheidbar.

Was ist denn mit prozeduraler Kontrolle?

Also da haben wir zum Beispiel gesehen, dass die Suchreihenfolge interessant ist. Vielleicht hat man schon eine Idee wie man suchen will. Im Beispiel aus der Vorlesung z.B. ob man im Stammbaum nach oben oder lieber nach unten suchen möchte. Ein anderes Beispiel war der Entwurf von Algorithmen. Und wir hatten auch den Cut-Operator kennen gelernt...

Was macht der denn?

Wenn wir etwas haben wie
$G : -T_1, ..., T_n, !, G_1, ..., G_m$ und die T_i nur als Tests dienen, nicht aber als Generator für Lösungen, dann wird das Ergebnis beim "!" festgeschrieben und als einzige Möglichkeit betrachtet.

Wie kann man denn damit das negation as failure realisieren?

$not(G) : -G, !, fail$ und $not(G)$.

Was ist denn daran nicht-monoton?

Also wenn wir keine Fakten zu $P(a, b)$ in der KB haben, dann könnten wir $\neg\exists x.P(x, b)$ folgern. Das geht halt nicht mehr, wenn wir $P(a, b)$ als Fakt einfügen.

Mal davon abgesehen, dass man in Prolog keine Anfragen dieser Form stellen kann, wars richtig. Wir haben aber noch andere Beschränkungen kennen gelernt. Z.B. Beschreibungslogiken. Wie ist das denn da?

Die Idee dahinter ist, dass man das Wissen über ein Objekt nicht mehr verstreut in der Wissensbasis hat, sondern zentral an einer Stelle sammelt - in einer Beschreibung. Die typischen Anfragen wie $C_1 \sqsubseteq C_2$ oder $i \rightarrow C$ sind wegen der eingeschränkten Sprache in $O(n^2)$ beantwortbar. Vorausgesetzt man lässt den $RESTR$-Operator weg, den wir ja zum Unterscheidung von FL und FL^- verwendet hatten.

Nur $RESTR$?

Also wenn wir NOT verwenden, dann klappt das strukturelle Matching auch nicht mehr. Wenn wir $AT\!-\!MOST$ erlauben ist es nicht mehr vollständig.

Zum Schluß noch kurz zur Vivid Knowledge. Was versteht man darunter?

Also wenn man über vollständiges Wissen verfügt, dann gibt es nur eine Interpretation, die die KB erfüllt. Die Idee dabei ist jetzt, diese Interpretation an die KB anzukleben, weil dann das Überprüfen von Folgerungen einfach wird. Wir brauchen dann eine KB mit CWA, DC und unique name assumption.

Gut. Dann gehen sie doch bitte mal kurz raus.