Künstliche Intelligenz
Vorlesung
SS 18

Johannes Waldmann

3. Juli 2018

Einführung — Definition

- was ist KI: maschinelles Nachbilden von (vermuteten) menschlichen Aufgabenlösungsverfahren
- welche Aspekte/Methoden werden betrachtet: u.a.
  - Repräsentation von Wissen
    (z.B. durch Algorithmen, Muster, Regeln, Parameter für Algorithmen)
  - Extraktion von Wissen (aus Rohdaten)
  - Wissensverarbeitung
    (z.B. Anwenden von Regeln, Hinzufügen von Regeln)
- warum macht man das:
  - um zu verstehen, wie der Mensch Aufgaben löst
  - um andere maschinelle Verfahren zu übertreffen

Die Definition von *Intelligenz*?

- gibt es einen Unterschied zwischen den Aussagen:

  ein technisches System . . .

  – zeigt intelligentes Verhalten,
  – *ist* intelligent?

- für Philosphen: möglicherweise, für Praktiker: nein.

  E. W. Dijkstra (EWD 898, 1984): The question of whether Machines Can Think . . . is about as relevant as the question of whether Submarines Can Swim.

  (Interpretation: U-Boote können sich im Wasser fortbewegen. Die Ingenier-Aufgabe ist, Effizienz (. . .) der Bewegung zu erhöhen — egal, wie diese heißt.

Die derzeitige „Definition“ von KI

. . . kann aus dieser Meldung entnommen werden:

> . . . Already, China has one of the biggest clusters of AI scientists. It has over 800m internet users, more than any other country, which means *more data on which to hone its new AI*.

(The Economist 17. 3. 2018, *America vs. China*)

KI als Mode-Erscheinung

- KI = maschinelles Lernen = „deep learning“ = das Bestimmen von Koeffizienten für konvolutionale neuronale Netze mit mehreren Schichten durch Gradientenabstieg
- mit der im wesentlichen einzigen Anwendung:

  das Ausforschen von Daten der sogenannten Benutzer der sogenannen sozialen Netzwerke

  mit dem Ziel, ihr Verhalten vorherzusagen und zu manipulieren, um den Werbekunden mehr „Benutzer“-Aufmerksamkeit (Zeit) zu verkaufen
- das aber pseudo-wissenschaftlich bis religiös überhöht (um möglichst viele GPUs zu verkaufen)

Das ist nicht die erste KI-Modewelle

- Fifth Generation Computer Systems (staatliche Forschungsförderung in Japan 1982 – 1993)

  Ziel: Soft- und Hardware für massiv parallele Wissensverarbeitung

  `https://web.archive.org/web/20090217105259/http://www.icot.or.jp/ARCHIVE/Museum/ICOT/FGCS-E.html`

- und ähnliche Projekte in USA und Europa

  (wegen FOMO - fear of missing out)

Sachliche Grundlagen der KI

- formale Logik
  (Aristoteles, Leibniz, Frege, Russel, Gödel, Turing, . . .)
  - Syntax (Formeln)
  - Semantik (Wahrheit, Modelle)
  - Kalkül (Axiome und Regeln zum Schließen)
- künstliche neuronale Netze (Warren McCulloch, Walter Pitts, 1943)
- symbolische Informationsdarstellung (d.h. Daten sind Term-Bäume) und -verarbeitung in Programmiersprache LISP (John McCarthy, 1958)
- Resolutions-Kalkül als operationale Semantik in Programmiersprache Prolog (Alain Colmerauer, 1972)

Übersicht nach Kalenderwochen

- KW 14: Einführung
- KW 15: Suche/Planung (Einpersonenspiele) Bsp: Sudoku, Lunar Lockout, Sokoban
- KW 16: blinde Suche (DFS, BFS), heurist. Suche (Greedy, A^*)
- KW 17: FD-Constraints, Propagieren, Entscheiden
- KW 18: Suche (Zweipersonenspiele) Bsp: Nim (exakte Lösung), Phutball, Go
- KW 19: (Feiertage)
- KW 20: Spielbaum, Pruning (Alpha-Beta) (Schach, Phutball)

- KW 21: Monte Carlo Tree Search (Go)
- KW 22: Regeln (Bsp: Regeln in uMatrix, Regeln in CSS)
- KW 23: Entscheidungsbäume, -Diagramme, BDD
- KW 24: Unifikation, Resolution, Prolog
- KW 25: Semantik f. erweiterte logische Prog.
- KW 26: Neuronale Netze (Bsp: Mustererkennung, AlphaGoZero)
- KW 27: Zusammenfassung

Organisatorisches

- pro Woche ein VL, eine Ü
- VL-Skript auf Webseite (nach und nach) `https://www.imn.htwk-leipzig.de/~waldmann/lehre.html`
- Ü-Aufgaben:
  - (in Gruppen) Vorrechnen an der Tafel bzw. am Computer
  - (individuell) autotool
- Klausur (120 min, keine Hilfsmittel)

  Zulassung: 50 Prozent der Ü-Punkte (jeweils)

Beispiele für Aufgaben der KI

- wir benutzen oft tatsächlich *Spiele*,
- Einpersonenspiele (*puzzle*)
  - ergänze unvollständige Information (Sudoku, Minesweeper)
    (Spielesteine/Zahlen hinzufügen)
  - finde Zugfolge zu einem bestimmten Zustand (Lunar Lockout, Sokoban) (Spielsteine bewegen)
- Zweipersonenspiele (*game*) (Nim, Gomoku, Go)
  - erzwinge das Erreichen eines bestimmten Zustandes trotz gegnerischer Züge

Sudoku

- seit 1895 mehrfach wiedererfunden
- Spezifikation:
  - Indizes $I = \{1, 2, 3\}$, Positionen $P = I^4$, Farben $F = I^2$
  - Aufgabenstellung (Instanz): partielle Abb. $a : P \hookrightarrow F$
  - Lösung: (totale) Funktion $s : P \to F$ mit $a \subseteq s$ und

$$\begin{aligned} &\forall (i,j) \in I^2 : \{s(i,j,k,l) \mid (k,l) \in I^2\} = I^2 \\ \wedge\ &\forall (k,l) \in I^2 : \{s(i,j,k,l) \mid (i,j) \in I^2\} = I^2 \\ \wedge\ &\forall (i,k) \in I^2 : \{s(i,j,k,l) \mid (j,l) \in I^2\} = I^2 \end{aligned}$$

  vier-dimensional! Zahlen $1 \ldots 9$ erschweren die Spezifikation unnötig (benötigt dann Division mit Rest)
- effizientes Lösungsverfahren?
- Erzeugung interessanter/schwerer Instanzen?

Lunar Lockout

- Erfinder: Hiroshi Yamamoto, $\leqslant 2000$, Binary Arts
- Spezifikation:
  - endl. Menge R von Robotern, P von Positionen
  - Ziel $\in R \times P$, Konfigurationen: $C = (R \to P)$
  - Konf. c heißt *gelöst* bzgl. Ziel (r, z), falls $c(r) = z$.
  - Zug: $c_1 \stackrel{r,d}{\to} c_2$ mit $c_1, c_2 \in C, r \in R, d \in$ Richtungen, so daß
    - $\forall s \in R \backslash \{r\} : c_1(s) = c_2(s)$
    - $c_2(r)$ liegt von $c_1(r)$ aus in Richtung d
    - die Position in Richtung d hinter $c_2(r)$ ist belegt
    - und ...
- effizientes Lösungsverfahren?
- Erzeugung schwerer Aufgaben-Instanzen?

Nim

- endliches 2-Personenspiel mit vollständiger Information
- Konfiguration: Multimenge M von natürlichen Zahlen
- Zug: ein $x \in M$ durch ein y mit $0 \leqslant y < x$ ersetzen
- verloren hat, wer nicht mehr ziehen kann
- ist *neutrales* Spiel: mögliche Züge hängen nicht vom Spieler ab. Gegensatz: z.B. Schach, der erste Spieler darf nur weiße Figuren führen.
- Nim hat vollständige mathematische Beschreibung, Status und optimaler nächster Zug können ohne Spielbaum bestimmt werden

Go

- Go (japanisch), Weiqi (chinesisch), Baduk (koreanisch)
- eines der ältesten und verbreitetsten Brettspiele
- Spezifikation:
  - Konfiguration: $\{1 \ldots 19\}^2 \rightarrow \{$weiß,schwarz,leer$\}$
  - Zug: Stein eigener Farbe setzen, evtl. gegnerische Steine ohne *Freiheiten* (Gefangene) entfernen
  - Ziel: möglichst großes beherrschtes *Gebiet* (leere Felder, auf welche der Gegner nicht setzen möchte, da er dort gefangen würde)
- Testfall für KI, nachdem Schach prinzipiell gelöst war.
- Seinsei's Library `https://senseis.xmp.net/`
  Deutscher Go-Bund `http://dgob.de/`

Übung KW14

1. Sokoban

   - ausprobieren: z.B. `https://sokoban.info/`
   - formale Spezifikation
   - Eine Startkonfiguration s enthält f Felder, darauf stehen k Kisten. Geben Sie eine möglichst gute obere Schranke für die Anzahl der von s aus erreichbaren Konfigurationen an (die keine weiteren Informationen wie Form des Feldes, Lage der Kisten benutzt)

2. Gomoku

   - Ziel: 5 in einer Reihe (waagerecht, senkrecht, diagonal)
   - (Brett und Steine wie Go, aber sonst keine Beziehung)
   - Gewinnen Sie gegen M-x gomoku?

(gesprochen: meta x, getippt: ESC dann x)
in Emacs (vgl. `https://xkcd.com/378/`)

- Diese „KI“ verwendet *keine* Spielbaumsuche, sonder nur eine einfache Heuristik zur Stellungsbewertung. Suchen Sie den Quelltext. In welcher Programmiersprache?

3. unterhalten Sie sich mit Eliza (M-x doctor).

   (Wann, von wem,) Warum wurde dieses Programm ursprünglich geschrieben?

   Falls Sie dazu Wikipedia benutzen: welche Information aus dem Anfang des englischen Textes ist im deutschen viel weiter hinten versteckt?

Aufgaben (Diskussion in KW 15)

1. Wolkenkratzer `https://www.janko.at/Raetsel/Wolkenkratzer/index.htm`
   - Lösen Sie (auf dem Papier) eine der dort als *schwer* bezeichneten Instanzen, beschreiben Sie das Vorgehen.
   - Geben Sie eine formale Spezifikation an.
2. Lunar Lockout (Teil A)
   - Lösen Sie Ihre autotool-Aufgabe, beschreiben Sie das Vorgehen
   - ergänzen Sie die Spezifikation aus dem Skript
3. Lunar Lockout (Teil B). Wir betrachten den gerichteten Graph G aller Konfigurationen (jeder Zug ist eine Kante),

die von einer Startkonfiguration mit k Robotern aus erreichbar sind.

- Ist G kreisfrei?
- Geben Sie eine Konfiguration mit $k \geqslant 4$ an, die in G keinen Vorgänger hat.

4. Go

- Lösen und erklären Sie eine der Aufgaben von `http://www.goproblems.com/` (für 30 . . . 20 kyu)
- Zeigen Sie eine Beispiel-Aufgabe mit einer Treppe (engl. *ladder*). Was bedeutet die Existenz von Treppen für die (automatisierte) Bewertung von Spielsituationen durch lokale Mustererkennung?

Such-Aufgaben und -Verfahren

Varianten

- Planungs-Aufgaben (z.B. Sokoban):
  Graph ist gegeben (Spiel-Konfiguration, -Züge),
  gesucht ist ein (kürzester) Weg von der
  Start-Konfiguration zu (einer) gelösten Konfiguration
- Constraint-Aufgaben (z.B. Sudoku):
  Graph beschreibt Teilschritte in Lösungsverfahren
  z.B. $C = (\text{Pos} \hookrightarrow \text{Farbe}), c_1 \to c_2$ für Färben *einer* Position
  gesucht ist eine gelöste Konfiguration (Weg ist egal)
- Planungs-Aufgabe als Constraint-Aufgabe:
  jede Konfiguration enthält den/einen Weg dorthin

Unterschied zu bekannten Graph-Aufgaben

z.B. single-source-shortest-paths
(kürzeste Wege von einem Knoten zu allen anderen),
gelöst durch Dijkstra-Algorithmus

- das ist eine andere Aufgabe (aber man könnte aufhören, sobald eine gelöste Konfiguration erreicht wird)
- Graph liegt nicht explizit vor (alle Knoten/Kanten), sondern implizit (Funktion z. Erzeugung der Nachbarn)

  für „alle Nachbarn von p besuchen“
  vorher „die unbekannten Nachbarn von p erzeugen“
- Graph soll gar nicht komplett erzeugt werden, sondern nur erfolgversprechende Knoten (Konfigurationen)

Eingaben für Such-Aufgabe

- Zustands/Konfigurarionsmenge C, Aktionsmenge A
- Nachfolgerfunktion $n : C \to (A \hookrightarrow C)$

  definiert gerichteten Graphen G über C mit Kantenbeschriftung aus A
- Startzustand $s \in C$
- Zielmenge $T \subseteq C$ oder Zielprädikat $C \to$ Bool
- Pfadkostenfunktion, Spezialfälle:
  - Pfadkosten $=$ Summe über Schrittkosten, Schrittkostenfunktion $w : C \times A \times C \to \mathbb{N}$
  - Pfadkosten hängen nur vom Ziel des Pfades ab

Beispiele für Such-Aufgaben

- Sokoban
- Schiebefax
- Quelltexte siehe `https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss18`

Aufgabenstellung in Such-Aufgabe

- finde *eine* Lösung

  d.h., einen Knoten $t \in T$ mit $s \rightarrow_G^* t$

  oder die Antwort, daß kein solcher existiert

  jedes Suchverfahren mit dieser Eigenschaft heißt *vollständig*

- finde eine *optimale* Lösung,

  d.h., einen Pfad $s \rightarrow_G^* t$, der die Pfadkosten minimiert

  oder die Antwort, daß Aufgabe unlösbar ist

  jedes Suchverfahren mit dieser Eigenschaft heißt *optimal*

Graph- und Baumsuche

- Eingabe lt. Spezifikation, Ausgabe: Folge von Knoten
- Knotenmengen: in Bearbeitung B, erledigt E, unbek. U
- `go (E, B) = wenn B leer, dann fertig, sonst`

```
  wähle x aus B;
  wenn x not in E, // nur bei Graphsuche
  dann print(x)
       go (E mit x, B ohne x mit n(x))
  sonst go (E, B ohne x)
```

  Aufruf mit `go (empty, {s})`, Rechnung bis `x in T`
- Baumsuche: ohne den markierten Test
- B als Stack: Tiefens. (DFS), als Queue: Breitens. (BFS)
- Kosten werden ignoriert (*blinde* Suche)

Informierte Suche

- Ziel: besseres Suchen durch mehr Wissen

  (ausgedrückt als *Heuristik* $h : C \to \mathbb{N}$ zur Kostenschätzung)

- Umsetzung: im Standard-Suchverfahren,

  verwende Prioritätswarteschlange für B,

  wähle jeweils ein x mit kleinster Priorität

- Priorität von x ist Summe von
  - (evtl.) Pfadkosten von $s \to^* x$
  - Schätzung $h(x)$

  ohne Pfadkosten: *greedy*-Suche, mit: A^*-Suche

Eigenschaften von Schätzfunktionen

- Bezeichnung: $\text{opt}(x) := \min\{c(p) \mid t \in T, p : x \to^* t\}$
- *perfekt*: $\forall x : h(x) = \text{opt}(x)$

  falls von x kein Ziel erreichbar, dann $h(x) = \min \varnothing = +\infty$

  deswegen Erweiterung des Wertebereiches:
  $h : C \to \mathbb{N} \cup \{+\infty\}$
- *zielerkennend*: $\forall t \in T : h(t) = 0$
- *nicht überschätzend* (in Russel/Norvig: *admissible*)

  $\forall x : h(x) \leqslant \text{opt}(x)$
- *monoton*: (in Russel/Norvig: *consistent*)

  $\forall x, a, y : n(x) \ni (a, y) \Rightarrow h(x) \leqslant c(x, a, y) + h(y)$

Übungen KW 15

- wir betrachten als Schätzfunktion für Sokoban:

  Anzahl der Kisten, die noch nicht im Ziel sind.

1. ist nicht überschätzend?
2. ist monoton?
3. Lösen Sie die Instanz `28_11`

- Schiebefax (vgl. `https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss18/blob/master/kw15/Fifteen.hs`)

1. Warum werden in einer Konfiguration *beide* Abbildungen `M.Map Pos Item` und `M.Map Item Pos` abgespeichert?

2. Die Implementierung realisiert quadratische Spielfelder, welche Änderungen sind für beliebige Rechtecke nötig?
3. Geben Sie ein mathematisches Modell für Schiebefax auf beliebigen Graphen an.
4. die Schätzfunktionen (siehe Hausaufgabe) sind: nicht überschätzend? monoton?

Hausaufgaben (für KW 16)

vgl. Serie 1 von `http://www.imn.htwk-leipzig.de/~schwarz/lehre/ss17/ki/`

1. Wegsuche (Aufgabe 1.2)

   (a) Tiefen- und Breitensuche
   (b) Greedy- und A^*-Suche mit Manhattan-Abstand als Heuristik
   (c) geben Sie eine Landkarte (Gitter mit Wänden, wie in Aufgabenstellung) sowie Start- und Zielpunkt an, so daß die Suche nach dieser Heuristik deutlich länger dauert als blinde Breitensuche.

2. Wegsuche (vgl. Aufgabe 1.2)

   (a) (1.2.4.) modellieren Sie die Kosten des Abbiegens

dadurch, daß Sie in jeder Konfiguration zusätzlich auch die Blickrichtung speichern. Ein „Abbiegen“ besteht dann aus zwei Zügen: Drehen (am Ort) und Gehen.

(b) Geben Sie eine Landkarte mit Start- und Zielpunkt sowie darauf einen Weg an, der mit Abbiegekosten optimal ist, aber mit Standardkosten nicht.

3. Schiebefax — aber mit Feldgröße 2×3.

Die gelöste Konfiguration ist $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & - \end{pmatrix}$.

(a) Aufgabe 1.1.1 für 4 Züge (statt 7)

(b) Aufgabe 1.1.2 (für die Zugfolge aus voriger Teilaufgabe)

(c) für die Startkonfiguration aus voriger Teilaufgabe und für die Heuristik „Summe der Abstände“: zeigen Sie jeweils die ersten 4 Schritte der Greedy-Baumsuche,

der A^*-Baumsuche.

4. Über eine Such-Aufgabe ist bekannt:

   - es gibt N von s aus erreichbare Konfigurationen (Z.B. für Schiebefax auf 3×3 ist $N = 181440$).
   - in jeder Konfiguration sind höchstens B verschiedene Züge möglich (Schiebefax: $B = ?$)

   Bestimmen Sie daraus eine möglichst große Zahl K, so daß gilt: es gibt eine Konfiguration t, die von s aus nicht in $< K$ Zügen erreichbar ist.

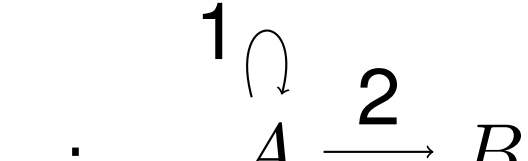
   Hinweis: bestimmen Sie eine obere Schranke für die Anzahl der in $< K$ Schritten erreichbaren Konfigurationen.

BFS/DFS: Vollständigkeit

- Satz: jede Graphsuche (DFS und BFS) ist vollständig für endliche Graphen.

  Beweis: jedes dieser Verfahren besucht *alle* Knoten des Graphen, die vom Startpunkt s erreichbar sind. (VL A& D)
- Satz: BFS (Graph und Baum) ist vollständig für unendliche Graphen mit beschränktem Ausgangsgrad (z.B. wg. endlicher Anzahl von Aktionen)

  Beweis: für jedes k ist $L(k) := \{x \mid$ es ex. Pfad der Länge k von s zu $x\ \}$ endlich. Falls $\exists t \in T$, dann $k :=$ dist(s, t). BFS zählt $L(0), L(1), \ldots, L(k)$ auf, evtl. mit Wdhlg.
- Satz: DFS-Baum ist nicht vollständig. Beweis:



BFS/DFS: Optimalität

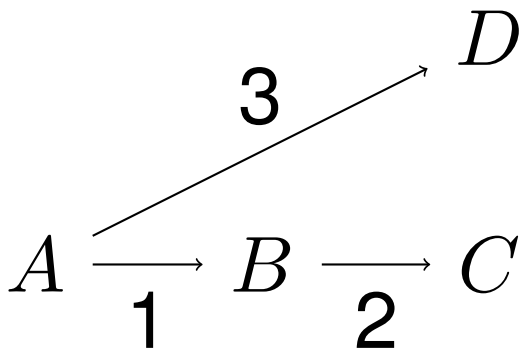
für Einheitskosten: jede Kante hat Gewicht 1

- Satz: BFS (Graph und Baum) ist optimal.

  Beweis: wie eben für $k = \min\{\text{dist}(s,t) \mid t \in T\}$

- Satz: DFS (Graph und Baum) ist nicht optimal.

  Beweis:



BFS/DFS: Kosten

- BFS ist besser als DFS?
- vollständig, optimal, aber oft auch teuer:

  Falls $C =$ vollst. binärer Baum der Höhe h,
  Start $=$ Wurzel, Ziel $=$ ein Blatt ganz rechts, dann

  – DFS: Keller hat Tiefe h
  – BFS: Warteschlange hat Größe 2^h (jede Schicht wird komplett abgespeichert)

- Ü: falls linkes Kind immer vor rechtem behandelt wird:

  – im o.g. Beispiel braucht DFS und BFS 2^h Zeit
  – wo müßte das Ziel liegen, damit
  DFS viel schneller als BFS? Umgekehrt?

Invariante der informierten Suche

- Satz: für Graph/Baum-Greedy/A^* gilt die Invariante:

  Für alle noch nicht besuchten Knoten y:

  jeder Weg von s zu y führt durch einen Knoten in Q ($=$ Menge der Knoten in der Warteschlange)

- Beweis: wenn Knoten q besucht, d.h., aus Q entfernt wird:

  werden alle seine Nachfolger $q_1, \ldots, q_k$ hinzugefügt.

  Jeder Pfad durch q wird dann ein Pfad durch ein q_i.

- Folgerung: falls das Suchverfahren erfolglos hält,

  dann wurden alle von s erreichbaren Knoten besucht.

- Folgerung: Graphsuche ist vollständig (unabh. von h)

Eigenschaften von Heuristiken

(Gegen)beispiele

(alle für $s = x_0 \to x_1 \to x_2 \to x_3 = t \in T$ mit Einheitskosten)

- perfekt:
- nicht zielerkennend:
- zielerkennend, aber nicht perfekt:
- nichtüberschätzend, aber nicht perfekt:
- nicht monoton:
- monoton, aber nicht perfekt:
- monoton, aber nicht zielerkennend:
- nicht monoton, aber nichtüberschätzend:

Eigenschaften der perfekten Heuristik

- Satz: h perfekt $\Rightarrow$ h zielerkennend $\wedge$ h monoton $\wedge$ h nicht überschätzend.
- Beweis:
  - ziel-erkennend: ...
  - nicht überschätzend: ...
  - monoton: folgt aus
    $\mathsf{opt}(x) = \min\{c(x, a, y) + \mathsf{opt}(y) \mid y \in C, x \xrightarrow{a} y\}$.

Monoton und zielerkennend $\Rightarrow$ nicht überschätzend

- Satz: h monoton $\wedge$ h zielerkennend $\Rightarrow$ h nicht überschätzend.
- Beweis: zu zeigen ist $\forall x : h(x) \leqslant \text{opt}(x)$.
- Plan: Induktion über Länge eines optimalen Pfades von x zu Ziel
  - Fall 1: es gibt keinen solchen Pfad
  - Fall 2: Pfad existiert und Länge l
    - I.-Anfang
    - I.-Schritt

Invariante der A^*-Suche

- Satz: h nicht überschätzend $\Rightarrow$

  für alle $x \in T$, die noch nicht besucht wurden:

  es gibt $q \in Q$ mit pri$(q) \leqslant$ dist(s, x).

- Beweis:

  wähle q auf optimalem Weg von s zu x

  (das ist möglich wg. Invariante)

  dist$(s, x) =$ dist$(s, q) +$ dist$(q, x) \geqslant$ dist$(s, q) + h(q) =$ pri(q).

Nicht überschätzend $\Rightarrow$ A^*-Baum ist vollst.

- Satz: h nicht überschätzend $\Rightarrow$ A^*-Baumsuche ist vollständig.
- Beweis: zu widerlegen ist:

  die Suche findet überhaupt keine Lösung, obwohl $T \neq \emptyset$.

  Sei $x \in T$. Aus Invariante folgt: die niedrigste Priorität in Q ist $\leqslant$ dist(s, x).

  Es gibt nur endlich viele Pfade $s \rightarrow^* y$ mit pri$(y) \leqslant$ dist(s, x)

  Deswegen wird x irgendwann besucht (oder schon vorher ein anderes $y \in T$)

Nicht überschätzend $\Rightarrow$ A^*-Baum ist optimal

- Satz: h nicht überschätzend $\Rightarrow$ A^*-Baumsuche ist optimal.
- Beweis: zu widerlegen ist:

  die Suche findet eine nicht optimale Lösung $y \in T$,

  d.h., es ex. $x \in T$ mit dist$(s, x) <$ dist(s, y).

  als y besucht (d.h., aus Q entfernt) wurde, gab es ein $q \in Q$ mit pri$(q) \leqslant$ dist(s, x).

  das widerspricht pri$(y) =$ dist(s, y)

Monotonie

- Lemma: h monoton $\Rightarrow$ wenn q entfernt und Nachfolger q_i hinzugefügt, dann pri$(q) \leqslant$ pri(q_i).
- Beweis: pri$(q_i) =$ dist$(s, q_i) + h(q_i) =$ dist$(s, q) + c(q \rightarrow q_i) + h(q_i) \geqslant$ dist$(s, q) + h(q) =$ pri(q).
- Satz: h monoton und ziel-erkennend $\Rightarrow$ A^*-Graphsuche ist optimal.
- Beweis: wg. Lemma werden die Knoten in schwach steigender Priorität besucht (und expandiert).

  Der erste besuchte $x \in T$ ist optimal, denn alle anderen $y \in T$ haben pri$(y) \geqslant$ pri$(x) =$ dist(s, x).

Schwere Such-Aufgaben

- es gibt Suchaufgaben (mit Parameter n), für die jede kürzeste Lösung exponentiell (in n) lang ist.
- das ist einfach zu bestätigen für *Türme von Hanoi*
- schwieriger für Sokoban (und ähnliche Verschieberätsel)

  Robert A. Hearn, Erik D. Demaine:
  PSPACE-completeness of sliding-block puzzles . . .,

  TCS 343(1-2): 72-96(2005), `http://erikdemaine.org/papers/NCL_TCS/paper.pdf`
- (meiner Kenntnis nach) ungeklärt für Lunar Lockout.

  $\Rightarrow$ genauere Untersuchung gern als Bachelorarbeit

3 Türme von Hanoi

- Konfiguration: endliches Wort über $\Sigma = \{A, B, C\}$.

  ($w_i = t$ bedeutet „die Scheibe i liegt auf Turm t“)
- Start: A^n, Ziel: C^n,
- Zug: $pxq \to pyq$ für $x, y \in \Sigma, p \in (\Sigma \backslash \{x, y\})^*, q \in \Sigma^*$
- eine kürzeste Lösung für $n = 2$: $AA \to BA \to BC \to CC$.
- Satz: jede Lösung benötigt $\geqslant 2^n - 1$ Züge.
- Beweis durch Induktion. Ist klar für $n = 0$. Für $n > 0$:

  Scheibe n muß wenigstens einmal bewegt werden.

  Es gibt also einen Zug $pA \to pX$ mit $X \neq A$ sowie einen (evtl. denselben) Zug $pY \to pC$ mit $Y \neq C$

3 Türme von Hanoi (Beweis)

- Es gibt einen Zug $pA \to pX$ mit $X \neq A$...
- Dann ist $p = Z^{n-1}$, wobei $\{A, X, Z\} = \Sigma$.
- Davor muß also $A^n = A^{n-1}A \to^* Z^{n-1}A$ gelöst sein, ohne das rechte A zu bewegen.

  Nach Induktion brauchen wir für $A^{n-1} \to^* Z^{n-1}$ wenigstens $2^{n-1} - 1$ Züge.
- Ebenso für $Z^{n-1}C \to^* C^{n-1}C = C^n$.
- insgesamt $\geqslant 2^{n-1} - 1 + 1 + 2^{n-1} - 1 = 2^n - 1$.
- Lösung ist optimal, wenn Scheibe n wirklich nur einmal bewegt wird. Dann genau $2^n - 1$ Züge.

Grenzen einfacher Suchverfahren

- für Türme von Hanoi (und ähnliche) gibt es keine einfache (kleiner Wertebereich) und wirksame Heuristik-Funktion.
- man kann jedoch eine perfekte Heuristik angeben, weil die optimale Lösung (für 3 Türme) bekannt ist.
- das nützt dann aber nichts mehr (wenn man die optimale Lösung schon kennt, braucht man keine Heuristik)
- hätte ein Programm die Struktur der optimalen Lösung erkennen können? Es müßte das wichtige Teilziel „die unterste Scheibe muß bewegt werden“ entdecken.

Hausaufgaben (für KW17)

1. Das Verfahren *iterierte DFS* führt nacheinander für $k = 0, 1, \ldots$ eine DFS mit Tiefenschranke k durch (d.h., wenn der Keller schon k Elemente enthält, ändert Push den Zustand nicht) und hält, sobald ein Ziel erreicht wird.

   Ist das Verfahren auf endlichen Graphen vollständig? optimal?

2. (Gegen)beispiele für Schätzfunktionen für informierte Suche auf dann „live“ gegebenem Graphen (vgl. Vorlesung, evtl. autotool)

3. autotool: Türme von Hanoi (Pflicht: 3 Türme, Zusatz: 4 Türme)

4. Geben Sie eine planare Zeichnung des

Übergangs-Graphen aller Hanoi-Konfigurationen für 3 Türme und 2 Scheiben an.

Verallgemeinern Sie auf 3, 4, . . . Scheiben.

Wir suchen nun eine Zugfolge von A^n nach C^n ohne Züge $A - C$.

Tragen Sie eine kürzeste Lösung für $n = 2$ und evtl. $n = 3$ in Ihre Zeichnung ein.

Geben Sie eine allgemeine Lösung der Aufgabe an.

5. (Zusatz) Besorgen Sie 5 (oder mehr) Papp- oder Holzscheiben verschiedener Größen, die man gut sehen und anfassen kann (Durchmesser zw. 10 und 30 cm), und führen Sie das Hanoi-Umstapeln konkret vor, mit Stoppuhr (Ziel: ein Zug in 1/4 Sekunde)

- Die Scheiben liegen einfach nur übereinander. Das Auffädeln auf einen senkrechten Mittelstab raubt zu viel Zeit.
- Der Algorithmus in der rekursiven Form ist hier ungeeignet, da man sich den Stack (der gerade noch laufenden Unterprogramme) praktisch nicht merken kann. Man muß stattdessen beim Ansehen einer Konfiguration sofort den nächsten Zug zu erkennen (D.h., die Rekursion durch eine Iteration ersetzen.) Fällt Ihnen dazu ein Verfahren ein? Beweisen Sie dessen Korrektheit.
- Läßt sich die Aufgabe zu zweit schneller lösen? Es muß trotzdem eine korrekte Zugfolge realisiert werden, aber die beiden können sich die Arbeit (das Bestimmen und

Ausführen des jeweils nächsten Zuges) geeignet teilen.

6. (Zusatz) Lösen Sie das Rätsel in Fig. 1 im zitierten Artikel von Hearn und Demain. Vorsicht!

7. (Zusatz) Finden oder konstruieren Sie Lunar-Lockout-Instanzen mit langer kürzester Lösung. Kann es länger dauern als:

```
. A . . . . . . . . . . .
. . . e . . . . . E F . B
. . . . . . . . . .   . .
D . . . . . . . . . . . .
. . . . . . . . . . C .
```

Sie können `https://gitlab.imn.htwk-leipzig.de/waldmann/`

`ki-ss18/blob/master/search/Lockout.hs`

verwenden, aber für ernsthaftes Arbeiten sollte man

- Lösungsverfahren verbessern (unlösbare Konfigurationen schnell erkennen)
- nicht alle Startkonfigurationen erzeugen

Finite-Domain Constraints

Beispiel, Einordnung

- ein *finite-domain-Constraint-System*:

  $x_0, x_1, x_2, x_3 \in \{0, 1, 2, 3\} \wedge$ alldiff$(x_0, x_1, x_2, x_3) \wedge$ alldiff$(x_0 + 0, x_1 + 1, x_2 + 2, x_3 + 3) \wedge$ alldiff$(x_0 - 0, x_1 - 1, x_2 - 2, x_3 - 3)$.
- die Semantik der Relationssymbole:

  alldiff$(t_0, \ldots, t_{n-1}) := \forall 0 \leqslant i < j < n : t_i \neq t_j$
- eine Lösung (Belegung) $s = \{\ldots, (x_1, 3), \ldots\}$
- finden durch schrittweises Einschränken von Bereichen für Unbekannte, bis diese Einermengen sind
- grundsätzlich anderes Vorgehen bei unendlichen Bereichen ($\mathbb{N}$, $\mathbb{R}$, Terme): Master-VL Constraint-Progr.

Definition FD-CS und Lösung

- ein FD-CS besteht aus
  - Zuordnung: dom: Variablen $V \rightarrow$ endlicher Bereich
  - Constraints: $F =$ Konjunktion von Atomen,
    Atom $=$ Relationssymbol über Termen,
    Terme aus Funktionssymbolen und Variablen
  - Struktur S (Interpretation der Relations- und Funktionssymbole)
- eine Lösung eines FD-CS ist Variablenbelegung b mit
  - $\forall v : b(v) \in \mathrm{dom}(v)$ (jede Variable hat einen zulässigen Wert)
  - wert$(F, S, b) = 1$ (die Formel F ist in der Struktur S unter der Belegung b wahr)

Beispiel: Sudoku

- $I = \{1, 2, 3\}$, Farben $F = I^2$,

  Positionen (Variablen-Indices) $P = I^4$
- $\mathrm{dom} : P \mapsto$ jeweils $\{$Vorgabe$\}$ oder F
- Constraints:

$$\bigwedge_{(i,j) \in I^2} \text{alldiff}\{v_{i,j,k,l} \mid (k,l) \in I^2\}$$

$$\wedge \bigwedge_{(k,l) \in I^2} \text{alldiff}\{v_{i,j,k,l} \mid (i,j) \in I^2\}$$

$$\wedge \bigwedge_{(i,k) \in I^2} \text{alldiff}\{v_{i,j,k,l} \mid (j,l) \in I^2\}$$

- Struktur: alldiff wie im ersten Beispiel

Graphkantenfärbung

- Bsp. $n = 5, k = 3$, Variablen: $\binom{n}{2}$, $\mathrm{dom} : v \mapsto \{0, 1\}$
- Constraints:

$$\bigwedge \left\{ \mathsf{Or}\{v \mid v \in \binom{s}{2}\} \wedge \mathsf{Or}\{\neg v \mid v \in \binom{s}{2}\} \mid s \in \binom{n}{k} \right\}$$

  Ü: einige Atome explizit angeben. Wieviele insgesamt?

- Semantik: Or: k-stelliges Oder, $\neg$: Negation
- Übungen:
  - Lösung für $n = 5, k = 3$ angeben
  - Unlösbarkeit für $n = 6, k = 3$ beweisen
  - Status für $n = 17, k = 4$? Für $n = 43, k = 5$?

Lösungssuche (Spezifikation)

- Konfiguration (während der Lösungssuche) ist Abbildung c, die jedem $v \in V$ eine Teilmenge von $\mathrm{dom}(v)$ zuordnet
- die Domain-Zuordnung ist eine Konfiguration
- die Lösung ist eine Konfiguration (mit $\forall v : |c(v)| = 1$)
- während der Suche betrachte Menge Q von Konfigurationen, Start mit $Q = \{\mathrm{dom}\}$ (Einermenge)
- in jedem Schritt werden ein c aus Q entfernt und alle Kinder von c hinzugefügt.
- Invariante: es gibt ein s, das F löst $\iff \exists c \in Q : s \in c$.
  Dabei Notation: $s \in c$, falls $\forall v \in V : s(v) \in c(v)$.
- Spezialfall: $(\exists v : c(v) = \emptyset) \Rightarrow \neg(\exists s : s \in c)$
  jedes solche c heißt fehlgeschlagen (failed)

Lösungssuche (Realisierung)

- Notation: für Abbildung $c : A \to B$

  $c[x := y]$ bezeichnet $z \mapsto$ (wenn $z = x$, dann y, sonst $c(z)$)

- Übergänge zw. Konfigurationen durch
  - *Entscheidung*: für jedes $d \in c(v)$ gilt $c \to c[v := \{d\}]$
    (c hat $|c(v)|$ solche Nachfolger)
  - oder *Propagation* (Def. folgt), (c hat 1 Nachfolger)

- Strategie:
  - immer propagieren
  - dann Variable mit kleinstem $|c(v)|$ entscheiden
    falls $\min\{|c(v)| : v \in V\} = 0$, dann c fehlgeschlagen

Konsistenz von Konfigurationen

- eine Konfiguration c heißt (Hyperkanten-) *konsistent*,

  wenn für jedes Atom A und jedes $v \in \mathrm{Var}(A)$ gilt:

  jede Belegung b von v mit $b \in c$

  kann zu einer Belegung b' von $\mathrm{Var}(A)$
  mit $b' \in c$ fortgesetzt werden (d.h., $b \subseteq b'$),

  die A erfüllt (d.h., $\mathrm{wert}(A, S, b') = 1$)

- Bsp: $F = (x < y), c(x) = \{1, 2\}, c(y) = \{1, 2, 3\}$

  ist nicht konsistent, betrachte $b(y) = 1$.

- Bsp: $F = (x = y) \wedge (x \neq y), c(x) = c(y) = \{1, 2\}$ ist . . . ?

- Bsp: $F = \mathrm{alldiff}(x, y, z), c(x) = c(y) = \{1, 2\}, c(z) = \{2, 3\}$?

Propagation

- für Konfiguration c, Atom A, Variable $v \in \mathsf{Var}(A)$:

  bestimme die Teilmenge D der $d \in c(v)$, die man fortsetzen kann.

- Falls $D \subset c(v)$, dann heißt $c \to c[v := D]$ eine (erfolgreiche) *Propagation*.

- jede Propagation ist gut (weil der Suchbaum dort nicht verzweigt)

- und eine mit $D = \varnothing$ ist besonders gut, weil man einen erfolglosen Teilbaum abschneiden kann

Beispiele für Propagatoren

- für Atome $x \neq y$ (Ungleichheit, $x, y \in$ Var)
  - wenn $|c(x)| = 1$, dann $c \rightarrow c[y := c(y) \backslash c(x)]$
- für Atome Or$(l_1, \ldots, l_n)$ (Disjunktion)

  die l_i sind Literale (Variable oder negierte Variable)
  - wenn alle l_i bis auf einen (l_j) falsch unter c sind, dann $c \rightarrow c[\text{Var}(l_j) := \text{sign}(l_j)]$
- für Atome alldiff$(x_1, \ldots, x_n)$ (die x_i sind Variablen)
  - wenn $\exists A \subset \{x_1, \ldots, x_n\} = V$ mit $|A| = |B|$ für $B = \bigcup_{x \in A} c(x)$, dann $c \rightarrow c[y := c(y) \backslash B \mid y \in V \backslash A]$

SAT-Kodierungen

- das „klassische“ Lösungsverfahren für FD-CS:
  - domainspezifische Propagatoren
  - Strategien (`https://arxiv.org/abs/1203.1095`)
- alternativ (oft einfacher und wirksamer)
  - Übersetzung des FD-CS in ein SAT-Constraint (die Atome sind Or$(\ldots)$, die Domains sind $\{0, 1\}$)
  - Lösung durch
    - einfachen und *effizienten* Propagator für Or
    - effiziente Strategie
    - Konflikt-Analyse $\Rightarrow$ *Lernen* von zusätzlichen Atomen, um Teilbäume der weiteren Suche abzuschneiden

Einzelheiten in VL Constraint-Programmierung (Master)

SAT-Kodierung für das Damen-Problem

- unbekannte Zahl $x \in \{1, 2 \ldots n\} = I$ kodiert als
  Folge von n unbekannten Booleans $b_1, \ldots, b_n \in \{0, 1\}$,
  von denen *genau einer* wahr sein soll (*one-hot* encoding)
- Konfiguration ist gelöst, wenn $\bigwedge_i \mathsf{EO}\{b_{i,k} \mid k \in I\}$ und
  $\bigwedge_{i<j} \bigwedge_k \mathsf{Or}(\neg b_{i,k}, \neg b_{j,k}) \wedge \ldots$
- $\mathsf{EO}(\vec{b}) = \mathsf{Or}(\vec{b}) \wedge \bigwedge_{i<j} \mathsf{Or}(\neg b_i, \neg b_j)$ mit $\Theta(n^2)$ Atomen
  es geht mit $O(n)$ Atomen (und $O(n)$ Hilfsvariablen)
- alternative Kodierung des Damen-Problems:
  $\bigwedge_i \mathsf{EO}\{b_{i,k} \mid k \in I\} \wedge \bigwedge_k \mathsf{EO}\{b_{i,k} \mid j \in I\} \wedge$
  $\bigwedge_s \mathsf{Or}\{b_{i,j} \mid i + j = s\} \wedge \bigwedge_d \mathsf{Or}\{b_{i,j} \mid i - j = d\}$
- `https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss18/blob/master/fd-cs/Q.hs`

Mini/Flat-Zinc als Standard für FD-CS

- MiniZinc `http://www.minizinc.org/`:

  constraint modeling language (für den Menschen)
- mit Compiler (`libminizinc`) nach FlatZinc:

  constraint modeling language für die Maschine
- FlatZinc-Implementierungen (d.h., Solver),

  Bsp: `http://www.gecode.org/`,
  `https://projects.coin-or.org/Cbc`
- `mzn-fnz -f fzn-gecode queens.mzn -Dn=4`
  `mzn-cbc  queens.mzn -Dn=4 -Glinear`
  `==>  q = array1d(1..4 ,[3, 1, 4, 2]);`

Hausaufgaben (für KW 18)

1. (autotool) ein FD-CS lösen. Eine Folge von Schritten beschreibt eine Tiefensuche bis zu einer Lösung oder dem Beweis der Unlösbarkeit. Die Schritte sind:

   - `Decide <var> <wert>` einen Wert festlegen und diesen Teilbaum betreten, die restlichen Belegungen ergeben eine Konfiguration im Keller
   - `Arc_Consistency_Deduction` eine Variable einschränken (unsere Bezeichnung war: Propagation)
   - `Backtrack` der aktuelle Teilbaum c ist fehlgeschlagen, hole nächste Konfiguration vom Keller
   - `Solved` aktuelle Konfiguration ist Lösung
   - `Inconsistent` es gibt keine Lösung

2. Hochhaus-Rätsel in Minizinc

- Ergänzen Sie die Vorlage `https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss18/blob/master/fd-cs/skyline.mzn` (benutzen Sie minizinc-Dokumentation)
- lösen Sie die zitierte Beispiel-Instanz.
  Aufruf z.B. so:

```
BASE=/usr/local/waldmann/opt
export PATH=$BASE/gecode/latest/bin:$BASE/
export LD_LIBRARY_PATH=$BASE/gecode/latest

mzn-fzn -f fzn-gecode skyline.mzn
```

- Modellieren Sie die Variante „Hochhäuser mit Parks“, lösen Sie eine Instanz.

3. Graph-Kanten-Färbung in CNF-SAT

- Geben Sie das FD-CS (wie in VL) für $n = 5, k = 3$ im DIMACS-Format an.
- Lösen Sie mit `minisat` (in `$BASE/minisat/latest/bin`)
- Beweisen Sie (auf dem Papier), daß die Instanz $n = 6, k = 3$ nicht lösbar ist (Jede 2-Färbung der Kanten eines K_6 enhält einen einfarbigen K_3).

Spielbäume (adversarial search)

Modellierung von 2-Personen-Spielen

- endliche Zweipersonenspiele mit vollständ. Information.
- Knoten (Konfigurationen) C, Aktionen (Züge) A, Werte V
- in jedem Knoten: Nachfolgermenge *oder* Bewertung

  final : $C \to$ Bool, next : $C \hookrightarrow 2^{A \times C}$, wert : $C \hookrightarrow V$
- – *normale Spiele*: wer nicht mehr ziehen kann, hat verloren, der andere gewonnen, z.B. R gewinnt: Wert $+1$, L gewinnt: Wert -1.
  - verallgemeinert: beliebige Punktzahl, ermöglicht
    - Unentschieden (Remis)
    - heuristische Bewertung bei verkürzter Suche

Modellierung von Spielzuständen

- bei einfachen Brett-Spielen

  Konfiguration $=$ das, was man auf dem Brett sieht

  $C = (\text{Pos} \rightarrow \{\text{Schwarz}, \text{Weiß}, \text{Leer}\})$

  – Gomoku (5 in einer Reihe)
  – Atari-Go (wer den ersten Stein fängt, gewinnt)

- Schach: außerdem

  – Rochaden, Schlagen en passant, . . . (Ü)

- Go: außerdem

  – Anzahl der Gefangenen
  – Ko-Status. Ü: was ist die *positional-super-ko*-Regel?

Wert eines Spielbaums

- der (tatsächliche) Wert eines Spielbaums ist der
  - bei *optimalem* Spiel des Anziehenden
  - und *beliebigem* Spiel des Nachziehenden

  beste erreichbare Wert einer finalen Konfiguration.
- Namen der Spieler: R = rechts = rot = positiv (heiß), L = links = blau = negativ (kalt)
- erweitere Wert-Funktion auf nicht-finale Konfigurationen:

  Falls R dran: wert$(G) := \max\{\text{wert}(H) \mid (a, H) \in \text{next}(G)\}$,

  sonst L dran: wert$(G) := \min\{\text{wert}(H) \mid (a, H) \in \text{next}(G)\}$.
- ist wohldefiniert durch Induktion nach Länge von G
- Bewertung für alle Konfig. bestimmt werden (von unten nach oben), *aber das dauert im allg. zu lange*.

Subtraktions-Spiele

- Parameter ist endl. Menge $S \subseteq \mathbb{N}_{>0}$, Bsp: $S = \{2, 5\}$
- Konfigurationen: $\mathbb{N}$
- Spielzüge: next$(n) = \{(s, n - s) \mid s \in S \wedge n - s \geqslant 0\}$

  normales Spiel (wer nicht ziehen kann, hat verloren)
- Spielwerte durch dynamische Programmierung

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13
wert(n)	$-$	$-$	$+$	$+$	$-$									

- Satz: $\forall$ endliche S: Folge ist *schließlich periodisch*.
- Beweis: benutzt Lemma: die Periodenlänge ist $\leqslant 2^{\max S}$

Abspeichern von Bewertungen

- Aufgabe: Funktion wert : $C \to V$ effizient repräsentieren
- bei Subtraktionsspielen war das einfach:
  $C = \mathbb{N}$, Repräsentation $=$ Zahlenfolge
- für (Brett)spiele allgemein: verwende Hashtabelle
  (die wird dann gern *transposition table* genannt, warum?)
- mit *inkrementeller* Bestimmung des Hashwertes, d.h.,
  es gibt einfache Funktion $i : A \times \mathbb{N} \to \mathbb{N}$ mit
  $c_0 \xrightarrow{a} c_1$ impliziert $i(a, h(c_0)) = h(c_1)$
- Bsp: (Albert L. Zobrist, 1969) $h(c) := \sum\{f(p, x) \mid (p, x) \in c\}$
  für eine willkürliche Funktion f : Pos $\times$ Farbe $\to \mathbb{Z}$
- damit komplette Analyse von (Schach)(End)Spielen

Verkürzte Spielbaum-Bewertung

vollst. Bewertung i.A. nicht möglich. Verkürzen durch:

- (exakte Methode) Abschneiden (*pruning*) von uninteressanten Teilbäumen

  (deren Wert das Minimum bzw. Maximum nicht ändert)
- (Heuristik) angenäherte Bewertung von Teilbäumen (ohne Züge auszuführen)

  (z.B. Schach: Anzahl von Figuren, v. bedrohten Feldern)
- (exakt) Kombination der Bewertung von Teilspielen

  (z.B. Go: Ecke links unten, Ecke rechts oben, usw.)

  klassisches Beispiel: vollständige Analyse von Nim (beliebig viele beliebig große Haufen)

  Roland P. Sprague 1935, Patrick M. Grundy 1939

Alpha-Beta-Suche (Motivation, Spezifikation)

- Beispiel 1 zur Motivation:
  - Baum $a \to b, a \to c, c \to d, c \to e$, Bewertung $b : 5, d : 3$.
  - wert(a) hängt nicht von wert(e) ab!
- Beispiel 2 zur Motivation:
  - Baum $a \to b, a \to c, c \to d, c \to e$, Bewertung $b : 5, d : 7$.
  - welcher Bereich für wert(e) ist interessant?
- benutze Funktion pwert : $V \times V \times C \to V$

  mit Spezifikation pwert$(\alpha, \beta, G) =$ reduce$_{\alpha,\beta}$(wert(G))

  mit reduce$_{\alpha,\beta}(p) :=$ falls $\alpha < p < \beta$, dann p;
  falls $p \leqslant \alpha$, dann $-\infty$; falls $p \geqslant \beta$, dann $+\infty$.
- Anwendung: wert$(G) =$ pwert$(-\infty, +\infty, G)$.

Alpha-Beta-Suche (Implementierung)

- R am Zug $\Rightarrow$ pwert$(\alpha, \beta, G) = h_R(\alpha, \beta,$ next$(G))$
  - Spez.: $h_R(\alpha, \beta, xs) =$ reduce$_{\alpha,\beta}(\max\{$wert$(x) \mid x \in xs\})$
  - Implementierung (Lückentext, ergänze lt. Spez.)

```
h_R alpha beta xs = case xs of
 Nil          -> _
 Cons x xs' ->
  let v = pwert alpha beta x
  in  if  v >= beta
      then  +inf  else  h_R  _  _  xs'
```

  - wenn $v \geqslant \beta$: folgende Kinder (xs') werden nicht bewertet (besucht, erzeugt), Kanten zu diesen heißen β-*cutoffs*
- ergänze: L am Zug $\Rightarrow$ pwert$(\alpha, \beta, G) = h_L(\alpha, \beta,$ next$(G))$

Computer-Schach

- Claude Shannon: *Programming a computer for playing Chess*, Philosophical Magazine, 1950;
  - Materialgewicht K 200, D 9, T 5, L 3, S 3, B 1,
  - Minimax-Bewertung (wenige Züge) mit *Ruhesuche*: tiefer suchen, falls Zug exist., der Bewert.g stark ändert
- Alan Turing, *Digital Computers applied to Games*, 1953
- Donald Knuth, Ronald Moore: *An Analysis of* α/β *pruning*, Artificial Intelligence 1975
- 1997 Deep Blue (Feng-hsiung Hsu et al., IBM) schlägt Weltmeister Garry Kasparov
- 2017 Alpha Zero (David Silver et al., Deep Mind) `https://arxiv.org/abs/1712.01815` schlägt Stockfish

Ü: Heuristik für Gomoku

- Suchtiefe ist nur 1 Halbzug!
- ```
  (defconst gomoku-nil-score   7  "Score of an empty
  (defconst gomoku-Xscore  15  "Score of a qtuple con
  (defconst gomoku-XXscore 400  "Score of a qtuple co
  (defconst gomoku-XXXscore     1800  "Score of a qtu
  (defconst gomoku-XXXXscore  100000  "Score of a qtu
  (defconst gomoku-Oscore  35  "Score of a qtuple con
  (defconst gomoku-OOscore 800  "Score of a qtuple co
  (defconst gomoku-OOOscore    15000  "Score of a qtu
  (defconst gomoku-OOOOscore  800000  "Score of a qtu
  ```
- Finden Sie Begründungen für diese Zahlen im Quelltext auf (autoritative Quelle) `*.gnu.org/*`.
- Können Sie gegen die Heuristik gewinnen?
- Finden Sie bessere Gewichte?
  ```

Ü: Heuristik für Atari-Go

- Atari-Go: wer als erster eine Kette fängt, hat gewonnen

  (Def: Kette $=$ zusammenhängende Teilmenge)

  mit vorgegebener Anfangsposition $\begin{matrix} O & X \\ X & O \end{matrix}$

- Def: $F(K) :=$ Anzahl der Freiheiten einer Kette K.

  Die Bewertung der Konfiguration für Spieler S ist

  $W(S) := \min\{F(K) \mid K$ ist Kette von $S\}$

- Betrachten Sie die Heuristik:
  - wenn $W(\text{ich}) < W(\text{Gegner})$,
  - dann $W(\text{ich})$ vergrößern,
  - sonst $W(\text{Gegner})$ verringern.

- Geben Sie eine Situation an, von der aus diese Heuristik sicher gewinnt, aber erst nach einigen Zügen
- Können Sie von o.g. Startkonfiguration (Kreuz) aus gegen die Heuristik gewinnen?
- didaktische Bemerkung:

  beim Go sind das (angedrohte) Fangen, aber auch das Opfern von Steinen nur Mittel zum eigentlichen Zweck: Gebiet machen.

  manche Lehrer empfehlen Atari-Go für Anfänger, andere finden es eher riskant, vgl. *idea and criticism*, `https://senseis.xmp.net/?AtariGoTeachingMethod`

Hausaufgaben für KW20

1. autotool-Aufgabe Alpha-Beta
2. Heuristik Gomoku (autotool)
3. Heuristik Atari-Go (live)

   im Pool Z430: GUI zum Go-Spielen: `qgo`,

   dort kann auch das Programm `gnugo` als Gegner eingestellt werden (alles unter `/usr/local/waldmann/opt/{qgo,gnugo}/latest/`

   eine bestimmte Startsituation vorgeben: im SGF-Editor erzeugen, abspeichern, dann unter “go engine” laden.

   Spielen Sie auf diese Weise Atari-Go gegen gnugo. (Das weiß aber nicht, daß es Atari-Go ist, also wird es nicht

jede einzelne Kette verteidigen. Deswegen können Sie leicht gewinnen.)

4. Geben Sie ein Subtraktionsspiel (d.h., eine Menge S) mit großer Periode an. (evtl. autotool, noch in Arbeit)

5. Geben Sie ein Programm an, das aus S die Folge der Spielwerte bestimmt.

   Kurze Lösung (kann man direkt in ghci ausprobieren)

```
import Data.List (transpose)
shift xs d = replicate d True ++ xs
values s = let xs = map (not . and) $ transpose $ m
take 10 $ values [2, 5] -- Beispiel von der Folie, 
[False,False,True,True,False,True,True,False,False,
```

6. . . . die Periodenlänge bestimmt

Kombinatorische Spieltheorie

Motivation, Definition

- KST (engl. combinatorial game theory, CGT) ist die Lehre von den *disjunkten Summen* von Spielen (normale Zweipersonenspiele mit vollständiger Information)
- Anwendung: modulare Analyse: anstatt wert(G) bestimmt man wert(G_1) + wert(G_2) + $\cdots$ + wert(G_n)
- Besonderheit: der Wertebereich ist nicht $\mathbb{Z}$, sondern komplizierter (z.B.: nicht total geordnet)
- John Conway: *On Numbers and Games*, 1976; Elwyn Berlekamp, John Conway, Richard Guy: *Winning Ways*, 1982.

Welche Spiele werden hier betrachtet?

- Def: ein Spiel ist ein Paar von Mengen von Spielen.
- Bsp: das sind Spiele (Namen werden später begründet)
  - 0. Tag: $0 = (\varnothing, \varnothing)$
  - 1. Tag: $-1 = (\varnothing, \{0\})$, $1 = (\{0\}, \varnothing)$, $*1 = (\{0\}, \{0\})$
  - 2. Tag: $2 = (\{1\}, \varnothing)$, $1/2 = (\{0\}, \{1\})$, $*2 = (\{0, *1\}, \{0, *1\})$
- Bezeichnungen: $G = (G_L, G_R)$, dabei $G_L =$ Menge der linken Optionen, $G_R =$ Menge der rechten Optionen.
- der Spielbaum von $1/2$ ist ...
- Abkz. $\{l_1, \ldots, l_i \mid r_1, \ldots, r_j\}$ für $(\{l_1, \ldots, l_i\}, \{r_1, \ldots, r_j\})$

  $1/2 = \{0 \mid 1\}, *2 = \{0, *1 \mid 0, *1\}, 1 = \{0 \mid \ \}, 0 = \{\mid\}$.

Hackenbush

- Konfiguration: Multimenge von Bäumen mit Wurzeln, Kanten sind blau (L), rot (R) oder grün (G)
- Spielzug: für $x \in \{R, L\}$:
  - x löscht eine x- oder G-Kante eines Baumes
  - sowie alle Knoten und Kanten, die nicht (mehr) mit Wurzel verbunden sind.
- Bsp: $* =$ eine G-Kante, $1 =$ R-Kante, $-1 =$ L-Kante
- Bsp: Wurzel–Blau–Rot $= \{0 \mid 1\} = 1/2$.
- Bsp: Wurzel–Grün $= \{0 \mid 0\} = *$
- Bsp: Wurzel–Grün–Grün $= \{0, * \mid 0, *\} = *2$

Spielklassen, Gegensätze

- Def: Spiel G heißt ...
  - positiv, wenn L bei optimalem Spiel gewinnt (Bsp: 1)
  - negativ, wenn R gewinnt (-1)
  - Null-Spiel, wenn der Nachziehende gewinnt (0)
  - unscharf (fuzzy), wenn der Anziehende gewinnt ($*$)
- Def: das zu G entgegengesetze Spiel ist

  $-G := (\{-g \mid g \in G_R\}, \{-g \mid g \in G_L\})$.
- Bsp: $-(1) = (-1)$, Bsp: $-* = *$
- Hackenbush: negieren $=$ (Rot $\leftrightarrow$ Blau)
- Satz: G positiv $\iff$ $-G$ negativ
- Satz: G unscharf $\iff$ $-G$ unscharf

Die disjunkte Summe von Spielen

- $G + H := \begin{array}{l} (\quad \{g + H \mid g \in G_L\} \cup \{G + h \mid h \in H_L\} \\ , \quad \{g + H \mid g \in G_R\} \cup \{G + h \mid h \in H_R\} \ ) \end{array}$
- d.h., in G oder H zu einer Option ziehen, der andere Summand bleibt unverändert
- ist assoziativ, kommutativ, 0 ist (rechts und links) neutral
- $0 + H = (\{0 + h \mid h \in H_L\}, \{0 + h \mid h \in H_R\}) = H$ (Indukt.)
- $1 + 1 = \{0 \mid\} + \{0 \mid\} = \{1 + 0, 0 + 1 \mid\} = \{1 \mid\} = 2$
- $-1 + 1 = \{\mid 0\} + \{0 \mid\} = \{-1 \mid 1\}$ ist Null-Spiel
- $G + (-G)$ ist Null-Spiel. $G + (-G) \to g + (-G) \to g + (-g)$
- $1/2 + 1/2 = \{0 \mid 1\} + \{0 \mid 1\} = \{1/2 \mid 1 + 1/2\}$
- $* + * = \{0 + * \mid 0 + *\} = \{* \mid *\}$ ist Null-Spiel

Äquivalenz von Spielen

- Bezeichnung: $G - H := G + (-H)$
- Def: $G \approx H$ gdw. $G - H$ ist Null-Spiel.
- Beweise durch Strategie für Hackenbush-Position:
  $1 + 1 \approx 2$, $1/2 + 1/2 \approx 1$, $* + * \approx 0$
- Satz: Wenn $H \approx 0$, dann $G + H \approx G$

  Bew: $G + H + (-G) = (G + (-G)) + H$. Satz folgt aus

  Lemma: Summe von Null-Spielen ist Null-Spiel.
- Bsp: $\{0, 1 \mid \ \} \approx \{1 \mid \ \} = 2$, Bew: Hackenbush
- Festlegung: ab jetzt soll „$=$“ die Bedeutung „$\approx$“ haben.

Die (Halb)Ordnung auf Spielen

- Def: $G > H$ gdw. $G - H$ positiv (gdw. L gewinnt)
- Beispiele: $2 > 1 > 1/2 > 0$

  Beweise durch Hackenbush-Strategien
- Nicht-Beispiel: $* \not> 0$ und $0 \not> *$.
- Def: $G < H$ gdw. $H > G$
- Def: $G \geqslant H$ gdw. $G - H$ positiv oder Null-Spiel
- Satz: diese Relation $\geqslant$ ist reflexiv und transitiv.

  Nicht antisymmetrisch, aber $(x \geqslant y \wedge x \leqslant y) \Rightarrow x \approx y$.

  D.h., antisymmetrisch auf Äq.-Klassen bzgl $\approx$.

Vereinfachung von Spielen, Domination

- Ziel: zu gegebenem G suchen wir ein einfacheres (im Idealfall: das einfachste) H mit $G \approx H$.
- bisher: müßten wir erst H raten und dann beweisen, daß $G - H$ Null-Spiel ist
- jetzt: äquivalentes Umformen von Spielen
- Bsp: $\{0, 1 \mid \ \} \approx \{1 \mid\}$

  Option 0 ist unwichtig für L, denn 1 ist besser ($1 > 0$)
- Satz (Löschen von dominierten Optionen) (vgl. α/β-cuts)

  $x > y \Rightarrow \{\ldots x, y, \ldots \mid \ldots\} \approx \{\ldots, x, \ldots \mid \ldots\}$.

  $x < y \Rightarrow \{\ldots \mid \ldots x, y \ldots\} \approx \{\ldots \mid \ldots, x, \ldots\}$.

Reversible Optionen, Normalformen

- Def: für Spiel G:

  Option $x \in G_L$ heißt *reversibel*, wenn $\exists y \in x_R$ mit $y \leqslant G$.
- Satz: (das Ersetzen einer reversiblen Option)

  $x \in G_L$ revertiert durch $y \Rightarrow G \approx (G_L \backslash \{x\} \cup y_L, G_R)$.
- Bsp: für $\uparrow = \{0 \mid *\}$ gilt $G = \{\uparrow|\uparrow\} \approx \{0 \,|\uparrow\} = H$.

  Zeige $* < G$ (d.h., L gewinnt $G - *$, äq. $G + *$)

  Damit wird $\uparrow \in G_L$ revertiert durch $* \in \uparrow_R$

  und kann ersetzt werden durch $*_L = \{0\}$.
- Satz: zu jedem G exist. genau ein H ohne dominierte und reversible Optionen mit $G \approx H$. Heißt Normalform.
- Bsp. $\{0 \,|\uparrow\}$ ist Nf von $\{\uparrow|\uparrow\}$. Bsp. Nf. von $\{0, * \mid *\}$

Beispiel: Zahlen und die Größe von $\uparrow$

- Def (Wdhlg): $0 = \{|\}$, $G_0 = 1 = \{0\,|\}$, $G_1 = 1/2 = \{0 \mid 1\}$.
- Def: $G_2 = \{0 \mid 1/2\}$. Satz: $G_2 + G_2 = G_1$.
- Def: $G_{k+1} = \{0 \mid G_k\}$. Satz: $\forall k \geqslant 0 : G_{k+1} + G_{k+1} = G_k$

  Bew: zeige: $G_{k+1} + G_{k+1} - G_k$ ist Null-Spiel

  das rechtfertigt die Bezeichnung $1/2^k$ für diese G_k

- Def (Wdhlg): $\uparrow = \{0 \mid *\}$
- $0 <\uparrow$, $\uparrow< 1$
- $\uparrow + \uparrow< 1$, $\uparrow< 1/2$
- $\uparrow + \cdots + \uparrow< 1$, $\forall k :\uparrow< 1/2^k$

Neutrale Spiele

- Def: G heißt neutral gdw.
  - linke und rechte Optionen gleich: $G_L = G_R$
  - und alle Optionen von G_L (und G_R) sind neutral.
- Bsp:
  $0, \quad * = *1 = \{0 \mid 0\}, \quad *2 = \{0, *1 \mid 0, *1\}, \quad \{0, *2 \mid 0, *2\}.$
- abkürzende Schreibweise:

  Optionen nur einmal hinschreiben, kein Trennstrich

  $*2 = \{0, *1 \mid 0, *1\} = \{0, *1\}$
- Def (Nimbers): $*0 = 0, *(n+1) = \{*0, \ldots, *n\}$

  grüne Hackenbush-Ketten $=$ einzelne Nim-Haufen

Der Satz von Sprague und Grundy

- Satz: $\forall$ neutrales Spiel G $\exists n \in \mathbb{N} : G = *n$
- Bsp: $\{0, *2\} = *1$
- Lemma: $\forall n_1, \cdots \in \mathbb{N} : \{*n_1, \ldots, *n_k\} = *\,\text{Mex}\{n_1, \ldots, n_k\}$

  wobei Def: für $M \subseteq \mathbb{N}$ bezeichnet Mex $M := \min(\mathbb{N} \backslash M)$
- Beweis Lemma: sei $m = \text{Mex}(n_1, \ldots, n_k)$.

  Zeige, daß $\{*n_1, \ldots, *n_k\} + *m$ Null-Spiel ist.

  Fall 1: Zug links zu $*n_i$ mit $n_i < m$.

  Fall 2: Zug links zu $*n_i$ mit $n_i > m$ (ist reversibel)

  Fall 3: Zug rechts.
- Beweis Satz: durch Induktion über das Alter von G

Die Nim-Addition

- Satz: $*x + *y = *(x \oplus y)$,

  Def: $x \oplus y =$ binäre Addition *ohne Überträge*
- Bsp $1 \oplus 3 = 2$, deswegen $* + *3 = *2$,
- Anwendung: Nim-Position mit Haufen $\{1, 2, 3\}$ ist verloren
- Beweis: zeige, daß $*x + *y + *(x \oplus y)$ Null-Spiel ist.
  - schreibe Bin(x), Bin(y), Bin$(x \oplus y)$ untereinander.
    jede Spaltensumme ist 0.
  - für Zug in einem der drei Summanden:
    - $k :=$ die höchste geänderte Stelle (d.h., von 1 auf 0)
    - es gibt genau einen anderen Summa. mit 1 an Stelle k
    - dort kann man so spielen, daß alle Summen 0 werden
    - damit Beweis durch Induktion

Hausaufgaben für KW 20

1. autotool-Aufgaben zu voriger Ü-Serie (Minimax, Alpha/Beta, evtl. Gomoku-Heuristik)

2. Status und Gewinnzüge für eine Nim-Position berechnen (Bsp: $\{3, 7, 8\}$, weitere live an der Tafel)

3. Den Spielwert einer Hackenbush-Position angeben

4. (Zusatz) Def: ein Spiel G heißt *Zahl*, wenn

   - jede Option eine Zahl ist
   - und $\neg\exists x \in G_L, y \in G_R : x \geqslant y$.

   Bsp: $0, 1, -1, 1/2, 2$ sind Zahlen. $* = \{0 \mid 0\}$ ist keine Zahl.

   - Beweise: die Summe von zwei Zahlen ist eine Zahl.

- zwei Nicht-Zahlen angeben, deren Summe eine Zahl ist.
- Konstruiere eine Zahl x mit $1/2 < x < 1$.
- Finde ein/das kleinste Intervall von Zahlen a, b mit $a < \{1 \mid -1\} < b$.

5. (Zusatz - zur Diskussion in Übung) Beweisen Sie die Behauptung:

   Wenn w der (minimax-)Wert eines Spielbaumes B ist, und B' aus B dadurch entsteht, daß der Wert eines beliebigen Blattes in B auf w geändert wird, dann ist der Wert von B' immer noch w.

Wissensrepräsentation und -Verarbeitung durch Logik

Inhalt, Motivation

- die Welt wird beschrieben durch Aussagen mit Wahrheitswert

  (es regnet (jetzt), das Paket `emacs` ist installiert, ...)
- (Experten)wissen wird beschrieben durch Regeln

  (es regnet $\Rightarrow$ die Straße ist naß, ...)
- daraus möchte man weitere wahre Aussagen ableiten
- deswegen sind gesucht: korrekte und effiziente ...
  - Wissensrepräsentation (Bsp: Wahrheitswerttabellen)
  - Ableitungsverfahren (Bsp: Resolution)

Regelsysteme

- wir betrachten Mengen von Regeln der Form

  wenn (Voraussetzung), dann (Folgerung)

- es sind grundsätzlich als Semantik denkbar:
  - Welt-*Beschreibung*
    aus $R = (A \to B) \wedge W \models R \wedge W \models A$ folgt $W \models B$
    (Vorhersage von B aus Beobachtung von A)
    die bekannte Aussagen- und Prädikatenlogik
  - Welt-*Änderung*, Bsp. $(x > 0) \to (x := x - 1)$
    Modal-Logik, Wahrheitswerte sind Funkt. Zeit $\to$ Bool
- wir betrachten hier nur den Beschreibungs-Aspekt

  das enthält auch $A \to B$ als „wenn A, dann löse B aus“,
  aber wir beschreiben *nicht* die Wirkung von B

Beispiel für Regelsystem: CSS

- Cascading Style Sheets, `https://www.w3.org/TR/CSS22/`
- Beispiel `h1 , div * p {color:red}`
- Definition (Syntax)
  - 4.1.7 rule (set) $=$ selector declaration-block
  - 4.1.8 declaration $=$ property name `:` property value
- Definition (Semantik)

  benutzt das Modell: document tree $=$ Term-Baum t,
  selector $=$ Menge von Positionen $\subseteq$ Pos(t)
  - 5.1 pattern matching rules determine which style rules apply to elements in the document tree
  - 6.1 the user agent must assign, for every element in the tree, a value to every property ...

Einzelheiten zur CSS-Semantik

- wenn ein Element zu mehreren Selektoren gehört und deswegen mehrere Deklarationen einem seiner Attribute mehrere (unterschiedliche) Werte zuweisen,

  dann wirkt 6.4.1 cascading order:

  - user agent $<$ user normal $<$ author normal $<$ author important $<$ user important
  - bei Gleichstand: der spezifischere Selektor gewinnt 6.4.3. Spezifik: lexikografische Ordnung auf Tupeln von Attribut-Anzahlen
  - bei Gleichstand: der spätere im Quelltext gewinnt

Übungsaufgaben CSS

- user agent style sheet abschalten

  (Firefox: view → page style → no style)

  und dann Ihre bevorzugten Webseiten betrachten

- die Implementierung der CSS-Spezifikation im Quelltext Ihres bevorzugten Browsers suchen

  evtl. mit Hilfe von `https://searchfox.org/`.

  Hinweis: nach Begriffen aus der Spezifikation suchen, denn diese Bezeichnungen werden auch im Quelltext verwendet

Regeln in Tracking-Blockern

- worum geht es eigentlich?

  Vgl. Doc Searl: *GDPR will pop the adtech bubble*, `http://blogs.harvard.edu/doc/2018/05/12/gdpr/`

- Beispiel: Raymond Hill: μMatrix
  `https://github.com/gorhill/uMatrix`

- Beispiele für Regeln (add-ons manager → umatrix → preferences → my rules)

```
* * * block
* 1st-party * allow
autotool.imn.htwk-leipzig.de \
        htwk-leipzig.de * inherit
```

Übungsaufgaben Tracking-Blocker

- uMatrix installieren

  und dann Ihre bevorzugten Webseiten betrachten

- Semantik spezifizieren
  - für die Regeln (in Textform)
    welche Requests werden blockiert?
  - für die GUI
    was bedeuten: rot, schwach rot, schwach grün, grün

- den Quelltext suchen, der die Regelanwendung realisiert

Entscheidungstabellen

- stellen mehrstellige Boolesche Funktionen dar
- die Darstellung benutzt disjunktive Normalform, die aber nicht notwendig kanonisch ist

  (kanonisch $=$ in jeder Klausel kommen alle Variablen der Reihe nach vor)
- sind mglw. kürzer als eine vollständige Wahrheitswerttabelle ($=$ kanonische DNF)
- es gibt aber Funktionen, die sich so nicht komprimieren lassen (Beispiel: $(x_1, \ldots, x_n) \mapsto \sum x_i \mod 2$)
- äquivalente Darstellungsform, die in digitaler Schaltungstechnik verwendet wird:

  Karnaugh-Pläne (Maurice Karnaugh, 1953)

Bsp. Wahrheits/Entscheidungs-Tabelle

Formel $x \leftrightarrow (y \vee \neg z)$

links Wahrheitswerttabelle (Ergebnisse stehen unter $\leftrightarrow$)

rechts äquivalente Entscheidungstabelle (Regelmenge)

x	y	z	x	$\leftrightarrow$	$(y$	$\vee$	$\neg$	$z)$
0	0	0		0		1	1	
0	0	1		1		0	0	
0	1	0		0		1	1	
0	1	1		0		1	0	
1	0	0		1		1	1	
1	0	1		0		0	0	
1	1	0		1		1	1	
1	1	1		1		1	0	

x	y	z	e
0	0	1	1
1	0	0	1
1	1	$*$	1

Entscheidungsbäume

- Motivation: schnellere Auswertung

  (als lineare Suche nach passendem Muster in E-Tabelle)
- Definition: ein Entscheidungsbaum enthält
  - äußere Knoten (Blätter): Falsch (0), Wahr (1)
  - innere Knoten t: eine Variable, zwei Kinder

```
data EB v = Leaf Bool | Branch v (EB v) (EB v)
```

- Die Semantik von t ist Funktion Var$(t) \to$ Bool
  mit $s($`Branch v l r`$) =$ ite$(v, s(l), s(r))$
  und ite$(v, x, y) = (v \wedge x) \vee (\neg v \wedge y)$
- schnelle Auswertung unter einer Belegung: nur *ein* Pfad
- kann man den Platzbedarf noch reduzieren? (Ja: BDD)

Entscheidungsdiagramme (Begriff)

- ausgehend von Entscheidungsbäumen
  (ite-Knoten und Blätter bleiben, Semantik bleibt)
- *sharing*, Baum → gerichteter kreisfreier Graph, DAG

```
data Node v = Leaf Bool | Branch v Index Index
type BDD v = Map Index (Node v)
```

- *Ordnung*: man legt $(V, <)$ fest und es gilt: auf jedem Pfad $p_0 \to \cdots \to p_n$ von Wurzel zu Blatt ist $\forall i : p_i < p_{i+1}$
- *Reduktion*: alle Knoten sind paarweise verschieden
- Ü: ein solcher DAG für die Fkt $(x, y, z) \mapsto x \oplus y \oplus z$
  mögl. Lösungsweg: mit geordnetem E-Baum beginnen und dann identische Knoten zusammenfassen.
  Effiziente Implementierungen erzeugen den Baum *nicht*!

BDDs (Geschichte, Eigenschaften)

- Randall Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, 1986
- Reduziertes geordnetes binäres Entscheidungsdiagramm (ROBDD), oft abgekürzt zu BDD.
- Satz: zu jeder n-stell. aussagenlog. Funktion F ($=$ Formel mit n Variablen) und jeder Ordnung $(V, <)$

  gibt es *genau ein* ROBDD mit Semantik F

  (Beweis: das vorige Verfahren)
- Satz: für jedes $f : \mathrm{Bool}^k \to \mathrm{Bool}$ (z.B. 2-stell. Konjunktion) kann man aus BDDs $D_1, \ldots, D_k$

  das BDD D mit $\mathrm{Sem}(D) = f(\mathrm{Sem}(D_1), \ldots, \mathrm{Sem}(D_k))$

  in Zeit $|D_1| \cdot \ldots \cdot |D_k|$ konstruieren

BDDs (Anwendungen)

- aus Satz über Eindeutigkeit folgt: für aussagenlog. Formeln F, G gilt $F \leftrightarrow G$ gdw. BDD$(F) =$ BDD(G).

  Anwendung: F ein Schaltkreis-Entwurf (Spezifikation), G eine Implementierung. Korrektheitsbeweis durch Vergleich der BDDs.

- (Spezialfall des vorigen): für $F =$ False ergibt das ein Entscheidungsverfahren für Erfüllbarkeit von G

  (in der Praxis sind SAT-Solver oft schneller, man muß BDD(G) nicht konstruieren, um $\varnothing \neq$ Mod(G) zu beweisen)

- Mod$(\texttt{Br v l t}) = \{b \cup \{(v,0)\} \mid b \in \text{Mod}(l)\} \cup \{b \cup \{(v,1)\} \mid b \in \text{Mod}(r)\}$ ist disjunkte Vereinigung, deswegen kann man Modelle zählen

Hausaufgaben für KW 21

Wählen Sie eine der Aufgaben 1 und 2.

1. CSS
   - den lexikografischen Vergleich der CSS-Spezifiken (6.4.3) an (vorbereiteten!) Beispielen vorführen
   - den dafür relevanten Quelltext in Firefox finden und erklären
2. Blocking mit μMatrix:
   - die Farben der Anzeigematrix erklären und an (vorbereiteten!) Beispielen vorführen. (Der uMatix-Logger zeigt alle Requests.)
   - den Quelltext suchen, der Regelanwendung realisiert
3. (autotool) eine aussagenlogische Formel umformen

4. (autotool) ein BDD konstruieren

5. Konstruieren Sie das BDD für die Funktion

   $(x_1, \ldots, x_n) \mapsto$ genau die Hälfte der x_i sind wahr,

   zu Variablenordnung $x_1 < \ldots < x_n$, wobei n eine gerade Zahl ist, z.B. $n = 6$.

   Beschriften Sie jeden Knoten v mit der Anzahl der Modelle des Teil-BDDs, das bei v beginnt.

   Vergleichen Sie das mit der Wahrheit, d.h., bestimmen Sie die Anzahl der Modelle dieser Formel (für beliebiges n) noch auf eine andere Weise.

Resolution

Beispiel, Motivation

- Bsp: ein Resolutions-Schritt: $\frac{a \vee \neg b \vee c, \neg c \vee d}{a \vee \neg b \vee d}$

  aus zwei Klauseln wird eine weitere abgeleitet

  Klauseln modellieren Fakten und Regeln
- Resolution ist ein Inferenz-Verfahren
  - korrekt: wenn Voraus. wahr, dann abgel. Klausel wahr
  - widerlegungs-vollständig: für jede widersprüchliche Klauselmenge (ohne Modelle) gibt es eine Resolutionsableitung der leeren Klausel (die offensichtlich kein Modell hat)
- ist Grundbaustein für automatisches Schließen/Beweisen

Aussagenlogik: Syntax und Semantik

- Syntax: Variable, Literal, Operator, Formel, Formelmenge
- Semantik: Belegung b : Var $\rightarrow$ Bool,
  Wert einer Formel unter einer Belegung wert(b, F),
- b ist Modell von F, falls wert$(b, F) = 1$, Schreibweise $b \models F$

  Bsp: $\{(a, 0), (b, 0), (c, 0)\} \models a \vee \neg b \vee c$
- Modellmenge einer Formel Mod$(F) = \{b \mid b \models F\}$,

  Modellmenge einer Formelmenge
  Mod$(M) = \{b \mid \forall F \in M : b \models F\}$
- M widersprüchlich, falls $\varnothing =$ Mod(M), sonst erfüllbar

  Bsp: $\{a, \neg a\}$ ist widersprüchlich.

Disjunktive Klauseln

- Def: disjunktive Klausel $=$ Disjunktion von Literalen

  das Wort „disjunktiv“ wird hier oft weggelassen

  Beispiele $a, \quad a \vee \neg b, \quad a \vee \neg b \vee \neg c$
- alternative Schreibweise: nur die Literalmenge angeben

  Beispiele $\{a\}, \quad \{a, \neg b\}, \quad \{a, \neg b, \neg c\}$
- Spezialfall: False ist auch eine disjunktive Klausel (die Disjunktion über leere Literal-Menge)
- jede Klausel C ist äquivalent zu $\bigwedge \mathrm{Neg}(C) \rightarrow \bigvee \mathrm{Pos}(C)$

  wobei $\mathrm{Pos}(C) =$ Menge der in C positiv vorkommenden Variablen, $\mathrm{Neg}(C) =$ . . . negativ . . .
- Bsp: $\{a, \neg b, \neg c\}$ $(= a \vee \neg b \vee \neg c)$ ist äq. zu $(b \wedge c) \rightarrow a$

Spezialformen von Klauseln

- Def: eine Klausel heißt *Horn-Klausel*,

  falls sie höchstens ein positives Literal enthält.

- in der Implikations-Schreibweise jeder Horn-Klausel steht rechts (nach dem Pfeil) False oder eine Variable.

  das wird in Prolog benutzt:

  – Programm $=$ Menge solcher Klauseln (Regeln)
  – Anfrage Q wird umgeformt zu $\neg Q \rightarrow$ False

- Def: eine Klausel heißt *Krom-Klausel*, wenn sie höchstens zwei Literale enthält (Polarität ist egal)

Semantisches Folgern

- Def: Formel G folgt aus Formelmenge M,
  falls Mod$(M) \subseteq$ Mod(G). Schreibweise $M \models G$.
  Bsp: $\{a, \neg a \vee b\} \models b$
- Satz: M widersprüchlich gdw. $M \models$ False
  Bew: betrachte Mod(False)
- Satz: $M \models F$ gdw. $M \cup \{\neg F\}$ widersprüchlich
- aus Formelmenge wird Formel erzeugt (gefolgert),
  dabei werden Belegungen (Modelle) betrachtet,
  deswegen heißt das Vorgehen *semantisch*
- warum die gleiche Notation wie für Modell-Relation?
  Belegung als Formelmenge: $\{(a, 0), (b, 1)\}$ als $\{\neg a, b\}$.

Syntaktisches Schließen (Resolvieren)

- ein Resolutions-Schritt

$$\frac{x_1 \vee \ldots \vee x_m \vee y, \quad \neg y \vee z_1 \vee \ldots \vee z_n}{x_1 \vee \ldots \vee x_m \vee z_1 \vee \ldots \vee z_n}$$

- Sprechweise: Klauseln C_1, C_2 werden *nach y resolviert*
- Schreibweise: $C = C_1 \oplus_y C_2$

  Bsp: $(a \vee \neg b \vee c) \oplus_c (\neg c \vee d) = a \vee \neg b \vee d$,

  Bsp: $x \oplus_x \neg x =$ False
- Satz (Korrektheit eines Resolutions-Schrittes):

  $\{C_1, C_2\} \models C_1 \oplus_y C_2$

  Beweis: bel. $b \models \{C_1, C_2\}$, Fallunterscheidung nach $b(y)$

Resolution als Inferenzsystem, Korrektheit

- bisher *ein* Resolutions-Schritt, jetzt mehrere
- definieren Relation $\vdash$ (Klausel C *ist ableitbar* (durch Resolution) aus Klauselmenge M) durch
  - (Induktionsanfang) Wenn $C \in M$, dann $M \vdash C$
  - (Induktionsschritt)
    Wenn $M \vdash C_1$ und $M \vdash C_2$, dann $M \vdash C_1 \oplus_y C_2$
- beachte diese Unterschiede
  - Ableitung, $\vdash$, syntaktisch definiert (Term-Umformung)
  - Folgerung, $\models$, semantisch definiert (Term-Auswertung)
- Satz (Korrektheit der Resolution) $M \vdash C \Rightarrow M \models C$

  Beweis: Induktion über Länge der Ableitung,
  benutze Korrektheit eines Resolutions-Schrittes.

Beispiel Inferenz-Baum

- Für $M = \{a \vee b, \neg a \vee \neg b, a \vee c, \neg c \vee \neg b, b \vee d, \neg d \vee \neg a\}$

  ist das ein gültiger Inferenzbaum

$$\dfrac{\dfrac{\dfrac{a \vee c, \neg c \vee \neg b}{a \vee \neg b}, a \vee b}{a} \quad \dfrac{\dfrac{b \vee d, \neg d \vee \neg a}{b \vee \neg a}, \neg a \vee \neg b}{\neg a}}{\text{False}}$$

- es gilt Mod$(M) = \varnothing$. Beweis (z.B.) Falluntersch.
  - falls $a = 0$, dann $b = 1$ wegen $a \vee b$, $c = 1$ wegen $a \vee c$, Widerspruch zu $\neg c \vee \neg b$
  - falls $a = 1$, dann $b = 0$ wegen $\neg a \vee \neg b$, $d = 1$ wegen $b \vee d$, Widerspruch zu $\neg d \vee \neg a$

Vollständigkeit der Resolution

- Satz (Widerlegungsvollständigkeit)
  $M \models \mathsf{False} \Rightarrow M \vdash \mathsf{False}$
- Beweis: $M \models \mathsf{False} \Rightarrow \mathsf{Mod}(M) = \varnothing$.
- Induktion nach $|\mathsf{Var}(M)|$, dabei Induktionsschritt:

  wähle beliebiges $x \in \mathsf{Var}(M)$ und definiere

  $M_0 := M[x := 0], \quad M_1 := M[x := 1]$
  (durch Streichen von Literalen und Klauseln)
- zeige, daß $\varnothing = \mathsf{Mod}(M_0) = \mathsf{Mod}(M_1)$
- nach Induktionsannahme $M_0 \vdash \mathsf{False}, M_1 \vdash \mathsf{False}$
- dann $M \vdash x, M \vdash \neg x$, also $M \vdash x \oplus_x \neg x$

Illustration für Induktionsschritt

- $M = \{a \vee b, \neg a \vee \neg b, a \vee c, \neg c \vee \neg b, b \vee d, \neg d \vee \neg a\}$
- Wähle Variable a. Dann

  $M_0 = \{ \quad b, \qquad\qquad c, \neg c \vee \neg b, b \vee d \qquad\qquad \}$

  (Klauseln C mit $\neg a \in C$ löschen, aus anderen a entfernen)

  $M_1 = \{ \qquad\qquad \neg b, \qquad \neg c \vee \neg b, b \vee d, \neg d \qquad \}$

  (Klauseln C mit $a \in C$ löschen, aus anderen $\neg a$ entfernen)

- Nach Induktion gilt $M_0 \vdash$ False, z.B. durch $\dfrac{c, \dfrac{b, \neg c \vee \neg b}{\neg c}}{\text{False}}$
- Das ist Abl. in M_0, konstruiere daraus Ableitung in M

  durch Hinzufügen von a: $\dfrac{a \vee c, \dfrac{a \vee b, \neg c \vee \neg b}{a \vee \neg c}}{a}$
- entspr. $M_1 \vdash$ False und daraus $M \vdash \neg a$

Anwendungen der Resolution

- die Unerfüllbarkeit einer Klauselmenge M nachweisen durch Angabe einer Ableitung $M \vdash$ False
- nachweisen, daß $M \models F$, durch Beweis der Unerfüllbarkeit von $M \cup \{\neg F\}$, durch $M \cup \{\neg F\} \vdash$ False

  falls F eine Konjunktion von Literalen,
  dann ist $\neg F$ äq. zu einer disjunktiven Klausel.

  falls . . . positiven Literalen, dann . . . Horn-Klausel
- wie findet man die Ableitung $M \vdash$ False schnell?
  - es kann nicht immer schnell gehen, denn das Erfüllbarkeitsproblem ist NP-vollständig
  - es gibt aber erstaunlich gute Heuristiken (SAT-Solver)

Erfüllbarkeit von Hornklauselmengen

- Satz: Die Erfüllbarkeit einer Menge M von Hornklauseln ist in Polynomialzeit entscheidbar.
- Beweis (Algorithmus)
  - fixieren für jede Klausel eine Reihenfolge der Literale, dabei die negativen zuerst.
  - solange wie möglich Resolutionsschritte der Form $x \oplus_x C$, wobei $\neg x$ das linke Literal in C.
  - wenn $M \vdash$ False, dann M nicht erfüllbar, sonst $S :=$ die Menge der abgeleiteten Variablen, Belegung $b := S \mapsto 1, (\mathsf{Var}(M) \backslash S) \to 0$ erfüllt M
- Korrektheit: betr. vollständig/unvollst. reduzierte Klauseln
- Laufzeit: es entstehen $\leqslant |M| \cdot \max_{C \in M} |C|$ Klauseln

Bsp Erfüllbarkeit Hornklauselmenge

- $M = \{\neg a \vee \neg b \vee d, b, \neg b \vee a, \neg a \vee \neg c \vee e, \neg c \vee \neg f\}$
- entspricht Regelmenge
  $\{a \wedge b \rightarrow d, b, b \rightarrow a, a \wedge c \rightarrow e, c \wedge f \rightarrow$ False$\}$
- Resolutionsschritte: $b \oplus_b (\neg b \vee a) = a$,
  $a \oplus_a (\neg a \vee \neg b \vee d) = (\neg b \vee d)$, $a \oplus_a (\neg a \vee \neg c \vee e) = (\neg c \vee e)$,
  $b \oplus_b (\neg b \vee d) = d$
- abgeleitete Variablen $S = \{b, a, d\}$
- Belegung $\{(a, 1), (b, 1), (c, 0), (d, 1), (e, 0), (f, 0)\}$
- Korrektheit:
  - Klauseln $\neg a \vee \neg b \vee d, b, \neg b \vee a$ wurden vollständig reduziert, das jeweils rechte Literal ist wahr
  - $\neg a \vee \neg c \vee e$ unvollst. red. zu $\neg c \vee e$, Literal $\neg c$ wahr

Hausaufgaben für KW 22

1. (autotool) eine Resolutionsableitung für die leere Klausel
2. eine aussagenlogische Modellierung einer Rätselaufgabe, das Ableiten einer Aussage durch Resolution: Serie 3 Aufgabe 3.4 von `http://www.imn.htwk-leipzig.de/~schwarz/lehre/ss17/ki/`

   (Die anderen Aufgaben dieser Serie dürfen Sie sich natürlich auch anschauen.)
3. Die Widerlegungsvollständigkeit der Resolution ist nicht die Umkehrung der Korrektheit der Resolution, denn die Aussage $\forall M : \forall C : M \models C \Rightarrow M \vdash C$ ist nicht wahr. Begründen Sie das durch ein Gegenbeispiel.
4. (Zusatz) Die Erfüllbarkeit eine Menge M von

Krom-Klauseln ist in Polynomialzeit entscheidbar.

Realisieren Sie den folgenden Beweisplan und führen Sie ihn an einem Beispiel vor.

- konstruiere gerichteten Graphen G
  - Knoten $=$ alle Literale ($x, \neg x$ für $x \in \mathsf{Var}(M)$)
  - Knoten $=$ alle Implikationen, die man aus den Klauseln ablesen kann (welche genau?)
- Hilfssatz: nach Konstruktion gilt: wenn $x \rightarrow_G^* y$, dann auch $\neg y \rightarrow_G^* \neg x$.
- Hilfssatz: wenn $x \rightarrow_G^* \neg x \rightarrow_G^* x$, dann M nicht erfüllbar.
- Konstruiere Graphen G' der starken Zusammenhangskomponenten von G.
  Dann gilt: wenn eine Komponente C sowohl x als auch $\neg x$ enthält, dann M nicht erfüllbar.

- Falls keine Komponente mit x und $\neg x$, dann Belegung:
  - es gibt (wenigstens) eine (noch nicht belegte) Komponente S ohne Vorgänger, eine Komponente S' ohne Nachfolger,
  - belege (alle Literale in) S mit 0, belege S' mit 1.
  - wiederhole, bis alle Komponenten belegt sind

  Beweisen Sie, daß dieser Algorithmus durchgeführt werden kann (es gibt S und S') und daß die resultierende Belegung ein Modell ist

Prädikatenlogische Resolution

Motivation, Definition

- bisher Aussagenlogik (AL), jetzt Prädikatenlogik (PL)
- PL ist ausdrucksstärkere Beschreibungssprache:
  - Aussagen über unendliche viele Objekte $\forall x \exists y : x < y$
  - ... über strukturierte Objekte $\forall x \forall y$: first(Pair$(x, y)) = x$
- ist *zu* ausdrucksstark

  (Mod$(M) = \varnothing$, $M \models F$ u.ä. sind nicht entscheidbar)
- Prolog (Alain Colmerauer, 1972) ist PL mit
  - syntaktischer Einschränkung: definite Horn-Klauseln
  - einer fixierten Resolutions-Strategie: SLD-Res.
- verschiedene Erweiterg. (andere Klauseln, andere Strat.)

PL: Syntax (Wiederholung)

- Signatur Σ: Relationssymbole Σ_R, Funktionssymbole Σ_F
- Term: aus Variablen und Funktionssymbolen
- Formel:
  - Atom (Relationssymbol mit Term-Argumenten)
  - Quantor Variable Formel
  - aussagenlogische Kombination von Formeln
- eine Aussage (ein Satz) ist Formel ohne freie Variablen
- Bsp: für $\forall x \forall y (R(x, y) \Rightarrow R(f(x), f(y))) \wedge R(a, b)$
  - Signatur und abstrakten Syntaxbaum angeben,
  - alle Teilterme und Teilformeln markieren
- Literatur: Uwe Schöning: *Logik für Informatiker*, 1995

PL: Semantik (Wiederholung)

- Σ-Struktur: Universum U (nicht leer)

  Abb: $\Sigma_R \to$ Relationen auf U, $\Sigma_F \to$ Funktionen auf U,
- Belegung: Abb. Var $\to U$
- Interpretation: ist Struktur mit Belegung
- Wertfunktion: Formel $\times$ Interpretation $\to$ Bool
- Modell-Relation $I \models F$, Modellmenge Mod(M),
- Bsp: für $\forall x \forall y(R(x,y) \Rightarrow R(f(x), f(y))) \wedge R(a,b)$ ein Modell mit $U = \{1,2\}$ angeben
- Bsp: eine erfüllbare Formel ohne endliches Modell?

  Hinweis: $\forall x \exists y K(x,y) \wedge \ldots$ (geeignete Eigensch. von K)

PL: Äquivalenzen (Wiedeholung)

- Folgerungs-Relation $M \models F$ gdw. Mod$(M) \subseteq$ Mod(F)
- Äquivalenz $F \equiv G$ gdw. Mod$(F) =$ Mod(G)

  (gleichbedeutend: $F \models G$ und $G \models F$)
- alle aussagenlogischen Äquivalenzen gelten auch in PL, z.B. $\neg(F \vee G) \equiv (\neg F \wedge \neg G)$ für beliebige PL-Formeln F, G
- zusätzliche Äquivalenzen, Beispiele:
  - $\neg \forall x F \equiv \exists x \neg F$
  - wenn $x \notin$ Var(F), dann $(F \vee \forall x G) \equiv \forall x (F \vee G)$
  - $(\exists x P(x) \rightarrow P(y)) \equiv \forall x (P(x) \rightarrow P(y))$
- Ü: $(\forall x F \vee \forall x G) \not\equiv \forall x (F \vee G)$,

  $\models$ oder $\models\!\!\!\!=$ zwischen $\forall x \exists y P(x, y)$ und $\exists y \forall x P(x, y)$?

Bereinigte Form

- Def: eine Formel ist *bereinigt*, falls
  - alle Quantoren paarweise verschiedene Variablen binden
  - keine quantifizierte Variable frei vorkommt
- Satz: zu jeder Formel F gibt es eine äquivalente Formel G in bereinigter Form.
- Bsp: bereinige $F = \forall x \exists y P(x, f(y)) \wedge \forall y (Q(x, y) \vee R(x))$
- Beweis: (mehrfache) gebundene Umbenennung (vgl. $\rightarrow_\alpha$)

Pränex-Form

- Def: eine Formel ist in *Pränex-Form*,

  wenn sie die Form $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n . F$ hat,

  wobei $Q_i \in \{\forall, \exists\}$ und F keine Quantoren enthält

- Satz: zu jeder Formel F gibt es eine äquivalente Formel G in bereinigter Pränex-Form.

- Beweis (Induktion über Formelaufbau)

  – Negationen mit de-Morgan nach innen bewegen,
  – in $F_1 \vee F_2$ sowie $F_1 \wedge F_2$ quantif. Variablen disjunkt umbenennen, Quantoren vorziehen

- Bsp: $F = (\forall x \exists y P(x, g(y, f(x))) \vee \neg Q(x)) \vee \neg \forall x R(x, y)$.

Skolem-Form

- Def: eine Formel F ist in Skolem-Form, wenn F in bereinigter Pränex-Form ist und nur All-Quantoren erhält.
- Thoralf Skolem (1887–1963), `http://www-history.mcs.st-and.ac.uk/Biographies/Skolem.html`
- Nicht-Bsp: $F = \forall x \exists y \forall z \exists u R(x, y, z, u)$
- Satz: zu jedem F in bereinigter Pränex-Form gibt es eine *erfüllbarkeitsäquivalente* G in Skolemform.

  (d.h., Mod$(F) \neq \emptyset \iff$ Mod$(G) \neq \emptyset$)
- Beweis: für jeden $\exists$ ein neues Funktionssymbol, Stelligkeit: Anzahl der vorausgehenden $\forall$
- Bsp: $G = \forall x \forall z R(x, f(x), z, g(x, z))$.

Herbrand-Strukturen

- Jacques Herbrand (1908–1931)
  `http://www-history.mcs.st-and.ac.uk/Biographies/Herbrand.html`
- eine Σ-Struktur heißt Herbrand-Struktur, falls
  - Universum $=$ Term(Σ_F) (variablenfreie Terme)
  - falls kein 0-stell. Symbol in Σ_F, dann eines hinzufügen
  - jedes Funktions-Symbol durch sich selbst interpretiert
- Es gibt erfüllbare Aussagen ohne Herbrand-Modell:
  $F_1 = P(a) \wedge \exists x : \neg P(x)$
- Satz: für jede Aussage F in Skolemform:
  F besitzt Modell $\iff$ F besitzt Herbrand-Modell.
- Bsp: diesen Satz für Skolemisierung von F_1 prüfen

Klauselform

- Syntax — Bsp: $\{\{\neg P(x), R(x, f(x))\}, \{P(a), Q(y)\}\}$
  - Formel $=$ Menge von Klauseln
  - Klausel $=$ Menge von Literalen
  - Literal $=$ Atom oder negiertes Atom
- Semantik (durch Übersetzung in PL-Syntax)
  - Formel: Konjunktion über Klauseln,
  - Klausel: Disjunktion über Literale
  - vollständig all-quantifiziert (jede Klausel einzeln oder die ganze Formel)
- Bsp: $\forall x(\neg P(x) \vee R(x, f(x))) \wedge \forall y(P(a) \vee Q(y))$

  Bsp: $\forall x \forall y(\neg P(x) \vee R(x, f(x)) \wedge (P(a) \vee Q(y))$

Hornklauselmengen und Logische Progr.

- definite Klausel $\{\neg A_1, \ldots, \neg A_n, B\}$ (ein positives Literal!)
  bedeutet $\forall x_1 \ldots x_k (A_1 \wedge \ldots \wedge A_n \rightarrow B)$
  Notation in Prolog: `B :- A1 , ... , An .`
  Falls $n = 0$ (Fakt), dann `B .`
- Programm P (Menge von definiten Klauseln)

```
append (nil,Ys,Ys ) .
append (cons(X,Xs),Ys,Cons(Z,Zs))
   :- append (Xs,Ys,Zs) .
```

  Query Q `append(Xs,Ys,cons(a,cons(b,nil)))`
- Prolog-System *widerlegt* Erfüllbarkeit von $P \cup \{\neg Q\}$
  durch PL-Resolution (mit bestimmter Strategie, KW 23)

PL-Resolution (Motivation, Plan)

- (Wdhlg.) bei AL-Resolution der Klauseln C_1 und C_2 gibt es Literal l mit $l \in C_1$ und $\neg l \in C_2$.
- in PL können in den Literalen auch Variablen vorkommen,

  $\Rightarrow$ bei Vergleich der Literale $(l, \neg l)$ berücksichtigen
- Motivation (1): Klausel $\{\neg P(x), R(x, f(x))\}$

  bedeutet $\forall x : (P(x) \rightarrow R(x, f(x)))$

  soll mit $P(a)$ resolviert werden zu $R(a, f(a))$.
- Motivation (2): $\{S(x), R(x, f(y))\} \oplus \{\neg R(g(z), z)\}$ ergibt . . . ?
- Vorgehensweise:

  1. Grund-Instantiierung: (t ist Grund-Term gdw. $\text{Var}(t) = \varnothing$) eine PL-Klausel mit Var. $\Rightarrow$ viele Klauseln ohne Var.
  2. Instantiierung (Substitution) Var $\rightarrow$ Term(Σ_F, Var)

Grund-Instantiierung

- Def: Substitution ist Abb. Var $\to$ Term$(\Sigma_F,$ Var$)$

  Def: Grund-Substitution ist Abb. Var $\to$ Term$(\Sigma_F, \emptyset)$
- Notation: $t\sigma$ bezeichnet Bild des Terms t unter der Substitution σ, entspr. $F\sigma$ Bild der Formel F unter σ
- Satz: jede Skolemformel $\forall x_1 \ldots \forall x_k G$ ist äquivalent zu der (mglw. unendlichen) Menge von Grund-Formeln

  Ground$_\Sigma(G) := \{G\sigma \mid \sigma \in$ Var$(G) \to$ Term$(\Sigma_F)\}$

  (Substitution in das Herbrand-Universum)
- Bsp: $\Sigma_F = \{a/0, f/1\}$, $G = \forall x R(x, f(x))$, Ground$_\Sigma(G) = \{R(a, f(a)), R(f(a), f(f(a))), \ldots, R(f^k(a), f^{k+1}(a)), \ldots\}$
- jedes $f \in \Sigma_F$ ist 0-stellig $\Rightarrow$ Term(Σ_F) ist endlich

Von PL zu AL durch Grund-Instantiierung

- $G' :=$ jedes Atom in $\text{Ground}_\Sigma(G)$ als eine aussagenlogische Variable betrachten
- Herbrand-Modell von G: Interpretation der Relations-Symbole

  $\Leftrightarrow$ AL-Modell von G' durch entsprechende Belegung dieser Variablen
- Satz: AL-Resolution auf Grund-Formeln ist korrekt und widerlegungs-vollständig
- wir hatten W-Vollst. nur für endliche $\text{Var}(M)$ bewiesen,

  Behauptung folgt aus *Endlichkeitssatz (der AL)*:

  $\text{Mod}(M) = \varnothing \Rightarrow \exists M' \subseteq M : |M'| < \infty \wedge \text{Mod}(M') = \varnothing$.

PL-Resolution (Beispiel)

- Wdhlg.: Grund-Instantiierung und AL-Resolution:
  - wähle Grund-Substitution σ,
  - Resultat ist $C_1\sigma \oplus_x C_2\sigma$,

  Bsp: $C_1 = \{S(x), R(x, f(y))\}, C_2 = \{\neg R(g(z), z)\}$,

  $\sigma = \{(x, g(f(a))), (y, a), (z, f(a))\}$, $C_1\sigma \oplus C_2\sigma = \{S(g(f(a)))\}$.
- jetzt beliebige Substitutionen , Bsp.

  $\tau = \{(x, g(f(y))), (z, f(y))\}$

  $C_1\tau = \{S(g(f(y))), R(g(f(y)), f(y))\}$,
  $C_2\tau = \{\neg R(g(f(y)), f(y))\}$.

  Resolution ergibt $C_1\tau \oplus C_2\tau = \{S(g(f(y)))\}$.

PL-Resolution (Definition, Beispiel)

- Definition
  - für Klauseln C_1, C_2, Umbenennungen τ_1, τ_2 mit $\mathsf{Var}(C_1\tau_1) \cap \mathsf{Var}(C_2\tau_2) = \emptyset$,
    Atom l, Substitution σ (deren Def.- und Bildbereich keine gemeinsame Variablen enthalten),
  - falls $l \in C_1\tau_1\sigma$ und $\neg l \in C_2\tau_2\sigma$,
  - dann $C_1 \oplus_{\tau_1,\tau_2,\sigma,l} C_2 := (C_1\tau_1\sigma \backslash \{l\}) \cup (C_2\tau_2\sigma \backslash \{\neg l\})$.
- Beispiel: vorige Folie, mit $C_1 \oplus_{\mathsf{id},\mathsf{id},\tau,R(g(f(y)),f(y))} C_2$.
- Satz: PL-Resol. ist korrekt und widerlegungsvollständig.
- Beweisidee: benutzt *lifting lemma*: C_1, C_2 beliebig, $G_1 \in \mathsf{Ground}(C_1), G_2 \in \mathsf{Ground}(C_2)$, AL-Res: $R = G_1 \oplus_l G_2$.
  Dann ex. σ mit $R \in \mathsf{Ground}(C_1 \oplus_{\sigma,l'} C_2)$ (PL-Res)

Hausaufgaben für KW 23

siehe `http://www.imn.htwk-leipzig.de/~schwarz/lehre/ss17/ki/` und Kap 2.5 aus U. Schöning: *Logik für Informatiker*

1. KI 17 Aufgabe 5.1 (2. und 3.: erfüllbarkeitsäquivalent) (oder eine ähnliche Aufgabeninstanz) live an der Tafel
2. KI 17 Aufgabe 5.2
3. (bis auf Umbenennungen) alle Resolventen von:

   $C_1 = \{\neg P(x, y), \neg P(f(a), g(u, b)), Q(x, u)\}$

   $C_2 = \{P(f(x), g(a, b)), \neg Q(f(a), b), \neg Q(a, b)\}$
4. (autotool) Beispiel PL-Resolution

Logische Programmierung

Wiederholung, Motivation

- Resolution beweist $M \cup \{\neg Q\} \vdash$ False, also Mod$(M \cup \{\neg Q\}) = \varnothing$, also $M \models Q$
- Wenn Q freie Variablen $x_1, \ldots$ enthält, dann bedeutet $Q : \exists x_1 \ldots . F$, (damit $\neg Q$ eine Skolem-Klausel ist)

  Resolution liefert Belegung(en) σ, so daß $M \models F\sigma$
- man legt eine Suchstrategie für Ableitungen fest:
  - top-down (ausgehend von Q): Prolog
  - bottom-up (ausgehend von Fakten in M): Datalog

Lineare Resolution

- lineare Resolution erzeugt Folge $C_0, C_1, \ldots, C_n =$ False, wobei
  - $C_0 = \neg Q$ (negierte Anfrage),
  - $\forall i \geqslant 0 : C_{i+1}$ durch Resolution von C_i mit einer Programmklausel (definiten Klausel)
- beachte: alle C_i sind negative Klauseln
- wenn schließlich $C_n =$ False,

  dann $\sigma :=$ das Produkt der bis dahin angewendeten Substitutionen
- Bsp: $M = \{\{R(a,b)\}, \{\neg R(X,Y), R(Y,X)\}\}$, $Q = \exists Z.R(b,Z)$

  $C_0 = \{\neg R(b,Z)\}, C_1 = \{\neg R(Z,b)\}, C_2 = \varnothing$, $\sigma = \{(Z,a)\}$
- Satz: lineare Resolution ist vollständig für Hornklauseln

Die DFS-Strategie

- ergibt eine bestimmte lineare Resolutions-Ableitung
- schreibe Klausel als *Folge* von Literale, Programm als *Folge* von Klauseln,
- resolviere immer das linke Literal aus C_i mit der ersten M-Klausel, deren Kopf (positives Literal) dazu paßt
- wobei die Subst. nur die nötigen Einschränkungen enthält
- füge neue Literale (aus der substitutierten Programmklausel) links ein (dort wird weiter resolviert, die Literale der aktuellen Klausel bilden einen Stack)
- Bsp: $M = [[R(Y,X), \neg R(X,Y)], [R(a,b)]]$, $Q = \exists Z.R(b,Z)$ $C_0 = [\neg R(b,Z)], C_1 = \ldots$
- Satz: Lineare Resol. mit DFS-Strat. ist *nicht vollständig*.

Substitutionen

- Signatur $\Sigma = \Sigma_0 \cup \ldots \Sigma_k$,
- Term(Σ, V) ist kleinste Menge T mit $V \subseteq T$ und $\forall 0 \leqslant i \leqslant k, f \in \Sigma_i, t_1 \in T, \ldots, t_i \in T : f(t_1, \ldots, t_i) \in T$.
- Substitution: partielle Abbildung $\sigma : V \to$ Term(Σ, V), so daß kein $v \in \mathrm{dom}\,\sigma$ in $\mathrm{img}\,\sigma$ vorkommt,
- Substitution σ auf Term t anwenden: $t\sigma$
- Bsp: $\sigma = \{(X, a), (Y, f(Z))\}$, $(g(Y, X))\sigma = g(f(Z), a)$.

Produkt von Substitutionen

- Produkt von Substitutionen soll diese Eigenschaft haben:
  $t(\sigma_1 \circ \sigma_2) = (t\sigma_1)\sigma_2$
- Bsp: $\{(X, Y) \circ \{(Y, a)\}$

  ist *nicht* $\{(X, a)\}$, denn . . . , sondern . . .
- $\tau_1 = \{(X, Y)\}, \tau_2 = \{(Y, X)\}, \tau_1 \circ \tau_2 = \tau_2$
- $\circ$ ist assoziativ. Kommutativ? Neutrales Element?

Vergleich von Substitutionen

- Relation auf Substitutionen (σ_1 ist allgemeiner als σ_2)

  $\sigma_1 \lesssim \sigma_2 : \Longleftrightarrow \ \exists \tau : \sigma_1 \circ \tau = \sigma_2$

- $\lesssim$ ist transitiv. ($\lesssim$ ist Prä-Ordnung)
- $\lesssim$ ist nicht antisymmetrisch ($\lesssim$ ist keine Halbordnung)
- Bsp: $\{(X, Y)\} \lesssim \{(X, a), (Y, a)\}$, $\{(X, Y)\} \lesssim \{(Y, X)\}$, $\{(Y, X)\} \lesssim \{(X, Y)\}$.
- Relation $\sim := (\lesssim) \cap (\gtrsim)$ ist Äquivalenz-Relation
- Bsp: $\{(X, Y)\} \sim \{(Y, X)\}$
- Def: σ_1 und σ_2 *bis auf Umbennenung gleich*: $\sigma_1 \sim \sigma_2$

Das Unifikationsproblem (Definition)

- Unifikationsproblem
  - Eingabe: Terme $t_1, t_2 \in \text{Term}(\Sigma, V)$
  - Ausgabe: eine allgemeinster Unifikator (mgu): Substitution σ mit $t_1\sigma = t_2\sigma$.
- „allgemeinst“ $=$ minimal bzgl. der Prä-Ordnung $\lesssim$
- Satz: jedes Unifikationsproblem ist entweder gar nicht oder bis auf Umbenennung eindeutig lösbar
- Beispiel: $t_1 = f(a, f(X, Y)), t_2 = f(X, Z)$

  Unifikatoren $\sigma_1 = \{(X, a), (Y, a), (Z, f(a, a))\}$,

  $\sigma_2 = \{(X, a), (Z, f(a, Y))\}$,

  $\sigma_3 = \{(X, a), (Z, f(a, W)), (Y, W)\}$, mit $\sigma_3 \sim \sigma_2 < \sigma_1$
- Beweis: durch Angabe eines Algorithmus

Ein Algorithmus zur Unifikation

- mgu(s, t) nach Fallunterscheidung
  - $s \in$ Var: falls $s \notin$ Var(t), dann $\{(s, t)\}$, sonst unlösbar
  - $t \in$ Var: symmetrisch
  - $s = f(s_1, s_2)$ und $t = g(t_1, t_2)$:
    falls $f = g$,
    dann bestimme $\sigma =$ mgu(s_2, t_2),
    dann $\tau =$ mgu$(s_1\sigma, t_1\sigma)$,
    Resultat ist $\sigma \circ \tau$.
- Beispiel: $s = f(a, f(X, Y)), t = f(X, Z)$,
  - $\sigma =$ mgu$(f(X, Y), Z) = \{(Z, f(X, Y))\}$.
  - $\tau =$ mgu$(s\sigma, t\sigma) =$ mgu$(a, X) = \{(X, a)\}$.
  - $\sigma \circ \tau = \{(Z, f(a, Y)), (X, a)\}$.

Algorithmus zur Unifikation (Eigensch.)

- korrekt, übersichtlich, aber nicht effizient,
- es gibt Unifikations-Probleme mit exponentiell großer Lösung,
- komprimierte Darstellung benutzt Term-DAGs statt Term-Bäumen (d.h., mit sharing)
- diese kann man in Polynomialzeit ausrechnen.
- siehe Kapitel 4.8 in:

  Franz Baader, Tobias Nipkow: *Term Rewriting and All That*, Cambridge Univ. Press, 1998.

Beispiel f. Prolog-Programm

- Verkettung von einfach verketteten Listen.

  Programm: (in `app.pl`)

```
app(nil,XS,XS).
app(cons(X,XS),YS,cons(X,ZS))
  :- app(XS,YS,ZS).
```

  Hinweis: nicht `append`, weil das evtl. vordefiniert ist

- Query: (nach `swipl -l app.pl` oder `gprolog --consult-file app.pl`)

```
?- app(X,Y,cons(a,cons(b,nil))).
```

- weitere Lösungen mit Semikolon

Datalog (Definition, Beispiel)

- wir betrachten nur Programme mit 0-stelligen Funktionssymbolen (das Herbrand-Universum U ist endlich, besteht aus genau den F-Symb.)
- `R(a,b). R(a,c). R(c,d). R(d,e).`
  `R(X,Z) :- R(X,Y), R(Y,Z).`
- Jedes Rel.-Symbol interpretiert durch Relation auf U, kann als Tabelle einer Datenbank aufgefaßt werden
- Die Regeln entsprechen Joins (d.h., Produkten und Projektionen von Relationen), durch die neue Zeilen zu Tabellen hinzugefügt werden
- man kann bottom-up *alle* Ableitungen durchführen,
- und dann sehr schnell jede Query beantworten

Datalog (Semantik)

- Der Konsequenz-Operator Con_M eines Programms M:
  $\mathsf{Con}_M(A) :=$ Menge aller Köpfe aller Regel-Instanzen, für die alle (instantiierten) Rumpf-Klauseln $\in A$ sind.
- `R(a,b). R(a,c). R(c,d). R(d,e).`
  `R(X,Z) :- R(X,Y), R(Y,Z).`
  $\mathsf{Con}_M(\emptyset) = \{R(a,b), R(a,c), R(c,d), R(d,e), R(a,d), R(c,e)\}$
  $\mathsf{Con}_M(\mathsf{Con}_M(\emptyset)) = \ldots$
- Satz: Con_M ist monoton: $A \subseteq B \Rightarrow \mathsf{Con}_M(A) \subseteq \mathsf{Con}_M(B)$.
- Satz: die Folge $A_0 = \emptyset, A_{i+1} = \mathsf{Con}_M(A_i)$ ist monoton steigend und nach oben beschränkt (wodurch?)
- . . . und besitzt einen Fixpunk $A_k = A_{k+1} = \cdots = \mathsf{Con}^*_M(\emptyset)$
- Satz: Für Atome F gilt: $M \models F$ gdw. $F \in \mathsf{Con}^*_M(\emptyset)$

Beispiel: Affe, Stuhl, Banane (Version 1)

- $p(X, Y, Z)$ soll bedeuten: Affe in Position X, Stuhl auf Position Y, Banane auf Position Z (an der Decke)
- Programm:
  - $p(a, b, c)$ Startsituation.
  - $p(W, Y, Z) \leftarrow p(X, Y, Z)$. Der Affe kann zu jeder Position laufen.
  - $p(W, W, Z) \leftarrow p(X, X, Z)$. Der Affe kann den Stuhl mitschieben.
- Query $p(X, X, X)$. Der Affe steigt auf den Stuhl, um die Banane zu erreichen.
- Aufgabe: wird $M \cup \{\neg\exists X.Q\} \vdash$ False mit Prolog-Strategie gefunden? $M \vdash \exists X.Q$ mit Datalog-Stragie?

Beispiel: Affe, Stuhl, Banane (Version 2)

- wenn man $M \cup \neg Q \vdash$ False hat, weiß man zwar, *daß* das Ziel erreicht werden kann, aber nicht, *wie*.
- wir erweitern das Modell zu $p(X, Y, Z, F) =$ Affe in X, Stuhl in Y, Banane in Z, erreicht durch Aktionsfolge F.
- Programm
  - $p(a, b, c, \mathsf{start})$.
  - $p(W, Y, Z, \mathsf{walk}(X, W, F)) \leftarrow p(X, Y, Z, F)$.
  - $p(W, W, Z, \mathsf{push}(X, W, F)) \leftarrow p(X, X, Z, F)$.

  Query $p(X, X, X, F)$.
- das ist nun aber kein Datalog mehr

Hausaufgaben für KW 24

Hinweise:

- Prolog: im Pool installiert sind `swipl` (`http://www.swi-prolog.org/`) und `gprolog` (`http://gprolog.org/`)
- autotool-Aufgaben zu Prolog/Datalog: noch nicht fertiggestellt

1. (autotool) Aufgabe zu Unifikation
2. Zur Implementierung des Produktes von Substitutionen `https://gitlab.imn.htwk-leipzig.de/autotool/all0/blob/master/collection/src/Prolog/Substitution.hs#L55`

(a) ist das korrekt?

(b) kann man das verkürzen?

3. Vorführen (am Computer) und erklären (an der Tafel): für das Programm

```
plus(zero,Y,Y).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(zero,Y,zero).
times(s(X),Y,Z) :- times(X,Y,W), plus(Y,W,Z
```

(a) Wie kann man mit diesem Programm subtrahieren?
(b) was passiert bei Anfrage (und dann mehrfach `;`)
`times(X,Y,s(s(s(s(s(s(zero)))))))`?

4. Aufgabe 7.2 von `http://www.imn.htwk-leipzig.de/~schwarz/lehre/ss17/ki/`, zu Teil 2 (Fixpunktsemantik) siehe Datalog

Bayes-Netze

Motivation, Definition

- Bayes-Netz (alternativ: *believe network*) ist DAG
  - Knoten: Zufallsvariablen
  - Kanten: (vermutete) kausale (ursächliche) Beziehungen
- Anwendung: probabilistisches Schließen,
  Bestimmung wahrscheinlicher Ursachen für Symptome
- BN erfunden von Judea Pearl, erhielt (u.a.) dafür den *ACM Turing Award* 2011, `https://amturing.acm.org/award_winners/pearl_2658896.cfm`
- benannt nach Thomas Bayes (1701–1761),
  Satz von Bayes über bedingte Wahrscheinlichkeiten

(Wdhlg) Wahrscheinlichkeiten

- Begriffe
  - Zufalls-Experiment
  - (endlicher) Wahrscheinlichkeitsraum $(\Omega, 2^{\Omega}, P)$
  - Elementar-Ereignis
  - zufälliges Ereignis,
- Beispiele
  - Experiment: dreimal würfeln,
  - Ereignis V: Augenzahlen sind paarweise verschieden,
  - Elementar-Ereignisse: $\{(x, y, z) \mid x, y, z \in \{1, \ldots, 6\}\}$
  - $P(V)$ bei Gleichverteilung?

(Wdhlg) Bedingte Wahrscheinlichkeit

- Def: bedingte Wsk. von Ereignis A unter Ereignis B:

  $P(A \mid B) := P(A \cap B)/P(B)$

- Bsp: zwei Würfel, $A =$ Augensumme ist > 7,
  $B =$ beide Zahlen sind ungerade.

- Bsp: B eine Ursache (für Fehler, Krankheit, usw.),
  A eine Auswirkung (Symptom) (leichter zu beobachten)

- vergleiche zu bisher betrachteten Regelsystemen

  neu ist jetzt, daß keine Aussagen über Wahrheit, sondern über Wahrscheinlichkeit getroffen werden.

(Wdhlg) Satz von Bayes

- Satz (einfache Form): $P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A)$.
- Beweis: Def. von $P(X \mid Y)$ einsetzen, vereinfachen.
- Anwendung: Rechnen mit bedingten Wsk.
  - 1/3 aller Studenten haben ein Notebook.
  - 1/10 aller Studenten studieren Informatik.
  - 9/10 aller Informatik-Studenten haben ein Notebook.
  - Sie sehen einen Studenten mit einem Notebook.
  - Mit welcher Wahrscheinlichkeit studiert er Informatik?
- Das ist ein Beispiel für probabilistische Inferenz.

  wird verallgemeinert auf längere Ketten von Ursache-Wirkung-Beziehungen

(Whdlg) Unabhängige Ereignisse

- Def: Ereignisse A, B heißen (stochastisch) unabhängig, falls $P(A \cap B) = P(A) \cdot P(B)$.
- Satz: $P(B) > 0 \Rightarrow$ (A und B unabh. $\iff P(A \mid B) = P(A)$).
- Bsp: zwei Würfel, $A =$ Augensumme ist > 7, $B =$ beide Zahlen ungerade. A und B sind *nicht* unabhängig.
- Def: Nicht unabhängige A, B heißen *korreliert*.

  Vorsicht: das bedeutet nicht, daß A die Ursache für B ist, oder B die für A. Es könnte z.B. eine gemeinsame Ursache C für A und B geben.

  Bsp: $A =$ tiefe Temperatur, $B =$ hoher Umsatz, $C = \ldots$

  (correlation does not imply causation)

(Wdhlg) Diskrete Zufallsgrößen

- Def: Zufallsgröße ist Funktion $X : \Omega \to$ endl. Menge ($\subseteq \mathbb{R}$)
- einfachster Fall: $\Omega = \mathbb{B}^k$ mit $\mathbb{B} = \{\text{False}, \text{True}\} = \{0, 1\}$.
  $X_k = (\vec{x} \mapsto \vec{x}_k)$ (die k-te Komponente)
- dann Wsk-Raum bestimmt durch Wsk der Elementar-E.,
  Bsp: $P(0,0) = 1/3, P(0,1) = 1/6, P(1,0) = 0, P(1,1) = 1/2$
- (Motivation für Bayes-Netz: beschreibt solchen Wsk-Raum durch deutlich weniger als 2^k Parameter)
- zu Zufallsgröße X betrachte Ereignis $X = e$,
  Bsp (Fortsetzung): $P(X_1 = \text{False} \cap X_2 = \text{True}) = 1/6$.
  $P(X_2 = \text{True}) = 1/6 + 1/2 = 2/3$, $P(X_1 = \text{False}) = \ldots$
- Def. Zufallsgrößen X, Y sind unabhängig:
  jedes $X = e$ ist unabhängig von jedem $Y = f$

Definition Bayes-Netz

- Syntax: ein Bayes-Netz N ist ein Paar (G, Θ) mit
  - G ist DAG, Knoten sind Zufallsgrößen
  - Θ: für jeden Knoten X mit Eltern $X_1, \ldots, X_k$: Wahrscheinlichkeiten $P(X = e \mid X_1 = e_1 \cap \cdots \cap X_k = e_k)$ für alle $[e, e_1, \ldots, e_k] \in W^{k+1}$
- Semantik: N beschreibt Wsk-Raum durch

$$P(X = e) = P(X = e \mid \ldots X_k = e_k \ldots) \cdot \prod_k P(X_k = e_k)$$

das ist induktive Definition,

I.-Anfang sind die Quellen des DAG (ohne Vorgänger, d.h., ohne Bedingungen, d.h., $\prod \varnothing = 1$)

Beispiel Bayes-Netz

- aus Stuart Russel/Peter Norvig *Artif. Intell.* Abschnitt 14.1, dort zitiert nach Judea Pearl
- Knoten: Einbruch, Erdbeben, Alarmanlage (zuhause), John ruft (auf Arbeit) an, Mary ruft an.
- Kanten mit Parametern (Bsp)
  - $P(E = 1 \mid) = 0.002$
  - $P(A = 1 \mid R = 0, E = 1) = 0.29, \ldots$
- vollständig siehe `https://gitlab.imn.htwk-leipzig.de/autotool/all0/blob/master/collection/src/Bayes/Data.hs#L167`

Bedingte Unabhängigkeit und BN

- (Wdhlg.) Def A und B unabhängig, falls $P(A \cap B) = P(A) \cdot P(B)$.
- Def: A und B bedingt unabhängig bezüglich C:

  $P(A \cap B \mid C) = P(A \mid C) \cdot P(B \mid C)$.

  (Vorstellung: wir schränken den Wsk-Raum ein auf die Elementar-Ereignisse aus C, verwenden dort die Standard-Def. der Unabh.)
- Def: bedingte Unabh. von (diskreten) Zufallsgrößen entsprechend
- Satz: für jedes BN N, für alle $X, Y \in N$ mit $X \not\rightarrow^*_N Y$: X und Y sind bedingt unabh. bezüglich der Eltern von X.

(automatisierbare) Aufgaben zu BN

- das geht immer, wenn man ein syntaktisches Objekt (hier: Netz N) und eine Semantik-Funktion S hat:
  - autotool würfelt ein Netz N_0,
  - Aufgabenstellung ist $S(N_0)$,
  - Lösung ist ein Netz N_1 mit $S(N_1) = S(N_0)$
- Varianten:
  - zusätzliche Einschränkungen für N_1
    z.B. Knotenreihenfolge
  - Abschwächung zu $|S(N_1) - S(N_2)| \leqslant \epsilon$
  - Highscore-Wertung
    was ist geeignetes Größenmaß? z.B. $\sum_v 2^{\mathsf{indegree}(v)}$

Inferenz mit BN

- die Diagnose-Aufgabe: gegeben ein BN, gesucht sind bedingte Wahrscheinlichkeiten der Ursache(n), unter der Bedingung von Beobachtungen
- Bsp: $P(\text{Einbruch} = 1 \mid \text{John} = 1 \cap \text{Mary} = 1)$
- kann exakt bestimmt werden, dauert jedoch $2^{|N|}$

  kann nicht besser gehen, weil aussagenlog. Erfüllbarkeit auf dieses Inferenzproblem reduziert werden kann
- die Alternative sind schnellere (Simulations)Verfahren, die einen Näherungswert liefern

Einfaches Sampling für BN

- Würfeln (sampling) für eine diskrete Zufallsgröße mit Wertebereich $W = \{w_1, \ldots, w_n\}$:
  - bestimmte Partialsummen $S_k = \sum_{i \leqslant k} P(X = w_i)$,
  - ist monotone Folge mit $S_0 = 0, S_n = 1$
  - würfle v gleichverteilt aus Interval $[0, 1]$
  - bestimme k mit $S_{k-1} \leqslant v < S_k$, Resultat ist w_k.
- in topologischer Ordnung des DAG: für Knoten X mit Vorgängern $X_1, \ldots, X_k$:

  Werte e_1 für $X_1, \ldots$ sind schon erzeugt (Induktion), würfle Wert e für X bzgl. Verteilung $P(\cdot \mid X_1 = e_1 \cap \ldots)$.
- für die Versuche, in denen Belegung der Beobachtungen wie erwartet, zähle Belegungen der Ursachen

Lernen von BN

- die allgemeine Form der Aufgabe ist:
  - gegeben: eine Verteilung V, eine Menge M von BN
  - gesucht: ein $N \in M$, das V (möglichst gut) repräsentiert
- Spezialfall: DAG G von N ist gegeben (z.B. als Expertenwissen)

  Parameter Θ sind gesucht.

Hausaufgaben für KW 25

1. Erklären Sie `https://xkcd.com/552/`
2. X und Y spielen dieses Spiel:

   - X versteckt einen Schatz in Kiste A, B oder C.
   - Y zeigt auf eine Kiste K.
   - X öffnet eine leere Kiste $L \neq K$
   - Y zeigt auf eine Kiste K' (es könnte $K' = K$ sein)
   - wenn K' den Schatz enthält, hat Y gewonnen.

   Bestimmen Sie die Erfolgsaussichten der Strategien

   - niemals umentscheiden (d.h., immer $K' = K$)
   - immer umentscheiden (d.h., immer $K' \neq K$ und $K' \neq L$)

   Wer das Resultat nicht glaubt, programmiert eine

Simulation.

3. (autotool) Aufgabe zu Bayes-Netzen

4. (live in der Übung oder für KW 26) aus aktuellem Anlaß (und nach Russel/Norvig Kap. 14)

   Drei Mannschaften A, B, C spielen jeder gegen jeden einmal. Mögliche Ergebnisse sind Gewonnen, Unentschieden, Verloren. Jede Mannschaft hat eine fixierte, aber unbekannte Spielstärke $\in \{0, 1, 2\}$. Jedes Spielresultat hängt probabilistisch von der Differenz der Spielstärken ab.

   (a) Modellieren Sie das als Bayes-Netz. Setzen Sie plausible numerische Werte ein.
   (b) A gewinnt gegen B, B spielt gegen C unentschieden.

Wie geht das Spiel A gegen C aus? (Geben Sie die bedingte Wahrscheinlichkeitsverteilung an.)

Hinweis: Bayes-Netz-Beispiele im Autotool benutzen: siehe

`https://gitlab.imn.htwk-leipzig.de/autotool/all0/issues/541`

Statistische Textanalyse

Definition, Motivation

- was passiert technisch? statistische Analyse:
  - Text $=$ Folge von Wörtern $\in$ Wort*
  - Context$_k$ $=$ Wort^k $\times$ Wort^k (linker, rechter Kontext)
  - Korrelationsmatrix M : Text $\times$ Context : $(w, c) \mapsto \frac{P(w \cap c)}{P(w) \cdot P(c)}$
  - $M \approx M_W \circ M_C$, $M_W \in \mathbb{R}^{|\text{Text}| \times d}$, $M_C \in \mathbb{R}^{d \times |\text{Context}|}$ für d klein bestimme Koeffizienten v. M_W, M_C durch Lernverfahren
  - Semantik von $w \in$ Text $\approx$ Zeile w in M_W
- was passiert *nicht*: syntaktische, semantische Analyse
- Anwendungen (offensichtlich): bessere Text-Indizierung
- . . . (nicht o.): Erkennen von Analogien (Wortpaaren)

Quellen

- Tomas Mikolov et al.: *Efficient Estimation of Word Representations in Vector Space*, CoRR, https://arxiv.org/abs/1301.3781
- Tomas Mikolov et al.: *Distributed Representations of Words and Phrases and their Compositionality*, NIPS 2013, https://papers.nips.cc/paper/5021-distributed-representations-of-words-a
- Omar Levy, Yoav Goldberg: *Neural Word Embedding as Implicit Matrix Factorization*, NIPS 2014, https://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matr
- (Fork der) Original-Quelltexte:

https://github.com/dav/word2vec

- Quelltexte zur dieser VL: https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss18/ word-vec

Wörter und Kontexte

- Text $T = [w_1, \ldots, w_n] \in W^*$
- k-Kontexte $= W^k \times W^k$
- $D := \{(w_i, c_i) \mid c_i = ([w_{i-k}, \ldots, w_{i-1}], [w_{i+1}, \ldots, w_{i+k}])\}$

  D ist Multimenge von Paaren
- $D(w, c) = f$ bedeutet: w kommt f-mal im Kontext c vor
- Bsp: $T = [a, b, c, b, c, b, a, a, b]$, $k = 1$,

  $D = \{((b, (a, c)), 1), ((c, (b, b)), 2), ((b, (c, c)), 1), \ldots\}$
- betrachten D als Wsk-Raum, definiert Häufigkeiten von Wörtern, Kontexten und Wort-Kontext-Paaren

  Bsp. $P_W(c) = 2/7$, $P_C(b, b) = 2/7$, $P_{W,C}(c, (b, b)) = 2/7$.

Die Korrelationsmatrix

- die PMI-Matrix (pointwise mutual information)

  $$M(w, c) = \frac{P(w \cap c)}{P(w) \cdot P(c)}$$

- Bsp: $T = [a, b, c, b, c, b, a, a, b]$, $k = 1$,

  $M(c, (b, b)) = \frac{2/7}{2/7 \cdot 2/7} = 7/2$, $M(b, (c, c)) = ?$, $M(a, (c, c)) = ?$

- diese Matrix hat Dimension $|W| \times |C|$.

  in echten Anwendungen: $10^6 \leqslant |W| \leqslant |C|$

  man kann M nicht direkt verwenden (Speicherplatz!)

- man bestimmt eine komprimierte Form,

  aus der man noch weitere Informationen ablesen kann (besser als aus M)

Faktorisierung der PMI-Matrix

- komprimierte Form von M besteht aus M_W, M_C

  für $M_W \in \mathbb{R}^{|W| \times d}, M_C \in \mathbb{R}^{d \times |C|}$ mit $d \ll |W|, |C|$ (Bsp: 1000)
- so daß $M_W \circ M_C \approx M$

  diese Bedingung kann man aber nicht komplett prüfen,

  sonder nur für zufällig gewählte (w, c).
- man bestimmt $p(w, c) := M_W(w) \circ M_C(c)$

  (das ist ein Skalarprodukt: Zeilen- mal Spaltenvektor)
- für $(w, c) \in D$ soll $p(w, c)$ groß sein, für $(w, c) \notin D$ klein.
- . . . und wenn das nicht so ist, dann verschiebt man $M_W(w)$ und $M_C(c)$ etwas in die passende Richtung

Verbessern einer Faktorisierung

- die Aufgabenstellung ist:
  - gegeben: Zahlen $p, q, d \in \mathbb{N}$, Matrix $M \in \mathbb{R}^{p,q}$,
  - gesucht: Matrizen $A \in \mathbb{R}^{p,d}$, $B \in \mathbb{R}^{d,q}$ mit $A \circ B \approx M$
- ein Lösungsverfahren:
  - wähle A_0, B_0 mit zufälligen Einträgen, dann wiederhole:
  - wähle w, c zufällig, bestimme $p(w, c) = M_W(w) \circ M_C(c)$
  - falls $M(w, c)$ groß, aber $p(w, c)$ klein:
    nähere die Vektoren $M_W(w), M_C(c)$ einander an
  - falls $M(c, w)$ klein, aber $p(w, c)$ groß:
    vergrößere Unterschied zwischen $M_W(w), M_C(c)$
- mit geeigneter Lernrate (klein $\Rightarrow$ konvergiert langsam,
  groß $\Rightarrow$ konvergiert evtl. gar nicht, sondern oszilliert)

Skalierung

- das ist der nachträgliche Versuch einer systematischen Begründung (von Levy und Goldberg) für die vorausgegangene Bastelei (von Mikolov et al.)
- es geht um Wahrscheinlichkeiten ($0 \leqslant p \leqslant 1$),

  aber die Skalarprodukte sind beliebig.

  Benutze die (monotone, symmetrische, diff-bare) Funktion $s : \mathbb{R} \to (0,1) : x \mapsto 1/(1+\exp(-x))$
- Ü: bestimme 1. Ableitung von s, begründe Monotonie, beweise $s(-x) = 1 - s(x)$ (Punkt-Symm. bzgl. $(0, 1/2)$)
- Def $p(w,c) := s(M_W(w) \circ M_C(c))$

Vereinfachte Kontextmodelle

- bisher: Context$_k$ = Wort^k $\times$ Wort^k.
- Vereinfachung (CBOW - continuous bag of words, Skip-Gram)

  Context = Wort, $D = \{(w_i, w_{i+d}) \mid -k \leqslant d \leqslant k\}$
- damit wird nur noch gemessen, ob zwei Wörter nahe beeinander vorkommen
- weitere Modifikationen:
  - häufigste Wörter weglassen (der, die, das, und, . . .)
  - seltenste Wörter weglassen
- das Modell ist sowieso schon brutal, dann stören diese Modifikationen auch nicht weiter

Kann man so Texte verstehen?

- Verstehen eines Textes T in Sprache S geht so:
  - mit Grammatik G für S konstruiere (erst konkreten, dann abstrakten) Syntaxbaum für T
  - aus Semantik der Wörter von S konstruiere Semantik dieses Baums (Semantik-Funktion ist ein `fold`).
- das hier gezeigte statistische Modell benutzt nur Häufigkeiten gemeinsamer Vorkommen von Wörtern
- vergleiche: Student in der Prüfung:
  - Frage: . . . Relation . . . , Antwort: . . . symmetrisch . . .
  - Frage: . . . Menge . . . , Antwort: . . . $\{\{\ldots\}\ldots\}$ . . .
- rein statisches Textverstehen ist fragwürdig . . . aber (b.w.)

Erkennen von Analogien

- das hier betrachtete Vorgehen bestimmt eine Abbildung M_W : Wort $\to \mathbb{R}^d$ (die Zeilen der Matrix M_W)
- Wörter mit $M_W(u) \approx M_W(v)$ sind oft austauschbar $\Rightarrow$ haben ähnliche Bedeutung (das ist klar)
- (das ist überraschend) Wortpaare $(u_1, v_1), (u_2, v_2)$ mit $M_W(u_1) - M_W(v_1) \approx M_W(u_2) - M_W(v_2)$ sind Analogien

  (u_1 verhält sich zu v_1 wie u_2 zu v_2)

  Bsp: betrachte Kontext „. . . ist die Hauptstadt von. . .“

  $u_1 =$ Rom, $v_1 =$ Italien, $u_2 =$ Paris, $v_2 =$ Frankreich
- wenn u_1, v_1, v_2 gegeben, dann $u_2 =$ ein Wort mit Vektor nahe bei $M_W(u_1) - M_W(v_1) + M_W(v_2)$.

Warum funktioniert das?

- Durch Dimensions-Reduktion ($|M| > 10^6, d \approx 10^3$)

  wird das Verfahren gezwungen, Wörter zu *klassifizieren*

  (austauschbare, d.h., bedeutungsähnliche, Wörter

  auf benachbarte Vektoren abzubilden)

- ohne Reduktion hat das Optimierungsproblem

  die triviale Lösung: Wort w_i auf Vektor $[0, \ldots 0, 1, 0 \ldots 0]$

  (eine 1 an Position i)

  für kleineres d gibt es nicht genügend unabhängige Vektoren

Hausaufgaben für KW 26

1. (die einfache Übungsaufgabe zur Wsk-Rechnung) Bei Korrekturlesen Ihrer Bachelorarbeit finden Anton 8 Fehler und Berta 9. Zwei Fehler wurden von beiden gefunden. Wieviele Fehler enthält die Arbeit? (unter der Annahme, daß Fehler zufällig verteilt sind und die Fehlersuchen unabhängig voneinander)
2. Wie behandeln Mikolov et al. *phrases* (Wortgruppen)?
3. Wie funktioniert das Lernen (Verschieben der Vektoren) in `https://github.com/dav/word2vec`?
4. (evtl. autotool) Aufgabe zur angenäherten Matrix-Faktorisierung
5. `https://gitlab.imn.htwk-leipzig.de/`

`waldmann/ki-ss18/ word-vec` benutzen und verbessern

Neuronale Netze

Zusammenfassung

- ein neuronales Netz N ist ein DAG, der eine Funktion $S_N : \mathbb{R}^{\text{Parameter}} \times \mathbb{R}^{\text{Eingaben}} \to \mathbb{R}^{\text{Ausgaben}}$ realisiert

  $x \mapsto S_N(P, x)$ soll durch gegebene Stützstellen verlaufen
- typische Anwendung: Klassifikation ($|\text{Ausgaben}| = 1$),

  Stützstellen sind $\subseteq \{(x, 1) \mid x \in C\} \cup \{(x, 0) \mid x \notin C\}$
- Struktur von N passend zu Aufgabenstellung
  - Konvolutions-Schicht — Translations-Invarianz
  - kleine Zwischenschichten erzwingen Kompression
- Bestimmen d. Parameterwerte durch Gradienten-Abstieg

Geschichte, Quellen

- Warren McCulloch, Walter Pitts: *A Logical Calculus of Ideas Immanent in Nervous Activity*, 1943, https://doi.org/10.1007/BF02478259
- Steven C. Kleene: *Representation of Events in Nerve Nets and Finite Automata*, in: Claude Shannon, John McCarthy: *Automata Studies*, 1956, https://doi.org/10.1515/9781400882618
- Yann Le Cun, Yoshua Bengio: *Word-level training of a handwritten word recognizer based on convolutional neural networks*, ICPR 1994, https://doi.org/10.1109/ICPR.1994.576881

Definition

- Syntax: ein neuronales Netz $N = (V_N, E_N)$
  ist ein gerichteter, geordneter, kreisfreier Graph,
  $V =$ Eingabe $\cup$ Parameter $\cup$ Neuronen, Ausgabe $\subseteq V$.
- Semantik:
  - für Neuron v mit p Parametern und e Eingaben:
    eine Funktion $S_v : \mathbb{R}^p \times \mathbb{R}^e \to \mathbb{R}$
  - für Netz: Funktion $S_N : \mathbb{R}^{\text{Parameter}} \times \mathbb{R}^{\text{Eingabe}} \to \mathbb{R}^{\text{Ausgabe}}$
- Bsp: $E = \{e_1, e_2\}, P = \{p_{i,j} \mid i, j \in \{1, 2, 3\}\}, A = \{c_3\}$.
  Neuron $c_i : (x, y) \mapsto f(p_{i,0} + p_{i,1}x + p_{i,2}y)$
  wobei $f : x \mapsto \max(0, \min(1, x))$
  Netz mit Neuronen $c_1(e_1, e_2), c_2(e_1, e_2), c_3(c_1, c_2)$.
- Ü: Param p, so daß $S_N(p, [x, y]) = \text{XOR}(x, y)$ auf $\{0, 1\}$

Spezielle Formen neuronaler Netze

- spezielle Funktionen:
  - Linearkombinationen: $f : \vec{x} \mapsto p_0 + \sum p_i \vec{x}_i$
  - Normalisierung auf geschlossenes Intervall $[0, 1]$
    - $x \mapsto \max(0, \min(1, x))$ stetig, monoton
    - $x \mapsto 1/(1 + \exp(-x))$ stetig, streng monoton, diff-bar
- spezielle Graphen:
  - flache Netze: Eingabe-Schicht, Ausgabe-Schicht
  - nicht flache („deep“) Netze: weitere („versteckte“) S.
- mehrfache Benutzung von Parametern: Konvolutionen
- diese (und andere) Spezialformen motiviert durch:
  - Nachbildung natürlicher Neuronen
  - Anwendg.sbezug (2D-Muster → Translationsinvarianz)

Modellierung von NN

- ```
  data Function arg
    = Dot_Product [arg] [arg] | Normalize arg
  data Ref e = Intern Index | Extern e
  data Net e = Net (M.Map Index (Function (Ref e)
  data Source = Param Int Int | Input Int
  n0 :: Net Source
  n0 = Net $ M.fromList
    [ (1, Dot_Product [ Extern (Param 1 0), ..
  ```
- ```
  evaluate
    :: Num z => (e -> z) -> Net e -> M.Map Index z
  ```
- blindes „Lernen“: Parameter-Vektor würfeln und bewerten
- `https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss18/tree/master/tiefl`

Konvolutions-Netze (CNN)

- Konvolution (Faltung) von Funktionen f, g

  ist conv$(f, g) : z \mapsto \sum\{f(x) \cdot g(y) \mid z = x + y\}$.
- hier nur für Fkt $\mathbb{N}^d \to \mathbb{R}$ mit endl. Def.-Bereich
- Bsp (1-dim): $f : 0 \ldots 5 \mapsto 3, 6 \ldots 10 \mapsto 7$, sonst 0,
  $g : 0 \mapsto 1, 1 \mapsto -1$, sonst 0
  conv$(f, g)(4) = f(4)g(0) + f(3)g(1) = 3 \cdot 1 + 3 \cdot -1 = 0$,
- Ü: conv ist assoziativ? neutrales Element? Inverse?
- CNN : $f =$ eine (Eingabe-)Schicht, $g =$ Parameter, meist $|\operatorname{dom} g| \ll |\operatorname{dom} f|$, conv$(f, g) =$ die nächste Schicht.

  erkennt alle Translationen des durch g repräsentierten Musters

Parameterbestimmung für NN

- zu lösen ist diese (Optimierungs-)Aufgabe:
  - gegeben: Netz N, Stützstellen $T \subseteq E \times A$
  - gesucht: Parameter p, so daß
    - (exakte Aufgabe) $\forall (e, a) \in T : a = S_N(p, e)$
    - (Optimierung) $\sum_{(e,a)\in T}(a - S_N(p, e))^2$ minimal
- iteratives Verfahren: p_0 zufällig, dann für $i = 0, 1, \ldots$
  - (e, a) zufällig aus T,
  - Gradient $G = \nabla(p \mapsto (a - S_N(p, e))^2)$, ist Fkt: $\mathbb{R}^P \to \mathbb{R}$
  - $p_{i+1} = p_i - c \cdot G(p_i)$, verschiebe in Richtung des stärksten Abstiegs
- Satz: wenn . . . , dann exist. $\lim p_i$ und ist *lokales* Min.
- Anwendung: wir wollen *globales* Min. unter realen Bed.

Verfahren zur Gradientenbestimmung

- löst diese Aufgabe:
  - gegeben: Vektor $x \in \mathbb{R}^d$, arithmetische Fkt. $f : \mathbb{R}^d \to \mathbb{R}$ als DAG (z.B. NN, Datenflußgraph eines Programms)
  - gesucht: $(\nabla f)(x)$, Wert des Gradienten an Stelle x
- Lösungsverfahren:
  - symbolische Differentiation: konstruiert DAG für ∇f
    ... der kann aber sehr groß werden
  - numerische Diff.: berechnet $(f(x + h \cdot \vec{e_i}) - f(x))/h$
    ... anfällig für Rundungs- und Auslöschungs-Fehler
  - automatische Diff.: symbolische Rechng. auf DAG von f
- Baydin, Pearlmutter, Radul, Siskind: *Automatic differentiation in machine learning: a survey*, 2015–2018
  `https://arxiv.org/abs/1502.05767`

Automatische Differentiation (AD)

- für DAG, die arithm. Fkt. repräsentieren:
- für jeden Knoten nicht nur Funktionswert,
  sondern auch Wert des Gradienten bestimmen
- repräsentiert durch „Zahlen“ der Form

```
data N v = N { absolute :: Double
             , linear :: M.Map v Double }
```

- dafür die arithm. Op. implementieren

```
instance Ord v => Num (N v) where
  x + y = _  ; x * y = _ ; exp x = _
```

- `evaluate :: Num w => (e -> w) -> Net e -> ...`
  kann für `w = N v` unverändert benutzt werden!

AD vorwärts und rückwärts

- allgemeiner Fall: mehrere Ausgaben. Netz realisiert Fkt. $f : \mathbb{R}^P \to \mathbb{R}^A$ (bei fixiertem Eingabevektor)
- Gradient ∇f ist lineare Fkt (Matrix) $\mathbb{R}^{P \times A}$.
- Kettenregel $=$ Matrixmultiplikation
- Matrixmult. ist assoziativ: kann man als foldl oder als foldr realisieren
- eben beschriebene Berechnung des Gradienten beginnend bei Eingabeknoten (foldl) ist *Vorwärts-Modus* der AD
- Alternative (foldr) ist *Rückwärts-Modus*
- abhängig von Dimensionen P, A kann das effizienter sein

Diskussion (Beispiel)

- für unser Beispiel (XOR) ist NN sinnlos, da man
  - XOR(x, y) direkt exakt und schnell ausrechnen kann
  - . . . es mehr Parameter (9) als Stützstellen (4) gibt
- Aufwand für Parameter-Optimierung lohnt sich nur, wenn $|\text{Testfälle}| \ll |\text{Anwendungsfälle}|$
- aber dann ist fraglich, ob das Netz die Eingaben $e \in$ Anwendungsfälle $\setminus$ Testfälle richtig behandelt.

Diskussion (allgemein)

Gary Marcus: *Deep Learning: A Critical Appraisal*, 2018,
`https://arxiv.org/abs/1801.00631`

The real problem lies in misunderstanding what deep learning is, and is not, good for. The technique excels

- at solving *closed-end classification problems*,
- in which a *wide range of potential signals*
- must be mapped onto a *limited number of categories*,
- given that there is *enough data available*
- and the test set *closely resembles the training set*.

Deviations from these assumptions can cause problems. Deep learning is just a statistical technique. All statistical techniques suffer from deviation from their assumptions.

Hausaufgaben für KW 27 (optional)

1. zu `https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss18/tree/master/tiefl`
   - experimentieren Sie mit verschiedenen Lernraten in `learn_gradient`
   - approximieren Sie durch das gegebene `n0` die Funktion $f : I^2 \to I : (x, y) \mapsto$ wenn $x^2 + y^2 \leqslant 1$, dann 1, sonst 0;
     a) für $I = [0, 1]$ b) für $I = [-1, 1]$ (Intervalle reeler Zahlen)

Monte-Carlo-Baumsuche

Inhalt, Motivation

- Ideen hinter aktuellen Go-Programmen (AlphaGoZero)
  - upper confidence bound (UCB) (ca. 1985)
  - Monte-Carlo-Baumsuche (MCTS) (ab 1993)
  - reinforcement (selbstverstärkendes) learning für CNN zur Verstärkung der MCTS (ab 2016)
- stärker als die besten professionellen Go-Spieler (4:1 gegen Lee Sedol 2016, 3:0 gegen Ke Jie 2017)
- das wurde lange für unmöglich gehalten
- *ohne* (Zero) Go-Expertenwissen beim Programmieren
- das ist einer der wenigen meßbaren Erfolge der KI
- bisher nicht reproduzierbar wg. hohen Rechenaufwands

UCB: Exploration und Exploitation

- gegeben: Spielautomaten $G_1, \ldots, G_n$ mit unbekannten Erwartungswerten (für Auszahlung minus Einzahlung) (mglw. einige davon > 0)

  gesucht: Strategie für maximalen Gewinn
- Lösung: betätige immer den Automaten, für den oberes Ende des Vertrauensbereiches (upper confidence bound)

$$U_i = \frac{\text{Gewinn}_i \text{ bisher}}{\text{Anzahl}_i \text{ bisher}} + \frac{1}{\sqrt{\text{Anzahl}_i \text{ bisher}}}$$

  maximal ist
- T. Lai, Herbert Robbins: *Asymptotically efficient adaptive allocation rules*, 1985, `https://doi.org/10.1016/0196-8858(85)90002-8`

Monte-Carlo-Baumsuche (MCTS) für Go

- Idee: ersetze *vollständige* (min-max, alpha-beta) Baumsuche durch *stochastische* Baumsuche

  Bernd Brügmann: *Monte Carlo Go*, 1993, `http://www.ideanest.com/vegos/MonteCarloGo.pdf`
- möglichst viele lineare (nicht verzweigende) *playouts*:

  in der Wurzel beginnend, in jedem Knoten Nachfolger *nach UCB-Kriterium* auswählen

  Remi Coulom: *Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search*, CG 2006, `https://www.remi-coulom.fr/CG2006/`
- Vorteile: leicht zu implememententieren, flexibel bzgl. Zeit

  Nachteil: Initialisierung neuer Knoten ist unklar/teuer

MCTS Beispiel-Implementierung

- `https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss18/tree/master/alpha-0`

  wie besprochen, aber nur für Gomoku (statt Go)
- playout:
  - innerer Knoten: select (nach UCB)
  - äußerer K.: expand and evaluate (heuristisch/exakt)
  - backup
- Zugvorschlag in der Wurzel: nach Häufigkeit
- DAG statt Baum: jeden Knoten in Transpositionstafel (Hashtabelle) nach Zobrist 1969 `https://www.cs.wisc.edu/techreports/1970/TR88.pdf`

  Hashwert bleibt bei Zugumstellungen (Transp.) gleich!

Alpha(Go)Zero: MCTS und CNN

- David Silver et al., *Mastering the Game of Go without Human Knowledge*, Nature 2017, `https://deepmind.com/blog/alphago-zero-learning-scratch/`
- MCTS *mit CNN zur Initialisierung neuer Knoten*:
  - Eingänge: Belegung des Spielbretts ($+$ 4 Züge zurück)
  - Ausgänge: $(v, \vec{p})$
    - v: Schätzung des Spielwertes
    - $\vec{p}$: für jeden möglichen Zug eine Wahrscheinlichkeit
- aktuelles Netz wird trainiert mit (einigen) während der Playouts (mit bisher bestem Netz) bestimmten Werten
- Turnierspiele zw. aktuellem und bisher bestem Netz, Austausch bei Gewinnrate $> 55\%$

Einzelheiten zu AlphaGoZero

- extern implementiertes Spielwissen (nur) für: Erzeugen und Ausführen gültiger Züge (z.B. Fangen von Steinen), Bewerten von Endknoten (Resultat, Punktevergleich).
- $\Rightarrow$ leicht übertragbar auf andere Spiele (Schach, Shogi)
- NN minimiert die Fkt $(z - v)^2 - \pi^T \log \vec{p} + c|\theta|^2$
  wobei (z, π): Spielwert und Häufigkeiten in MCTS
- NN mit insg. 40 oder 80 Schichten (Konvolutionen, lineare Normalisierung, nichtlineare Norm. auf $[0, 1]$)
- lineare Normalisierung: Ioffe und Szegedy `https://arxiv.org/abs/1502.03167`, 2015
- ausgeführt auf Google TPUs (die Werbung dafür ist die zugrundeliegende ökonomische Motivation)

Aktuelle Entwicklungen im Computer-Go

- open-Source-Nachbauten von AlphaGo und Versuch, CNN auf verteilten Privat-Rechnern zu lernen, z.B.

  Gian-Carlo Pascutto:

  `https://github.com/gcp/leela-zero`

  Interview (2018) `https://www.eurogofed.org/?id=205`
- Veröffentlichung von Go-Programmen großer Konzerne
  - (Mai 2018) `https://github.com/Tencent/PhoenixGo`
  - (Mai 2018) `https://research.fb.com/facebook-open-sources-elf-opengo/`
- Tencent World AI Weiqi Competition, Juni 2018

  `http://computer-go.org/pipermail/computer-go/2018-June/010897.html` (FineArt 7, LeelaZero 6, ELF 4)

Zusammenfassung, Ausblick

Themen

- Suche:
  - blinde, heuristische; in Baum, in Graph
  - FD-Constraints, SAT, Konsistenz, Propagation
  - Spiele: min/max, alpha/beta, CGT
- Logik, Regelsysteme
  - Entscheidungsdiagramme (BDD)
  - Hornklauseln, aussagenlogische Resolution
  - prädikatenl. Resolution, Unifikation, Prolog
- Statistische Modelle und Verfahren
  - Bayes-Netze, Textanalyse, Neuronale N., MCTS/UCB

Themen für Projekte und Abschlußarbeiten

- Verbesserung der in Ü benutzten autotool-Aufgaben (vor allem: bessere Aufgaben-Generatoren)
- Verbesserung der vorgeführten Software, insbesondere
  - Wortvektoren, neuronale Netze, Alpha Go(moku)
- dabei Kriterien:
  1. lesbares Haskell (Struktur und Bezeichnungen im Programm möglichst nahe beim jeweiligen Paper)
  2. leicht bau- und ausführbar (`stack test`)
  3. so effizient wie möglich (ohne Bedingung 1 zu verletzen, ggf. optimierte Implementierg. auslagern)
- die o.g. Themen sind speziell; Betreuung allgemeiner KI-Themen durch Prof. Schwarz (nach Anfrage)