Prof. Dr. Stefan Brass
Institut für Informatik
MLU Halle-Wittenberg

13.06.2024

Logische Programmierung & deduktive Datenbanken — Übungsblatt 9 (Vier gewinnt) —

Ihre Lösungen zu Hausaufgabe 1 geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist Mittwoch, der 19. Juni, 16^{00} (die Aufgaben werden in der Übung am Donnerstag besprochen).

Hausaufgaben

Aufgabe 1 (Hausaufgabe, 10 Punkte)

„Vier gewinnt“ [https://de.wikipedia.org/wiki/Vier_gewinnt] wird wie folgt gespielt:

- Es gibt jeweils 21 gelbe und rote Spielmarken. Das Spiel wird von zwei Spielern gespielt, einer bekommt die gelben Spielmarken, einer die roten.
- Das Spielfeld hat sieben Spalten, die jeweils sechs Spielmarken fassen können. Es handelt sich also um eine 6×7-Matrix. Die Elemente können „gelb“, „rot“ oder leer sein.
- Die Spieler ziehen abwechselnd. Ein Spielzug besteht im Einwerfen einer Marke der eigenen Farbe in eine noch nicht komplett gefüllte Spalte. Sie fällt dann auf den untersten noch freien Platz in dieser Spalte.
- Falls ein Spieler vier Steine der eigenen Farbe horizontal, vertikal oder diagonal in einer Reihe hat, hat er gewonnen.
- Ist das Spielfeld voll, ohne dass vier Steine einer Farbe in einer Reihe stehen, endet es unentschieden.

Schreiben Sie ein Modul in SWI Prolog, das folgende Prädikate exportiert:

- `init_yellow(-State)`:
  Liefert den leeren Spiel-Zustand in einer von Ihnen gewählten Repräsentation, wenn Sie „gelb“ spielen („gelb“ fängt an).
- `choose_move_yellow(+State, -Col, -NewState)`:
  Dieses Prädikat wird mit einem Zustand `State` in Ihrer Repräsentation aufgerufen. Sie sollen einen Zug für den Spieler „gelb“ wählen, also eine Spalten-Nummer von 1 bis 7 liefern. Außerdem sollen Sie den Ergebnis-Zustand liefern (nach diesem Zug).
- `opponent_move_red(+State, +Col, -NewState)`:
  Dieses Prädikat wird aufgrufen, wenn Sie „gelb“ spielen, und der Gegner (also „rot“) am Zug war. Es informiert Sie über den Zug des Gegners. Sie müssen diesen in Ihre Zustands-Repräsentation einbauen.

Entsprechend gibt es die drei Prädikate auch in einer Variante, wenn Sie „rot“ spielen:

- `init_red(-State)`:
  Liefert den leeren Spiel-Zustand in einer von Ihnen gewählten Repräsentation, wenn Sie „rot“ spielen. Es wird dann anschließend `opponent_move_yellow` aufgerufen, um Ihnen den ersten Zug des Gegners mitzuteilen.
- `choose_move_red(+State, -Col, -NewState)`:
  Hier wählen Sie einen Zug für „rot“.
- `opponent_move_yellow(+State, +Col, -NewState)`:
  Dieses Prädikat wird aufgrufen, wenn Sie „rot“ spielen, und der Gegner (also „gelb“) am Zug war. Es informiert Sie über den Zug des Gegners.

Je nachdem, mit welcher Farbe Ihr Programm spielen soll, werden nur die einen oder die anderen drei Prädikate aufgerufen.

Sie finden ein Steuerprogramm für das Spiel, sowie ein Modul für einen menschlichen Spieler, ein Modul für einen ganz einfach zu schlagenden Gegner und ein Modul mit einer möglichen Implementierung von Spielzuständen unter den folgenden Adressen:

[http://www.informatik.uni-halle.de/~brass/lp24/prolog/connect4.pl]

[http://www.informatik.uni-halle.de/~brass/lp24/prolog/gamestate.pl]

[http://www.informatik.uni-halle.de/~brass/lp24/prolog/human.pl]

[http://www.informatik.uni-halle.de/~brass/lp24/prolog/trivial.pl]

Sie sind eingeladen, eine eigene Implementierung von Spielzuständen zu entwickeln (Sie müssen aber nicht). Das obige Modul „`gamestate`“ wird in jedem Fall gebraucht (für das Steuerprogramm, das gewissermaßen der Schiedsrichter ist). Nennen Sie Ihre Module möglichst so, dass es beim späteren Turnier keine Namenskonflikte gibt.
Natürlich darf Ihr Programm keine Spalte wählen, die schon voll ist, d.h. bei der in Zeile `6` schon ein Stein steht (Zeilen seien hier von unten gezählt, so dass der erste Stein in Zeile `1` landet).
Außerdem sollte wohl eine Mindestanforderung sein, dass, wenn ein Gewinn in einem Zug möglich ist, auch in die entsprechende Spalte gesetzt wird. Ansonsten können Sie sich aussuchen, wie „clever“ Ihr Programm ist. Sie könnten z.B. noch einen Zug vorausschauen, ob der Gegener dann gewinnen wird, und solche Züge soweit noch möglich vermeiden. Wenn Sie wollen, informieren Sie sich z.B. über den min-max-Algorithmus:

[https://de.wikipedia.org/wiki/Minimax-Algorithmus]

Achten Sie aber darauf, dass Sie nicht zu viel Rechenzeit verbrauchen. Ihr Programm wird disqualifiziert, wenn es für einen einzelnen Zug mehr als 3 Minuten Rechenzeit braucht, oder in der Summe über alle Züge mehr als 15 Minuten. Für ein Spiel Mensch gegen Computer wäre das schon viel zu lang. Die Programme werden auf einem Rechner mit zwei AMD EPYC 7281 16-Core Prozessoren (2.1 GHz) ausgeführt. Große vorberechnete Tabellen sind verboten (nur, was Sie mit der Hand tippen könnten, insgesamt also

max. 1000 Zeilen mit max. 80 Zeichen). Natürlich sind auch Netzwerk-Zugriffe ausgeschlossen.

Wenn es bei dem Spiel zu viele „Unentschieden“ geben sollte, könnte eventuell die verbrauchte CPU-Zeit zur Entscheidungsfindung herangezogen werden.

Wie allgemein üblich, sollten Sie Ihre Quellen offenlegen, wenn das Programm in wesentlichen Teilen nicht von Ihnen ist. Möglicherweise können Sie dann nicht am Turnier teilnehmen.

Das Turnier wird voraussichtlich in der vorletzten Übung (am 27.06.2024) durchgeführt. Sie hätten also die Gelegenheit, Ihr Programm nochmals zu verbessern. Einen ersten Test machen wir in der Übung in der kommenden Woche.

Zur Wiederholung

Aufgabe 2 (Wiederholungs-Fragen)

Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zum minimalen Modell und zum T_P-Operator antworten?

a) Definieren Sie den Begriff „Minimales Herbrandmodell“ eines logischen Programms P.

b) Gegeben sei das folgende logische Programm:

```
p(a).
q(b).
r(X) :- p(X).
```

Was ist das minimale Herbrandmodell dieses Programms? Geben Sie ein Beispiel für ein nicht-minimales Herbrandmodell dieses Programms. Warum gibt es nicht-minimale Modelle?

c) Hat jedes logische Programm (Menge definiter Hornklauseln) ein Herbrandmodell? Gibt es immer ein minimales Modell? Kann es auch mehr als ein minimales Herbrandmodell geben?

d) Wie kann man das minimale Herbrandmodell konstruieren?

e) Gegeben sei ein logisches Programm P. Definieren Sie den Operator T_P.

f) Was ist ein „Fixpunkt“ einer Abbildung, speziell des Operators T_P? Gibt es immer einen Fixpunkt? Geben Sie ein Beispiel für ein logisches Programm, für das der T_P-Operator mehr als einen Fixpunkt hat. Wie ist „kleinster Fixpunkt“ definiert? Gibt es immer einen kleinsten Fixpunkt? Ist dieser eindeutig? (Es wäre schön, wenn Sie den entsprechenden Satz wiedergeben könnten.)

g) Warum ist der kleinste Fixpunkt von T_P wichtig? Was ist der Zusammenhang zum minimalen Modell?

h) Wie kann man das minimale Modell mit dem T_P-Operator berechnen? Schreiben Sie die Iteration formal auf.

i) Angenommen, die Iteration des T_P-Operators terminiert nicht. Was kann man dennoch über die Beziehung zum minimalen Modell aussagen?

j) Was ist ein kausales Modell („supported model“)? Welche Beziehung gibt es zum minimalen Modell?

Übungsaufgaben (gemeinsam in der Übung zu bearbeiten)

Aufgabe 3 (Präsenzaufgabe)

Die folgende Aufgabe stammt aus der Klausur von 2021. Bei der Planung der Klausur wurde davon ausgegangen, dass die Teilnehmer ca. 15 Minuten zur Lösung benötigen (es gab aber noch 10 Minuten, die nicht verplant waren, und die Studierenden konnten ihre Zeit zwischen den Aufgaben beliebig verteilen). Es wurde „auf Papier“ programmiert, so dass keine Zeit zum Testen und Debuggen verbraucht wurde.

Ein Feuerwerker hat seinen Vorrat an Feuerwerksartikeln folgendermaßen als Prolog-Fakten erfasst:

```
artikel(malachit,
        [kauf(20180930, 5, 15.00),
         kauf(20201130, 5, 18.00),
         verbrauch(20201231, 2)]).
artikel(erzengel,
        [kauf(20190630, 4, 20.00),
         verbrauch(20191231, 2),
         kauf(20200630, 4, 26.00),
         verbrauch(20201231, 1)]).
artikel(eden,
        [kauf(20190630, 4, 12.00),
         verbrauch(20191231, 4)]).
```

Für jeden Artikel wird also eine Liste von Lager-Bewegungen gespeichert:

- `kauf(Datum, Anzahl, Stueckpreis)`,
  (also ein Lager-Zugang durch Kauf einer gewissen Stückzahl zu einem Preis),
- `verbrauch(Datum, Anzahl)`
  (also ein Abgang aus dem Lager durch Verbrauch bei einem Feuerwerk).

Die Listen der Lager-Bewegungen sind chronologisch geordnet (nach dem Datum). Sie können sich darauf verlassen, dass jede Liste mit einer Kauf-Transaktion beginnt (Artikel mit leeren Listen brauchen nicht berücksichtigt zu werden, und ein Verbrauch vor einem Kauf wäre unmöglich). Außerdem sind natürlich alle Stückzahlen und Preise positiv.

Definieren Sie ein Prädikat `vorrat(Name, Anzahl, Stueckpreis)`, das zu jedem Artikel die nach allen Lagerbewegungen aktuell noch vorhandene Stückzahl und den Stückpreis für diese Artikel liefert. Im Beispiel wären die Ergebnisse:

Name	Anzahl	Stueckpreis
malachit	8	16.5
erzengel	5	24.0
eden	0	12.0

Wenn vor einem Kauf noch n Stück zum Preis p vorhanden waren, und dann m Stück zum Preis q gekauft werden, ist der neue Preis das gewichtete Mittel:

$$(n * p + m * q)/(n + m).$$

Z.B. sind vor dem zweiten Kauf der Batterie „Erzengel“ noch zwei Stück zum Preis von jeweils 20 € vorhanden. Es werden dann vier Stück zum höheren Preis von 26 € gekauft. Damit sind insgesamt sechs Stück zum Gesamtpreis von $2*20+4*26 = 144$ € vorhanden, der neue Stückpreis ist also $144/6 = 24$ €.

Sie können nur die im Skript genannten eingebauten Prädikate verwenden. Sie können sich selbstverständlich beliebige Hilfsprädikate in Prolog definieren.

Zum Selbstudium

Aufgabe 4 (Zusatzaufgabe)

Schauen Sie sich die folgenden Webseiten an. Sie brauchen für diese Aufgabe nichts abzugeben. Ziel ist, dass Sie einen Eindruck davon gewinnen, was es im Internet zum Thema dieser Vorlesung gibt, und dabei für sich nützliche Quellen entdecken.

Dieses Mal möchte ich auf einige Arbeiten meiner Arbeitsgruppe zur effizienten Auswertung von Datalog und zum Leistungsvergleich hinweisen. Sie sind natürlich eingeladen, daran mitzuarbeiten.

a) Brass, Stefan and Stephan, Heike: Experiences with Some Benchmarks for Deductive Databases and Implementations of Bottom-Up Evaluation.
30th Workshop on (Constraint) Logic Programming (WLP 2016).

   [https://www.imn.htwk-leipzig.de/~schwarz/wlp16/program.html]

   [https://.../wlp16/wlp2016_pre-proceedings_paper_3.pdf]

   [http://eptcs.web.cse.unsw.edu.au/paper.cgi?WFLP2016.5]

b) Brass, Stefan and Wenzel, Mario: Performance Analysis and Comparison of Deductive Systems and SQL Databases. Datalog 2.0 2019 — 3rd International Workshop on the Resurgence of Datalog in Academia and Industry (2019).

   [http://ceur-ws.org/Vol-2368/]

   [http://ceur-ws.org/Vol-2368/paper3.pdf]

c) Dazu gibt es im Cantor-Heft 23 (2021) noch eine etwas ausführlichere Version auf Deutsch („Analyse der Laufzeiten verschiedener Datenbanksysteme für rekursive Anfragen“):

   [https://wcms.itz.uni-halle.de/download.php?down=62329&elem=3434418]

   Alle Cantor-Hefte finden Sie hier:

   [https://www.cantor-vereinigung.uni-halle.de/cantor-hefte/]

d) Webseite zum Projekt:

   [https://dbs.informatik.uni-halle.de/rbench/]

e) Wir waren inspiriert vom OpenRuleBench:

   [https://www3.cs.stonybrook.edu/~pfodor/openrulebench_web/]

Hier noch zwei Artikel zu der „Push“-Auswertungsmethode:

f) Brass, Stefan and Stephan, Heike (2018): Pipelined Bottom-Up Evaluation of Datalog Programs: The Push Method. In: Perspectives of System Informatics — PSI 2017, Springer, LNCS 10742, 43-58, 2018.

[https://users.informatik.uni-halle.de/~brass/push/publ/psi17.pdf]

[https://users.informatik.uni-halle.de/~brass/push/publ/]

g) Brass, Stefan and Wenzel, Mario (2018): An Abstract Machine for Push Bottom-Up Evaluation of Datalog. In: Database and Expert Systems Applications. DEXA 2018. Springer, LNCS 11030, 270–280.

[https://users.informatik.uni-halle.de/~brass/push/publ/dexa18.pdf]

[https://link.springer.com/chapter/10.1007/978-3-319-98812-2_23]