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

17.06.2022

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

Ihre Lösungen zu den Hausaufgaben a) geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist der 23. Juni, 10^{00} (die Aufgaben werden anschließend in der Übung besprochen).

Hausaufgaben

a) „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 folgenden Adressen:

[http://www.informatik.uni-halle.de/~brass/lp22/prolog/connect4.pl]

[http://www.informatik.uni-halle.de/~brass/lp22/prolog/gamestate.pl]

[http://www.informatik.uni-halle.de/~brass/lp22/prolog/human.pl]

[http://www.informatik.uni-halle.de/~brass/lp22/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 seinen 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 07.07.2022) 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

b) Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zu Datalog mit eingebauten Prädikaten antworten?

- Nennen Sie Unterschiede von Datalog zu Prolog.
- Nennen Sie einige Varianten von Datalog, d.h. welche Konstrukte könnte man zu Basis-Datalog noch hinzutun? (Ein oder zwei Möglichkeiten reichen.)
- Warum denken Prolog-Programmierer und Datalog-Programmierer anders über definite Hornklauseln?
- Welche Regeln sind „allowed“? Geben Sie zunächst die erste/einfachste Variante der Definition wieder. Warum ist diese Eigenschaft für die praktische Bottom-Up-Auswertung mit dem T_P-Operator wichtig? Geben Sie ein Beispiel für eine Regel, die nicht „allowed“ ist, und erläutern Sie, welches Problem es bei der Berechnung des minimalen Modells geben würde.

  Hinweis: Beachten Sie aber, dass der Satz über den Zusammenhang des minimalen Modells mit dem kleinsten Fixpunkt des T_P-Operators ohne die Einschränkung auf „allowed“-Regeln gilt. Diese Eigenschaft ist nur für die praktische Berechnung durch Anwendung des Satzes wichtig.
- Wenn man auch eingebaute Prädikate im Regelrumpf erlaubt, z.B. „<“, warum ist die einfache Definition zulässiger Regeln („allowed“, s.o.) nicht ausreichend? Was wäre eine einfache Korrektur? Warum ist diese Lösung doch sehr eingeschränkt? Würde es für einfache SQL-Anfragen ausreichen?
- Für die entgültige Definition zulässiger Regeln („range-restricted rule given a binding pattern β for the head“) haben wir angenommen, dass eine Menge erlaubter Bindungsmuster $valid(p)$ für jedes Prädikat p definiert sind. Was ist dann

die Intuition der Definition einer bereichsbeschränkten Regel? Warum braucht man für die Bottom-Up-Auswertung mit dem T_P-Operator „stark bereichsbeschränkte“ Regeln? Was ist da der Unterschied?

- Warum braucht man in Datalog keine volle Unifikation, sondern nur einfaches „Matching“? Was ist der Unterschied?
- Wie kann man Funktionssymbole von Prolog mit eingebauten Prädikaten in einer deduktiven Datenbank simulieren? Erläutern Sie das am Beispiel von Listen. Was wäre die wesentliche Einschränkung gegenüber Prolog? Wenn man in der deduktiven Datenbankbank auch eine funktionale Notation erlauben würde, was könnte man damit gegenüber Prolog gewinnen?

Zum Selbstudium

c) 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.

- 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]

- 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]

- Webseite zum Projekt:

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

- Wir waren inspiriert vom OpenRuleBench:

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

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

- 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/]

- 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]