Лекции по изкуствен интелект

Здравко Марков

January 11, 1999

Contents

Chapter 1

Увод

1.1 Лекции

1.1.1 Предмет на изкуствения интелект (ИИ)

Създаване на *интелигентни* (мислещи) системи (машини, роботи, същества) с цел:

- Моделиране на естествения интелект (човека) и изучаването му. За разлика от философията и психологията основната цел на ИИ включва и създаване на интелект (освен изучаването му).

- Автоматизиране на функции и решаване на задачи, които изискват интелигентност.

Въпроси: Решаването на кои задачи изисква интелигентност? Какво е интелигентност? Какво е интелект?

1.1.2 Дефиниции

- Да накараме компютрите да "мислят", създаване на "машини с мозъци" (Haugeland, 1985).

- Изучаване на психическите функции на човек чрез изчислителни модели (Charniak & McDermott, 1985).

- Изследване на изчислителни модели, които могат да възприемат, разсъждават и действуват рационално (разумно) (Winston, 1992).

- Изкуството да се създават машини, които изпълняват функции, изискващи интелигентност, когато се изпълняват от човек (Kurzweil, 1990).

- Как да накараме компютрите да вършат неща, които засега ние (хората) вършим по-добре (Rich & Knight, 1991).

- Област на информатиката, която се занимава с моделиране на интелигентно поведение (Luger & Stubblefield, 1993).

1.1.3 Подходи (методи)

1. Моделиране на човешки разсъждения – когнитивно моделиране (когнитивна наука). GPS (Newell and Simon, 1972).

2. Моделиране на действията (поведението) на човек. Тест на Тюринг (1950), Loebner Prize (http://www.loebner.net/Prizef/loebner-prize.html). Необходими когнитивни способности: *обработка (разбиране) на естествен език, представяне на знания, разсъждения и самообучение.*

3. Моделиране на рационално (разумно) мислене. Логически подход - закони на мисленето (силогизмите на Аристотел - "Сократ е човек. Всички хора са смъртни. Следователно Сократ е смъртен"). Недостатъци - необходимост от пълно и 100% вярно знание.

4. Моделиране на рационално (разумно) действие (поведение). Подходът на *рационалните (разумни) агенти.* Включва необходимост от когнитивни способности (2) и логическия подход, но е по-общ от тях.

1.1.4 Свързани области (основи на ИИ)

- Философия. "Историята на ИИ започва от Сократ (-450) ..." (H. Dreyfus, 1979) - алгоритъм. Аристотел - анализ на целите и средтствата - GPS (Newell and Simon, 1972).
- Математика. Логика, алгоритми, изчислимост, резолюция.
- Психология. Експериментална психология, когнитивна психология, когнитивно моделиране.
- Информатика.
- Лингвистика. Синтактични структури (Chomsky, 1957).
- Наука за мозъка. Неврони и невронни мрежи.

1.1.5 История

- Предистория – раждането на информатиката. Развенчаване на мита за сметалата, т.е. компютрите могат да манипулират символи и моделират логически разсъждения. Дж. Бул (законите на мисленето), Ч. Бабидж (аналитична машина), А. Тюринг (тест на Тюринг), Дж. Фон Нойман (самовъзпроизвеждащи се клетъчни автомати).
- Раждане (1943-1956). Preceptron (Rosenblatt, 1957), LT (Newell, Simon,1956), Конференцията в Dartmouth (McCarthy, 1956).
- Начален ентусиазъм (1952-1969). GPS, Игра на шашки (Samuel, 1952), ANALOGY, SIR, STUDENT, ARCH.
- Разочарования и реализъм (1966-1974). ELIZA (Weizenbaum, 1965). Машинен превод. "The spirit is willing but the flesh is weak - водката е хубава, но месото е развалено". Perceptrons (Minsky and Papert, 1969). Борба с комбинаторната експлозия.
- Представяне на знания и експертни системи. DENDRAL (Buchanan et al, 1969). MYCIN (Feigenbaum et al, 1970). PROSPECTOR (Duda et al,1979). SHRDLU (Winograd).
- ИИ - индустрия (1980-1988). R1 (DEC), Пето поколение, Пролог, Ядра на експертни системи. 2 милярда USD, 1988.
- Завръщането на невронните мрежи. PDP (Rumelhart and McClelland, 1986).
- Настояще.
  - Програмата Deep Blue на IBM бие Каспаров на шах, 1997 г. (http://www.chess.ibm.com/home/html/b.html).
  - Разпознаване на реч в реално време.
  - Автоматично доказателство на теореми (1996, системата EQP решава 60-годишен нерешен проблем, http://www-unix.mcs.anl.gov/~mccune/papers/robbins/).
  - Използуване на експертни системи в практиката (http://auai.org/BN-Routine.html).
  - Научни открития в химията и биологията чрез методите на машинното самообучение (http://www.cs.jhu.edu/~salzberg/glimmer-nar.pdf).
  - Обработка на изображения и роботика ("No hands across America" – управление на автомобил от Питсбърг до Сан Диего, 1995, http://www.cs.cmu.edu/~pomerlea/nhaa.html)

1.1.6 Интелигентни агенти

- Агент = Архитектура + Програма.
- Видове агенти: рефлексен, агент с памет (вътрешно състоятие), целеви агент, агент максимизиращ ползата от своите действия.
- Среда, в която действуват агентите.

1.1.7 Книги

- Bratko, I. *Prolog Programming for Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts, 1986.
- Bratko, I. *Prolog Programming for Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts, second edition, 1990.
- Haugeland, J. (editor). *Artificial Intelligence: The Very Idea*, MIT Press, Cambridge, Massachusetts, 1985.
- Charniak, E. and D. McDermott. *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts, 1985. Български превод: Д. Дочев, Х. Дичев, З. Марков, Л. Синапова, TEMPUS JEP 1497 & СОФТЕХ, София, 1997.
- Winston, P.H. *Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts, third edition, 1992.
- Kurzweil, R. *The Age of Intelligent Machines*, MIT Press, Cambridge, Massachusetts, 1990.
- Rich, E. and K. Knight. *Artificial Intelligence*, McGraw-Hill, New York, second edition, 1991.
- Luger, G.F and W. A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Benjamin/Cummings, Redwood City, California, second edition, 1993.
- Russell, S and P. Norvig. *Artificial Intelligence: A Modern Approach*, Prentice Hall, Upper Saddle River, New Jersey, 1995.
- Дочев Д., Хр. Дичев, З. Марков, Г. Агре. *Програмиране на Пролог: Основи и приложения*, Наука и изкуство, София, 1989.
- Марков, З. *Индуктивни методи за машинно самообучение*, TEMPUS JEP 1497 & СОФТЕХ, София, 1996.

1.1.8 Ресурси достъпни чрез Интернет

- World-Wide Web Virtual Library: Artificial Intelligence (`http://www.cs.reading.ac.uk/people/dwc/ai.html`)
- CMU Artificial Intelligence Repository (`http://www.cs.cmu.edu/Groups/AI/html/repository.html`)
- NRC AI Page (`http://ai.iit.nrc.ca/ai_top.html`)
- ASM SIGART (`http://sigart.acm.org/`)
- AI, Cognitive Science, and Robotics (`http://www-mice.cs.ucl.ac.uk/misc/ai/`)

1.2 Семинарни упражнения

- Решаването на кои задачи изисква интелигентност? Какво е интелигентност? Какво е интелект? Притежават ли животните интелект ? (дискусия)
- Тест на Тюринг и Loebner Prize (виж дискусията с Марвин Мински на http://www.loebner.net/Prizef/minsky.html и статията на Stuart Shieber)

1.3 Лабораторни упражнения

Работа със системата Sicstus Prolog.

Chapter 2

Пространство на състоянията и търсене

2.1 Лекции

2.1.1 Агенти решаващи задачи

За да удовлетворят целите си агентите трябва да изпълнят последователност от действия, които последователно променят средата до достигане на целевото и състояние. За това е необходимо формално представяне на състоянията на света (агент + среда). Начини за реализация:

- Таблица от двойки "състояние-действие", пълно описващи всички възможности и съответното правилно действие.

- Използване на знания за моделиране на състоянията на света и за начина, по който действията биха ги променили (преходите от едно в друго състойние) – *симулиране на изпълнението на действия с цел намиране на последователност, която би довела до целевото състояние.* Това обикновено се прави чрез *търсене в пространството на състоянията.*

2.1.2 Фармализация на задачата

- Състояния (допустими, начално)

- Оператори

- Целево състояние или тест за проверка на целевото състояние

2.1.3 Задача

- Дадено: начално и целево състояние

- Търси се: последователност от оператори (път), които водят от началното до целевото състояние.

2.1.4 Примери

- Търсене на път между два града по карта.

- Задачата за фермера, вълка, зелето и козата.

- Играта с 8 плъзгащи се плочки.

- Нареждане на 8 царици на шахматната дъска.

- Криптоаритметика (SEND+MORE=MONEY).

- Реални задачи: търговски пътник, проектиране на чипове, управление на роботи.

2.1.5 Решаване на задачата

- Търсене
  - Разширение на състояние (възел).
  - Функция за избор на наследник.
  - Дърво на търсене.
  - Алгоритми: слепи (пълно изчерпване, uninfromed, blind, exhaustive) и евристични (informed).
  - Оценка на алгоритмите: цена на пътя и цена (сложност) на търсенето.
- Удовлетворяване на ограничения (constraint satisfaction).
  - Множество от променливи (с области на стойностите).
  - Множество от ограничения (допустими/недопустими комбинации от стойнсти на променливите – двойки, тройки и т.н.).
  - Целево състояние (състояния) – множество от стойностите на всички променливи, които не нарушават ограниченията.
  - Пример: задачата за 8-те царици.
  - Алгоритми: търсене с възврат (backtracking), разпространение на ограниченията (forward checking).

2.1.6 Алгоритми за пълно изчерпване

- Общ алгоритъм – реализация чрез използуване на опашка.
- Търсене в дълбочина (depth-first) – добавяне на нов възел в началото на опашката. Вариант: итеративно търсене по нива (iterative deepening).
- Търсене в ширина (breadth-first) – добавяне на нов възел в края на опашката. Вариант: търсене с равномерна цена на пътя (uniform cost search) – сортиране на опашката по цената на изминатия път.
- Избягване на повторения и зацикляне.
- Пълнота, оптималност и сложност (по време = брой изследвани възли и по памет = максимална дължина на опашката) на алгоритмите.

2.2 Семинарни упражнения

- Решаване на задачата за 8-те царици чрез търсене в пространство на състоянията и чрез удовлетворяване на ограничения (със и без forward checking). Сравнение на сложността на двете решения.
- Оценка на сложността на алгоритмите за търсене в трафи.
- Оценка на сложността на решаването на задачи с ограничения.

2.3 Лабораторни упражнения

- Програмиране на задачата за фермера, вълка, зелето и козата.
- Програмиране на базовия алгоритъм за търсене в графи (с използуване на опашка) и всички негови варианти (използуване на различни функции за избор на възел и добавянето му отпред или отзад на опашката). Необходима ли е опашка за реализацията на търсене в дълбочина?
- Решаване на задачата за 8-те царици чрез удовлетворяване на ограничения.
- Сортиране на списък чрез търсене в пространство на състоянията. Сравнение с "метода на мехурчето".

Chapter 3

Евристично търсене

3.1 Лекции

3.1.1 Защо са необходими евристики?

- Повечето задачи за търсене са експоненциални, следователно нерешими при големи размерности.
- Евристични функции – дават оценка (приближение) на цената на пътя до целта. Пример: разстоянието по права линия до целта при търсене на път по карта.
- При използуване на евристични функции може да се намери ефективно неоптимално решение на произволна задача или оптимално решение на среда по трудност задача.

3.1.2 Алгоритми за евристично търсене в графи

- Търсене на най-добър път (best-first search) – сортиране на опашката в съответствие с евристичната функция $h(n) = \{$оценка на цената на пътя от n до целта$\}$. Неоптимален алгоритъм.
- Търсене с ограничена ширина (beam search) – ограничаване на опашката до първите n най-добри възела (в съответствие с евристиката). Варинат ($n = 1$): метод на най-бързото спускане (hill climbing search). Непълен и неоптимален алгоритъм.
- Минимизиране на общата цена на пътя: A^*. Комбинира търсене с равномерна цена на пътя (пълно, оптимално, неефективно) с търсене на най-добър път (неоптимално, ефективно). Опашката се сортира в съответствие с функцията $f(n) = g(n) + h(n)$. ($g(n)$ – цена на пътя от началото до n.)

3.1.3 Свойства на A^*

- Пълнота: краен брой разклонения от всеки възел и положителна цена на преходите от едно състояние в друго.
- Приемлива (admissible) евристична функция: $h(n)$ не надминава цената на най-добрия път от n до целта.
- Оптималност при използуване на приемлива евристика.
- Сложност: $O(b^d)$ в най-лошия случай.

3.1.4 Евристични функции

- Пример: играта с 8 плъзгащи се плочки (среден брой разклонения от възел = 3, средна дължина на решенията = 20, пространство на състоянията $9! = 362880$, търсене с пълно изчерпване – 3^{20}). Евристики: разстояния до целевото състояние (Хеминг, Манхатън).
- Свойства и сравнение на евристики (доминираща евристика).
- Избор на евристики: абстракция (отслабване на ограниченията) и тегловна сума от свойства на състоянието (самообучение).

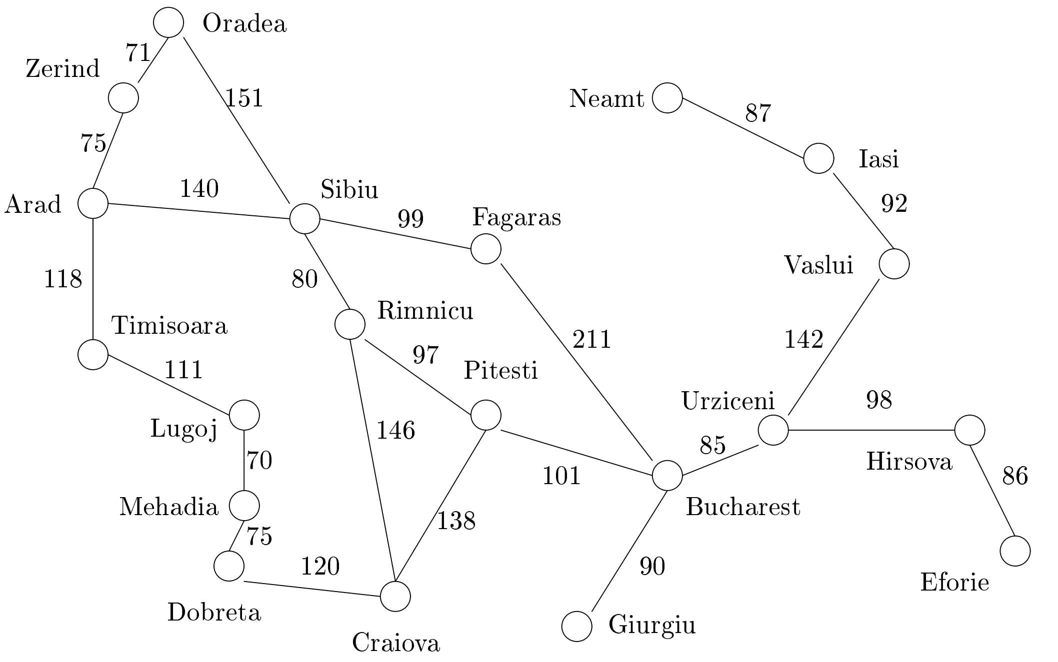


Figure 3.1: Пътна карта на Румъния

3.1.5 Евристики при удовлетворяване на ограничения

- Избор на променлива с най-малко възможни стойности (възможните стойности се ограничават чрез разпространение на ограниченията) – намалява се броят на разклоненията на текущата стъпка. Пример: оцветяване на карта (цветът на E е син – единствената възможна стойност).

- Избор на променлива, която участва в най-много ограничения върху други неприсвоени променливи – намалява броят на разклоненията на бъдещите стъпки.

- След като променливата за присвояване се избере, за нея се избира стойност, която минимизира ограниченията върху останалите промелниви (ако се избере C, то за негов цвят трябва да се избере червен, което минимизира ограниченията върху E и F).

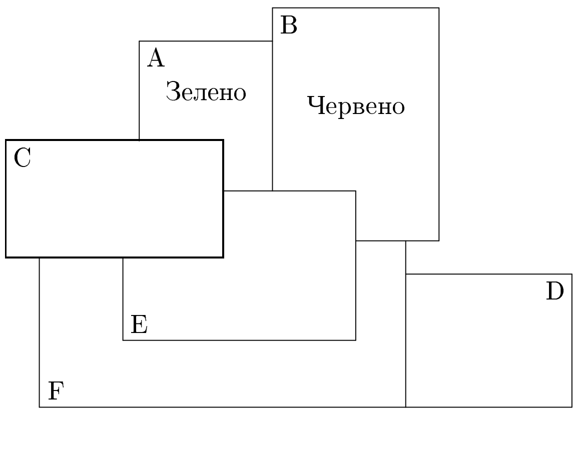


Figure 3.2: Задача за оцветяване на карта с три цвята (червен, зелен и син)

3.1.6 Евристични подобрения на алгоритмите

- Итеративно подобряване на началното състояние (когато целевото състояние съдържа решението, например 8-те царици, задачата за фермера, вълка, зелето и козата).
- Избор на случайно начално състояние и многократно прилагане на метода на най-бързото спускане.
- Симулирано "закаляване" (simulated annealing). Използуване на параметър T (условно "температура"), който внася случайност при работата на алгоритъма (например избора на следващ възел при метода на най-бързото спускане). T се променя по някакъв закон (метала "изтива") по време на работа на алгоритъма. Цел: избягване на попадане в локален екстремум.

3.2 Семинарни упражнения

- Реализация на алгоритмите за търсене на най-добър път, търсене с ограничена ширина и A^*.
- Доказване на оптималността на A^*.
- Изследване сложността на алгоритмите.
- Намиране на евристики за: сортиране на списък, играта с 8 плъзгащи се плочки и нареждането на 8 царици на шахматната дъска.
- Избор на алгоритъм и евристики за решаване на задачата за оцветяване на карта (фигура 3.2).

3.3 Лабораторни упражнения

- Програмиране на алгоритмите за търсене на най-добър път, търсене с ограничена ширина и A^*.
- Експериментално сравняване на работата на алгоритмите (време и брой на изследвани състояния) върху задачите: търсене на път от Arad до Bucharest (фигура 3.1), сортиране на списък, играта с 8 плъзгащи се плочки.
- Решаване на задачата за оцветяване на карта (фигура 3.2).

Chapter 4

Моделиране на игри

How Intelligent is Deep Blue?
Drew McDermott
`http://www.cs.yale.edu/HTML/YALE/CS/homepage/op-ed/deep_blue.html`

4.1 Лекции

4.1.1 Връзка с ИИ

- Добре дефинирани задачи, изискващи интелигентност.
- Трудни задачи (1): предполагат неопределеност (не може предварително да се определи ходът на противика).
- Трудни задачи (2): пространството на състоянията в повечето случаи е огромно (шах: разклонения = 35, дълбочина = 50, възли = 35^{100}) .
- Въпреки това хората играят добре без да се притесняват от огромното пространство на търсене.
- Подходяща област за изследване на методите за търсене.

4.1.2 Формализация на задачата

- Частен случай на задачата за търсене в пространството на състоянията.
- Състоянията, в които играта свършва се наричат *терминални*.
- Функция на полезност (utility) – присвоява стойност на терминалните състояния (например: победа = 1, загуба = -1, равен = 0).
- Дървото на търсене е разделено на нива, на всяко от които са ходовета на един играч. Играчите условно могат да се нарекат MIN (този, който се стреми на минизира фунцията на полезност) и MAX (стараещ се да максимизира функцията на полезност).

4.1.3 Минимаксен алгоритъм

- Генериране на пълното дърво на играта до терминалните възли (виж фигура 4.1).
- Изчисляване на функцията на полезност за всеки възел отдолу нагоре, започвайки от терминалите (виж фигура 4.2).
- За всеки възел MAX се избира максималната стойност на неговите наследници (т.е. ходът, който максимизира полезността).
- За всеки възел MIN се избира минималната стойност на неговите наследници (т.е. ходът, който минимизира полезността).
- По този начин се определя оптималния ход в корена на дървото (изборът на текущ ход).

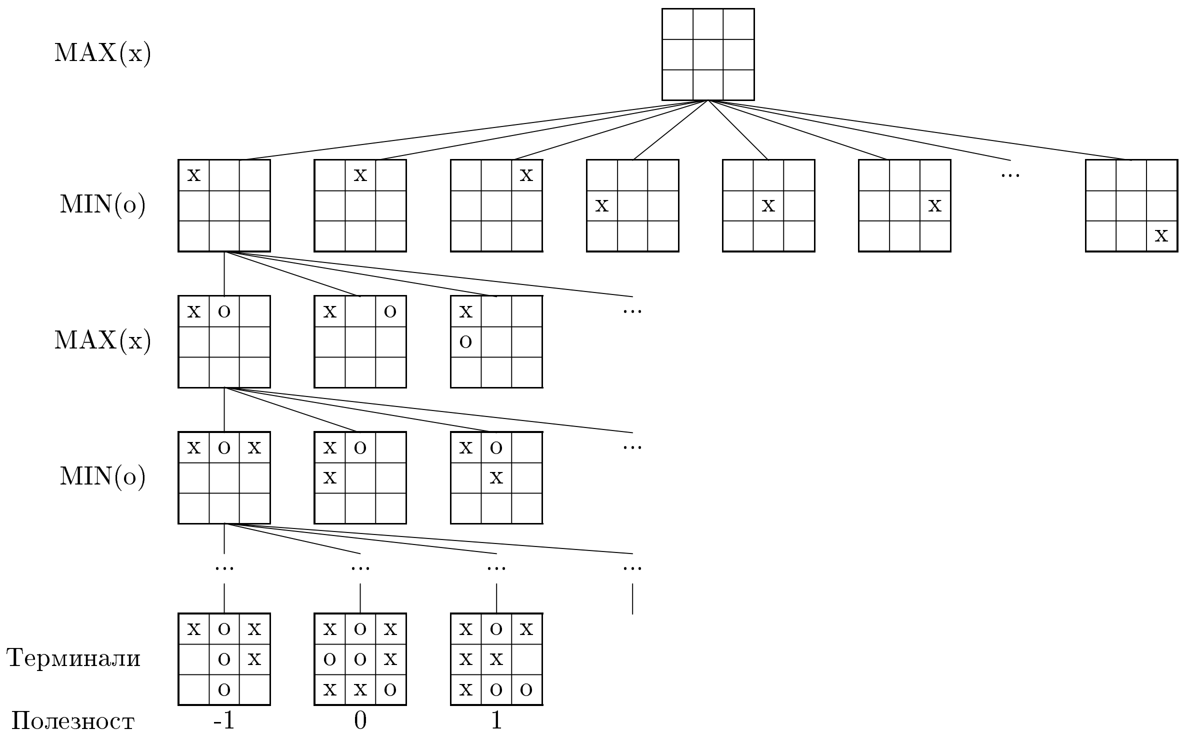


Figure 4.1: Дърво на играта "кръстчета и нули"

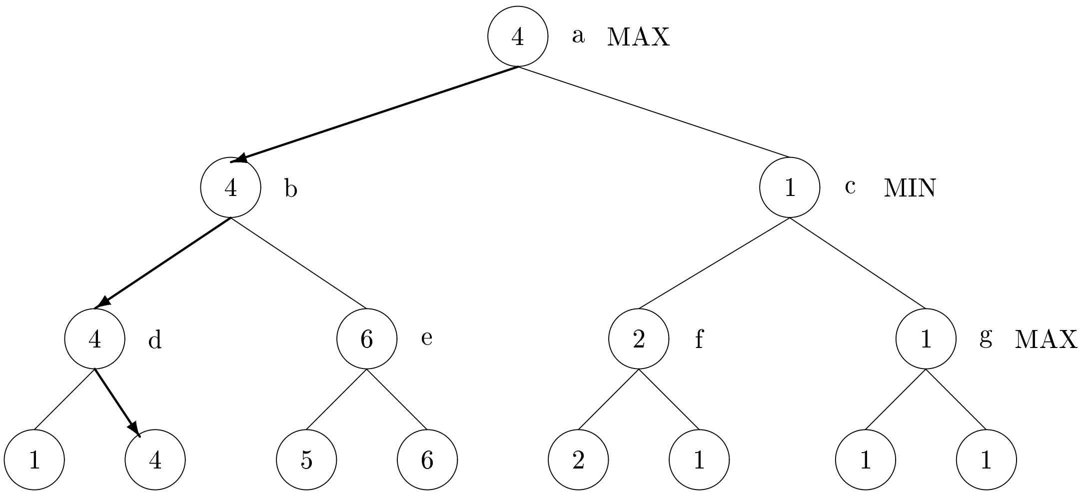


Figure 4.2: Прилагане на минимаксния алгоритъм към примерно игрово дърво (ходовете, избрани по този алгоритъм с означени със стрелки с плътна линия)

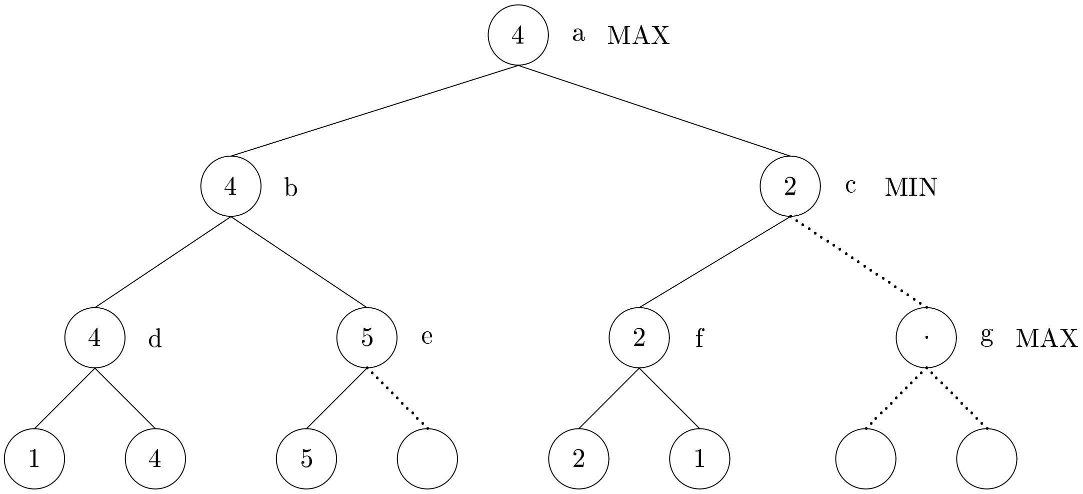


Figure 4.3: Прилагане на алфа-бета процедура към примерно игрово дърво

4.1.4 Усъвършенствуване на минимаксния алгоритъм

- Освен за съвсем прости игри (например 3X3 "кръстчета и нули") алгоритъмът е неприложим поради огромния брой възли в дървото.
- Отсичане на дървото от определен възел надолу и намиране на евристична оценка на полезността.
- Идеалната евристика би трябвало да отразява вероятността, че MAX ще победи (MIN ще загуби) при зададента позиция на играта.
- Пример за проста евристика (шах): тегловна сума от броя на притежаваните фигури (недостатък: не отчита позицията).
- Определяне на нивото на отсичане (предполага се, че евристиките за по-долните нива на дървото се намират по лесно и са по точни):
  - Предварително задаване на граница на дълбочината d.
  - Използуване на итеративно търсене по нива при предварително зададено време за изчисление.
  - Прилагане на евристична оценка само в "спокойните" позииции на играта (позиции, които не предполагат рязка промяна на оценката при по-нататъшната игра).
  - Проблемът с хоризонта: някои ситуации изискват продължаване на търсенето още по-дълбоко (не могат да се оценят, защото са "зад хоризонта"). Например превръщането на пешка в царица при шаха.

4.1.5 Алфа-Бета отсичане (ефективна реализация на минимаксния алгоритъм)

- Приближено изчисляване на стойностите на полезността (отбележете разликите между фигури 4.2 и 4.3), което обаче е достатъчно за определяне на верния ход.
- Търсене в дълбочина.
- Два текущи параметъра по време на търсене в игровото дърво:
  - α = максималната стойност за MAX, получена до момента.
  - β = минималната стойност за MIN, получена до момента.
- Ако MIN получи стойност по-голяма от β, то неговите съседи на същото ниво могат да не се оценяват (отсичане на дървото). Например в позиция левия наследник на е (фигура 4.3) MIN получава 5, което е по-голямо от $\beta = 4$, следователно десния наследник на е не се оценява.

- Ако MAX получи стойност по-малка от α, то неговите съседи на същото ниво могат да се пропуснат. Например в позиция f (фигура 4.3) MAX получава 2, което е по-малко от $\alpha = 4$, следователно позиция g може да се пропусне.

- Недостатък: зависи от реда на обхождане на наследниците на позициите в дървото. Ако най-перспективните ходове се разглеждат първи, то се получава намаляване на сложността по време от $O(b^d)$ до $O(b^{\frac{d}{2}})$.

4.1.6 Игри с елемент на случайност

- Например игри със зарове (табла).

- Освен MIN и MAX дървото на играта се разширява с още един вид нива – нива със случайни възли.

- Случайните нива съдържат толкова възела, колкото са възможните изходи при хвърлянето на заровете, с различна роля за играта (например за таблата има 21 различни изхода).

- Вариант на минимаксния алгоритъм, включващ случайност:

  - d_i $(i = 1, ..., n)$ – различните изходи от случайното ниво;
  - $P(d_i)$ – вероятност за случаен изход d_i;
  - $S(N, d_i)$ – множества ходове от позиция N за всеки един случаен изход d_i;
  - Ако в N MAX е на ход: $utility(N) = \sum_{i=1}^{n} p(d_i) \max_{s \in S(N,d_i)} utility(s)$
  - Ако в N MIN е на ход: $utility(N) = \sum_{i=1}^{n} p(d_i) \min_{s \in S(N,d_i)} utility(s)$

- Стойностите на функцията на полезност не са само измежду тази на терминалите. Следователно освен наредбата между терминалите (печели, губи, равен) важна е и абсолютната им стойност. Например различни решения ще се вземат при $+1$, 0 и -1, и при $+10$, 0 и -10.

- Нарастване на сложността по време: $O(b^d n^d)$ (n – случайни изхода). В реални игри рядко се използува голяма дълбочина.

- Аналогът на алфа-бета отсичането е по-сложен. Необходимо е на се намери граница на полезността, която обаче е средна стойност и зависи от броя на изследваните възли. Това може да стане като се поставят граници на стойностите на терминалите (напр. $+1$ и -1) и използувайки броя на случайните възли да се изчисли граница и за случайните възли.

4.2 Семинарни упражнения

- Представяне на игровото дърво в общия случай (позиции, ходове, оценки на терминалите).

- Реализация на минимаксния алгоритъм.

- Реализация на алфа-бета алгоритъма.

- Реализация на играта "кръстчета и нули" с размерност 3X3.

- Намиране на подходяща оценъчна функция за нетерминални позиции на играта "кръстчета и нули". Реализация на алгоритъм за итеративно търсене по нива (минимакс и алфа-бета).

4.3 Лабораторни упражнения

- Програмиране на минимаксния алгоритъм.

- Програмиране на алфа-бета алгоритъма.

- Експериментални сравнения (брой на изследвани възли) на минимакс и алфа-бета върху симулационно игрово дърво (фигура 4.2).

- Програмиране на играта "кръстчета и нули" в три варианта: минимакс, алфа-бета и итеративно търсене по нива (минимакс и алфа-бета) и експерименти (брой на изследвани ходове и бързодействие).

Chapter 5

Представяне на знания - логически подход

5.1 Лекции

5.1.1 Ителигентен агент използуващ знания

- База от знания – множество от изречения (на езика за представяне на знания) относно факти, описващи света.
- Език за представяне на знания.
- Механизъм за извод. Какво следва (може да се изведе) от базата знания и това, което е възприел агентът за света (средата) в даден момент. Кое действие е най-подходящо да се изпълни в момента.
- Основни знания – начални знания за света (агент + среда).
- Нива на описание на агента:
  - Ниво на знанията.
  - Логическо ниво. Знанието, записано на езика за представяне на знания.
  - Реализационно ниво. Реализацията на езика за представяне на знания.
- Създаване на агент, използуващ знания
  - Декларативен подход.
  - Подход на самообучение.

5.1.2 Симулационна среда за агенти използуващи знания – wumpus world

Идеята идва от стара компютърна игра, наречена "Wumpus". Мрежа от квадрати (пещери), оградени със стена. Съседните квадрати (хоризонтално и вертикално) са свързани с проходи, през които може да минава агентът. В един от квадратите има злато. В началото агентът се намира в долния ляв ъгъл и трябва да достигне до златото, да го вземе и да се върне в началото, където може да излезе навън (което е целта на играта). В един от квадратите има звяр, който изяжда агента, ако той влезе в неговия квадрат. В някои от квадратите има пропасти, в които агентът пропада ако влезе в тях. Пример на този свят е даден на фигура 5.1.

Агентът може да възприема следната информация от средата:

- Миризма, когато се намира в съседните (без диагоналните) на звяра пещери.
- Течение, когато се намира в съседните (без диагоналните) на пропастите пещери.
- Блясък, когато се намира в пещерата със златото.
- Светлина, когато се намира в началната пещера (долния ляв ъгъл).
- Когато се блъсне в стената усеща удар.
- Координатите на пещерата, към която се е насочил.

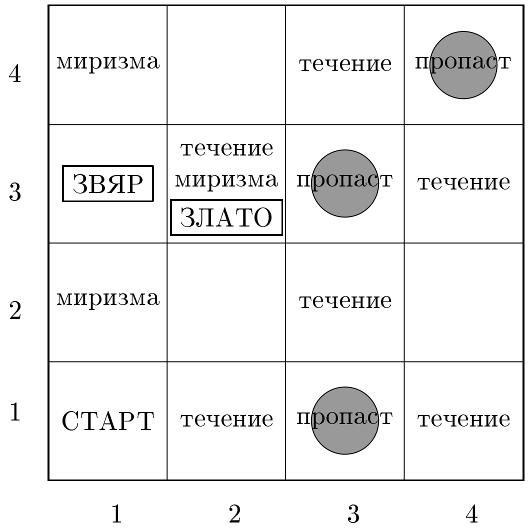


Figure 5.1: Симулационна среда за интелигентни агенти – wumpus world

Агентът може извършва следните действия:

- Движение един квадрат напред.
- Завъртане на 90° наляво или надясно.
- Вземане на златото (възможно само в пещерата със злато).
- Излизане навън (възможно само в началното положение)

5.1.3 Езикът на предиктното смятане от I-ви ред

Синтаксис

- *Азбука* (първите четири класа символи не са фиксирани и могат да се дефинират по различен начин):
  - *Променливи* (главни букви).
  - *Константи* (малки букви или числа).
  - *Функции* (f, g, h).
  - *Предикати* (p, q, r, `баща`, `майка`, `обича`).
  - *Логически съюзи*: $\wedge$ (*конюнкция*), $\vee$ (*дизюнкция*), $\neg$ (*отрицание*), $\leftarrow$ (*импликация*, записва се и като $\rightarrow$) и $\leftrightarrow$ (*еквивалентност*).
  - *Квантори*: $\forall$ (*квантор за общност*) и $\exists$ (*квантор за съществуване*).
  - *Символи за пунктуация*: (,) и ,.
- *Терм*:
  - Променливата е терм.
  - Константата е терм.
  - Ако f е n-местна функция ($n \geq 0$) и $t_1, t_2, ..., t_n$ са термове, то $f(t_1, t_2, ..., t_n)$ е терм.
- *Формула*:
  - Ако p е n-местен предикат ($n \geq 0$) и $t_1, t_2, ..., t_n$ са термове, то $p(t_1, t_2, ..., t_n)$ е формула (наречена *атомарна формула* или просто *атом*).
  - Ако F и G са формули, то $(\neg F)$, $(F \wedge G)$, $(F \vee G)$, $(F \leftarrow G)$ и $(F \leftrightarrow G)$ са формули.
  - Ако F е формула и X е променлива, то $(\forall X F)$ и $(\exists X F)$ са формули.

- Пример: $\forall X$ (мъж$(X) \rightarrow (\exists Y$ (жена$(Y) \wedge$ обича(X,Y)))). Неформално: "За всеки мъж съществува жена, която той обича". Тази формула може да се запише и по-просто: $\forall X\ \exists Y$ мъж$(X) \rightarrow$ жена$(Y) \wedge$ обича(X,Y).

- Формула, в която всички променливи са *квантифицирани* (срещат се непосредствено след квантор и след това само в областта на неговото действие), се нарича *затворена* (такава е горната формула).

- Термове и формули, които не съдържат променливи, се наричат *основни*.

- Трансформации, запазващи истинността на формулите (преобразуване в еквивалентна форма).

- *Клаузи:*

  – Атом или отрицание на атом се нарича *литерал*.

  – Двойката литерали A и $\neg A$ се наричат *контрарни*;.

  – Клаузата е универсално квантифицирана дизюнкция от литерали.

  – Клауза без литерали се нарича *празна клауза* ($\Box$) и означава логическата константа `лъжа`.

  – Съществува формална процедура за превод на формули в еквивалентна *клаузна форма*. Пример:

    1. $\forall X man(X) \rightarrow \exists Y woman(Y) \wedge loves(X,Y)$
    2. $\forall X \neg man(X) \vee \exists Y woman(Y) \wedge loves(X,Y)$ (премахване на импликациите)
    3. $\forall X \neg man(X) \vee (woman(s(X) \wedge loves(X, s(X)))$ (сколемизация)
    4. $\neg man(X) \vee (woman(s(X)) \wedge loves(X, s(X)))$ (премахване на кванторите за всеобщност)
    5. $(\neg man(X) \vee woman(s(X)) \wedge (\neg man(X) \vee loves(X, s(X)))$ (преход в нормална конюнктивна форма)

- *Клаузи на Хорн*: клауза с не повече от един положителен литерал (атом без отрицание).

- *Пролог*: Клаузите на Хорн от вида $\forall X_1 ... \forall X_n (A \vee \neg B_1 \vee \neg B_2 \vee ... \vee \neg B_m)$, където $A, B_1, ..., B_m$ $(m \geq 0)$ са атоми, а $X_1, ..., X_n$ са всички променливи в тях, се означават също и така (кванторите за общност се изпускат, тъй като се подразбират): $A \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_m$ или $A : -B_1, B_2, ..., B_m$. A се нарича *глава на клаузата* (ако липсва клаузата се нарича *цел*), а $B_1, B_2, ..., B_m$ – *тяло на клаузата* (ако липсва клаузата се нарича *факт*). Клаузите в Пролог са по-общи от клаузите на Хорн, тъй като в телата им се допуска употребата на още един вид отрицание – т. нар. *отрицание чрез неуспех*, което ще означаваме с `not` B_i, където B_i е атом от тялото на клаузата. *Програмата на Пролог* е конюнкция (множество) от клаузи от горния вид.

Субституции и унификация

Множество от вида $\theta = \{v_1/t_1, v_2/t_2, ..., v_n/t_n\}$, където v_i са различни променливи ($v_i \neq v_j$ за всяко $i \neq j$), а t_i – термове ($t_i \neq v_i$, $i = 1, ..., n$), се нарича *субституция*. Субституцията θ се прилага към формулата E, като за всяко i на мястото на всяко участие на променливата v_i в E се поставя t_i. Заместването се прави *едновременно* на всички променливи. Резултатът от прилагането на θ към E се означава с $E\theta$ и се нарича *екземпляр* на E. Субституция, която замества променливи с други променливи (т. е. всички t_i са променливи), се нарича *преименуване на променливите*, а резултатът от прилагането – вариант. Две формули F и G са варианти една на друга, ако съществуват субституции θ и σ, такива че $F\theta = G$ и $G\sigma = F$.

Нека $\theta = \{u_1/s_1, ..., u_m/s_m\}$ и $\sigma = \{v_1/t_1, ..., v_n/t_n\}$. *Композицията* на субституциите θ и σ, означена с $\theta\sigma$, се дефинира с множеството $\{u_1/s_1\sigma, ..., u_m/s_m\sigma, v_1/t_1, ..., v_n/t_n\}$, от което са премахнати всички $u_i/s_i\sigma$, за които $u_i = s_i\sigma$, и всички v_j/t_j, такива че $v_j \in \{u_1, ..., u_m\}$. Прилагането на композицията $\theta\sigma$ е еквивалентно на последователното прилагане на θ и σ, т. е. $E\theta\sigma = (E\theta)\sigma$.

Прилагането към два различни терма t_1 и t_2 на субституция θ, която ги прави еднакви, т. е. $t_1\theta = t_2\theta$, се нарича *унификация*, а θ – унификатор. Един унификатор μ на t_1 и t_2 се нарича *най-общ унификатор* (*mgu*), ако за всеки друг унификатор γ на t_1 и t_2 съществува субституция δ, такава че $\mu\delta = \gamma$.

Модел, логическо следствие

Нека S е конюнкция от клаузи. Множеството от всички основни атоми с предикати от S и аргументи, получени чрез комбинации на константи и функции, участващи в S, се нарича *ербранова база* на S и се означава с B_S. Нека M е подмножество на B_S, а C – клауза от S. Казваме, че M *е модел на* C, ако за всички основни екземпляри $C\theta$ на C или съществува положителен литерал $P \in C$, такъв се $P\theta \in M$ или съществува отрицателен литерал $N \in C$, такъв се $N\theta \notin M$. M е модел на конюнкция от клаузи S, ако е модел на всяка клауза от S. Интуитивният смисъл на модела на една клауза е да опише случаите, в които тя е *вярна*. В този смисъл моделът може да се разглежда като *изображение* на клаузата в множеството {`истина, лъжа`}, което се задава от *интерпретацията* (смисъла) в някаква проблемна област.

Нека S_1 и S_2 са формули (конюнкции от клаузи). S_2 е *логическо следствие* от S_1, означено с $S_1 \models S_2$, ако всеки модел на S_1 е и модел на S_2. Една формула (множеството от клаузи) S е *удовлетворима*, ако има модел и *неудовлетворима* в противен случай. S е неудовлетворима, ако $S \models \Box$. $S_1 \models S_2$ е еквивалентно на $S_1 \wedge \neg S_2 \models \Box$ (*теорема за дедукцията*).

Нека P е конюнкция от клаузи на Хорн (програма на Пролог без `not`). Сечението на всички модели на P е също модел, който се нарича *минимален ербранов модел* на P и се означава с M_P. Важен резултат в логическото програмиране е, че минималният ербранов модел е единствен за всяка програма P и се състои от множеството от всички основни атоми, които са логически следствия от P, т. е. $M_P = \{A | A \text{ е основен атом}, P \models A\}$.

Неразрешимост на предиктното смятане от I-ви ред

Нека F_1 и F_2 са формули. Задачата за проверка на верността на твърдението $F_1 \models F_2$ е *неразрешима.* (Един клас от задачи, изискващи отговор да или не, е разрешим, ако съществува алгоритъм, който при вход – произволен проблем от класа, винаги спира с отговор да или не.) По-точно, задачата е неразрешима само в случая, когато F_2 не е логическо следствие от F_1 (полуразрешимост).

Правило за логически извод

Правило за логически извод I е процедура, чрез която от една формула S_1 може да се изведе друга формула S_2, означено с $S_1 \vdash_I S_2$. Правилото I е *пълно* и *коректно*, ако $S_1 \vdash_I S_2$ тогава и само тогава, когато $S_1 \models S_2$.

5.1.4 Логическо представяне на агента

- Недостатъци на съждителното смятане при описание на знанията на агента.
- Основни знания:
  - Ако в (X,Y) има звяр, то в съседните на (X,Y) пещери има миризма.
  - Ако в (X,Y) има пропаст, то в съседните на (X,Y) пещери има течение.
- Знания, които се извеждат от наблюденията на агента по време на обхождане на пещерите:
  - Ако в (X,Y) няма звяр и няма пропаст, то тя е безопасна.
  - Пещерата, в която се намира агентът в момента (и още е жив) е безопасна.
  - Ако в пещерата има блясък, то агентът е взел златото.
- Правила за действия:
  - Ако усещаш светлина и си взел златото излизай навън.
  - Ако усещаш блясък вземи златото.
  - Ако не усещаш миризма, течение или удар върви напред.
  - Ако пещерата, към която си се насочил, е безопасна, то върви напред.
  - Ако усещаш миризма, течение или удар завърти се надясно.

5.2 Семинарни упражнения

- Превеждане на формули в клаузна форма.
- Кодиране на знанията на агента на езика на предикатното смятане.
- Превеждане на знанията на агента в клаузна форма.
- Представяне на правилата за действие и ролята на наредбата им.

5.3 Лабораторни упражнения

- Програма за превод на формули от първи ред в клаузна форма (clausify.pl).
- Програмиране на агенти за wumpus world (agents.pl).

Chapter 6

Моделиране на разсъждения - дедуктивен извод

6.1 Лекции

6.1.1 Логически извод

- *Правило за логически извод* I е процедура, чрез която от една формула S_1 може да се изведе друга формула S_2, означено с $S_1 \vdash_I S_2$. Правилото I е *пълно* и *коректно*, ако $S_1 \vdash_I S_2$ тогава и само тогава, когато $S_1 \models S_2$.

- *Естествена дедукция* [Gentzen,1935].

  $\wedge - I$: $\frac{A \; B}{A \wedge B}$
  $\wedge - E$: $\frac{A \wedge B}{A}$ $\frac{A \wedge B}{B}$
  $\vee - I$: $\frac{A}{A \vee B}$ $\frac{B}{A \vee B}$
  $\vee - E$: $\frac{A \vee B \;\; C[A] \;\; C[B]}{C}$
  $\forall - I$: $\frac{F}{\forall c \; F}$
  $\forall - E$: $\frac{\forall x \; F}{F\{x/t\}}$
  $\exists - I$: $\frac{F\{x/t\}}{\exists x \; F}$
  $\exists - E$: $\frac{\exists c \; F \;\; C[F]}{C}$
  $\rightarrow -I$: $\frac{B[A]}{A \rightarrow B}$
  $\rightarrow -E$: $\frac{A \;\; A \rightarrow B}{B}$
  $\neg - I$: $\frac{\Box[A]}{\neg A}$
  $\neg - E$: $\frac{A \;\; \neg A}{F}$ $\frac{\Box}{D}$

- Пример (променливите a, b и c не трябва да участвуват едновременно в зависими формули):

$$
\begin{array}{lll}
 & 2\!: \forall y \; p(a,y) & \\
 & \quad p(a,b) & \forall - E \\
 & \quad \exists x \; p(x,b) & \exists - I \\
1\!: \exists a \; \forall y \; p(a,y) & \quad \forall b \; \exists x \; p(x,b) & \forall - I \\
\multicolumn{2}{c}{\forall b \; \exists x \; p(x,b)} & \exists - E_2 \\
\multicolumn{2}{c}{\exists a \; \forall y \; p(a,y) \rightarrow \forall b \; \exists x \; p(x,b)} & \rightarrow -I_1
\end{array}
$$

- Логически извод = търсене в пространството от формули.

- Логически извод при клаузи

  - *Поглъщане на клаузи* (коректно, но непълно правило за извод). $C \succeq D$, ако съществува субституция θ, такава че $C\theta \subseteq D$. Коректност: Ако $C \succeq D$, то $C \models D$. Непълнота (контрапример): $C = p(f(X)) \leftarrow p(X))$, $D = p(f(f(X))) \leftarrow p(X))$, $C \models D$, $C \not\succeq D$.
  - *Резолюция* (пълно и коректно правило за извод).

6.1.2 Резолюция

- *Резолвента.* Нека C_1 и C_2 са клаузи, такива че съществуват двойка литерали $L_1 \in C_1$ и $L_2 \in C_2$, за които съществува най-общ унификатор μ, който ги прави контрарни, т. е. $L_1\mu = \neg L_2\mu$. Тогава клаузата $C = (C_1 \backslash \{L_1\} \cup C_2 \backslash \{L_2\})\mu$ се нарича *резолвента* на C_1 и C_2. Ясно е, че $C_1 \wedge C_2 \models C$. Например $C_1 = grandparent(X,Y) \leftarrow parent(X,Z), father(Z,Y)$, $C_2 = parent(A,B) \leftarrow father(A,B)$. Резолвентата на C_1 и C_2 е $C = grandparent(X,Y) \leftarrow father(X,Z), father(Z,Y)$. Литералите $\neg parent(X,Z)$ от C_1 и $parent(A,B)$ от C_2 стават контрарни чрез субституцията $\mu = \{A/X, B/Z\}$.

- *Отделяне на променливите.* Клаузите, които се резолвират, не трябва да съдържат общи променливи. Това се осигурява, като при резолюцията не се използват оригиналните клаузи, а подходящи техни варианти. По този начин една клауза може да се резолвира сама със себе си. Например $p(f(X)) \leftarrow p(X)$ може да се резолвира с нейния вариант $p(f(Y)) \leftarrow p(Y)$, при което се получава резолвентата $p(f(f(X))) \leftarrow p(X)$.

- *Резолюция* [Robinson, 65]. Процедура за намиране на опровержение (доказване на неудовлетворимост). Нека S е множество от клаузи. Да означим с $R^n(S)$ следната функция:
  - $R^0(S) = S$
  - $R^i(S) = R^{i-1}(S) \cup \{C | C$ е резолвента на клаузи от $R^{i-1}(S)\}$.

- *Пълнота и коректност при намиране опровержение* (refutation completeness). S е неудовлетворимо тогава и само тогава, когато съществува n, такова че $\Box \in R^n(S)$.

- *Теорема за поглъщането* [Lee, 67]. S – множество от клаузи, C – клауза. Ако $S \models C$, то C или е тавтология, или съществува клауза D, такава че $D \succeq C$ и D се извежда от S чрез резолюция.

- *Стратегии за резолюция.* Съществуват множество различни *стратегии за резолюция*, които се различават по начините за избор на клаузи за резолвиране, от които зависи тяхната ефективност и пълнота.

- *Пролог.* Правилото за извод в езика Пролог проверява дали един атом A е логическо следствие от програмата P, т.е. дали формулата $P \wedge \neg A$ е неудовлетворима чрез т. нар. *входна линейна резолюция* (SLD-резолюция). При тази стратегия на първата стъпка атомът A се резолвира с подходящ вариант на клауза от P, а на всяка следваща стъпка резолвентата, получена на предишната стъпка (линейна стратегия) се резолвира с вариант на клауза от P (входна стратегия).

- *Пълнота на SLD-резолюцията.* Ако $P \models A$ SLD-дървото на опровержението на $P \wedge \neg A$ винаги съдържа път, завършващ с $\Box$ (успешен клон).

- *Непълнота на Пролог* (стратегия търсене в дълбочина).
  $p(a,b)$
  $p(c,b)$
  $p(X,Y) \leftarrow p(X,Z), p(Z,Y)$
  $p(X,Y) \leftarrow p(Y,X)$
  $\leftarrow p(a,c)$

6.1.3 Немонотонен логически извод

- *Разсъждения по подразбиране* [Reiter,80].
  - Правило за извод по подразбиране: $\frac{\not\vdash P}{Q}$
  - Предположение за затворения свят (CWA): $\frac{\not\vdash P}{\neg P}$
  - Противоречивост, немонотонност, зависимост от реда на прилагане: $T = \{\frac{\not\vdash A}{B}, \frac{\not\vdash B}{A}\}$.
  - Присвояване на стойности по премълчаване: $\frac{\not\vdash \exists y \; p(x_1,,\ldots,x_n,y)}{p(x_1,\ldots,x_n,D_y)}$

- *Отрицание чрез неуспех в Пролог.* За дефинирането на `not` в Пролог се използва *немонотонно правило за извод*, което гласи, че ако P е логическа програма, а A – атом, то (`not` A) следва от P, ако A не следва от P и обратното. В общия случай използването на `not` в ЛП създава сериозни теоретични и практически проблеми.

6.2 Семинарни упражнения

- Стратегии за резолюция и тяхната реализация на Пролог.
- Реализация на разсъжденията на агента в wumpus world чрез клаузна резолюция.
- Стратегия за извод на действията на агента.

6.3 Лабораторни упражнения

- Реализация на входна линейна стратегия за клаузи на Хорн чрез ограничение на общия алгоритъм за резолюция (resolve.pl).
- Експерименти с клазузна резолюция. Избягване на зацикляването при Пролог (програмата от точка "Непълнота на Пролог").
- Работа със симулационната среда за интелигентни агенти (wumpus.pl).
- Програмиране на агенти за wumpus world.

Chapter 7

Представне на знания чрез правила

7.1 Лекции

7.1.1 Представяне на знания чрез правила (фигура 7.1)

- Формат:
  - АКО <предпоставки> ТО <заключение>
  - АКО <условие> ТО <действия>
- Кодират фрагменти от знание (не е необходима пълнота).
- Лесно се модифицират и разширяват.
- Лесна реализация на механизми за извод.
- Когнитивни характеристики:
  - Сравнително лесно извличане от експерти.
  - Разбираеми за потребителите.
- Директна реализация на:
  - Пролог (недостатъци: липса на обяснение).
  - Езици за програмиране на правила (OPS5, CLIPS и много други).

7.1.2 Извод назад и експертни системи

- Въвеждане на предпоставки при необходимост чрез въпрос към потребителя.
- Отговор на въпрос "Защо?".
- Отговор на въпрос "Как?".
- Ескпертни системи:
  - База от правила.
  - Машина за извод.
  - Интерфейс.
- Придобиване на знания:
  - Диалог с експерт (празни ЕС, ядра на ЕС).
  - Машинно самообучение (формиране на понятия от примери)

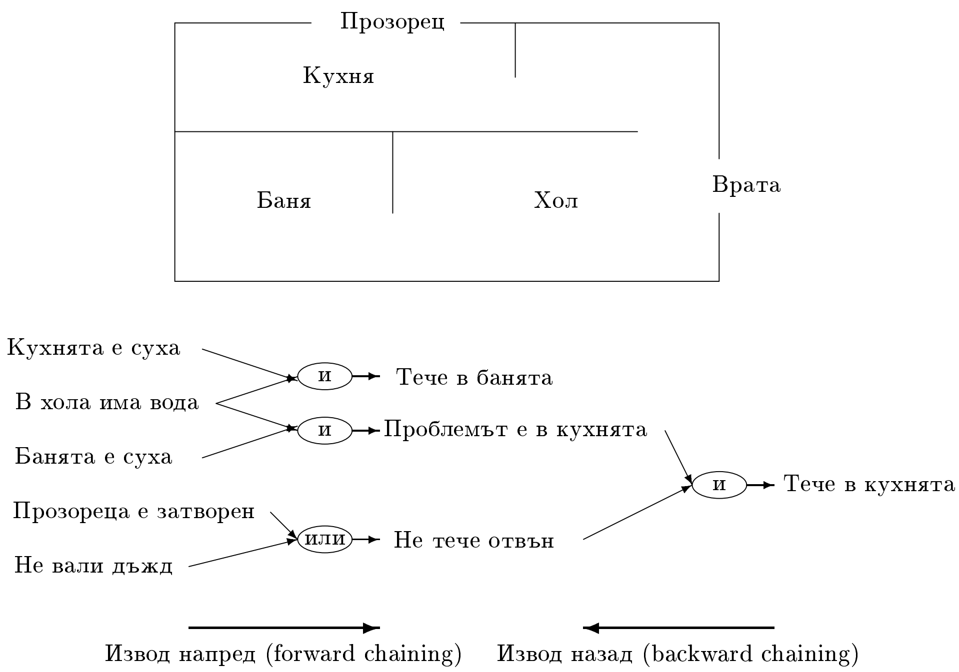


Figure 7.1: Знания и разсъждения за задачата "повреда във водопровода"

7.1.3 Извод напред и продукционни системи

- Работна памет от факти (положителни основни литерали)
- База от продукционни правила: $P_1 \wedge P_2 ... \wedge P_n \rightarrow A_1, A_2, ..., A_m$. P_i – предикати, A_j – действия (добави или изтрий факт от работната памет, вход/изход).
- Машина за извод:
  - Съпоставене и извличане на продукции (списък от "активни" правила).
  - Разрешаване на конфликти (избор на правило за изпълнение): избягване на повторения, по-нови условия, по-специфично правило.
  - Изпълняване на действия

7.1.4 Използуване на фактори за сигурност (фигура 7.2)

- Фактор за сигурност фактите (наблюденията).
- Фактор за сигурност на правилата (пренасяне на условието към заключението).
- Изчисляване на факторите за сигурност в условията на правилата.
- Използуване на праг за отсичане на извода.

7.2 Семинарни упражнения

- Извод назад върху правила – метаинтерпретатор на Пролог със задаване на въпроси и система за обяснение (виж файла rules.pl).
- Продукционни системи – извод напред (Unit Resulting resolution).
- Използуване на not в условията на правилата – извеждане на flies(tweety) чрез извод напред (виж файла rules.pl).
- Създаване на бази от знания за поправка на йерархични системи (виж файла cars.pl).

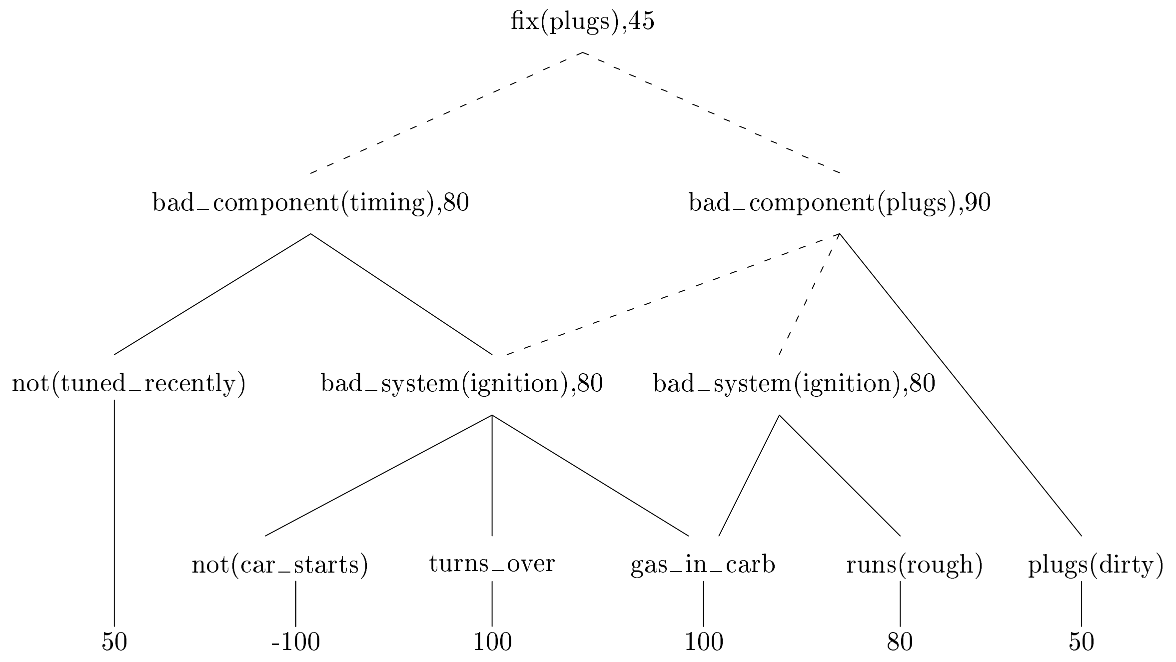


Figure 7.2: Знания и разсъждения за задачата "поправка на автомобил"

7.3 Лабораторни упражнения

- Експерименти с извод напред и назад (rules.pl).
- Разширaване на извода назад с въпрос "защо?" и съхранение на отрицателните отговори (rules.pl).
- Разширяване на извода напред с възможност за използуване на not (rules.pl).
- Експерименти с бази от правила за поправка на йерархични системи (es.pl и cars.pl).

Chapter 8

Семантични мрежи и фрейми

8.1 Лекции

8.1.1 Семантични мрежи

- Представяне на паметта и твърдения на естествен език
  - Подмножество на предикатното смятане от I-ви ред
  - Трансформиране на n-аргументни релации в дву-аргументни (дву-аргументното представяне е по-информативно): *Mary wants John to give her the book.*

```
                                        event(e)
                                        act_of(e,giving)
wants(mary,give(john,book,mary))        actor(e,john)
                                        object(e,book)
                                        recipient(e,mary)
```

  - Представяне на текст на естествен език: *John gave book to Mary. She was grateful to him.*

```
event(e1)                               event(e2).
act_of(e1,giving)                       act_of(e2,gratefullness).
actor(e1,john)      follows(e2,e1)      actor(e2,mary).
object(e1,book)                         object(e2,john).
recipient(e1,mary)
```

8.1.2 Типове релации

- $R(a,b) \leftrightarrow a$ и b са единични обекти;
- $R(a,B) \leftrightarrow a$ принадлежи на B, a е екземпляр на B (instance_of);
- $R(A,B) \leftrightarrow A$ е подмножество на B (is_a);
- $R(A,B) \leftrightarrow$ за всяко a от A съществува b от B, т.че $R(a,b)$;

8.1.3 Фрейми

Структурирани обекти със слотове (стойности или указатели към други фрейми). Понякога им се добавят процедури (подобно на обектното програмиране).

```
               follows
Frame e1   ------------>   Frame e2
--------                   --------
type: event                type: event
act_of:giving              act_of: gratefullness
actor:john                 actor: ------------------>  Frame mary
object:book                object:john                 ----------
```

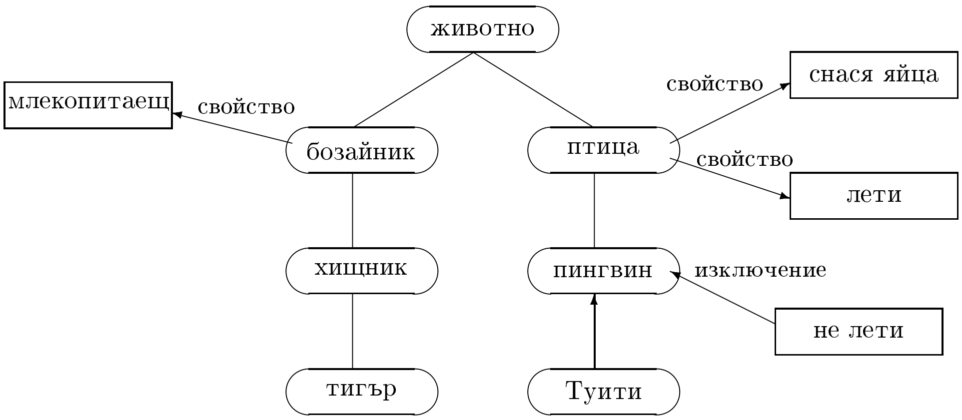


Figure 8.1: Семантична мрежа с наследяване на свойства и изключения

```
recipient: ----------------------------------------> type: person
                                                     age: 20
                                                     sex: female
```

8.1.4 Извод в семантични мрежи

- Съпоставяне
  - Директно съпоставяне: конюнктивен въпрос на Пролог и факти в базата..
  - Съпоставяне с наследяване (системата ARCH на Уйнстън):
    – задължителни релации
    – задължително отсъстващи релации
    – съпоставяне на типа на обектите чрез използуване на таксономия

```
%  object(Name,Parts,IsaRelations,MustRelations,MustNotRelations)
object(arch,[a,b,c],
            [isa(a,4-side),isa(b,4-side),isa(c,polygon)],
            [support(a,c),support(b,c)],
            [touch(a,b)]).
?- object(Object,[X,Y,Z],
            [isa(X,rectangle),isa(Y,rectangle),isa(Z,triangle)],
            [support(X,Z),support(Y,Z)]).
Object=arch
```

- Наследяване на свойства и изключения (фигура 8.1).

```
is_a(птица,животно).
is_a(пингвин,птица).
is_a(туити,пингвин).

млекопитаещо(бозайник).
млекопитаещо(X) :-
    is_a(X,Y),      % наследяване по is_a-йерархията без изключения
    млекопитаещо(Y).

лети(птица).
лети(X) :-
    \+ X=пингвин,   % изключение - не наследява свойството "лети"
    is_a(X,Y),
    лети(Y).
```

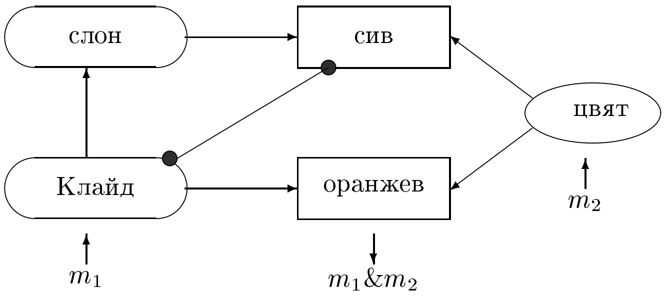


Figure 8.2: Семантична мрежа с предаване на маркери

- Многозначно наследяване.
  - Натрупване на свойства
  - Разрешаване на конфликт
- Разпространаваща се активация (предаване на маркери): *Какъв е цветът на Клайд* ? (фигура 8.2).

8.2 Семинарни упражнения

- Алгоритъм за съпоставяне на семантични мрежи (файл match.pl) – представяне на други обекти, освен арки.
- Реализация на наследяване на свойства и изключения (фигура 8.1).
- Реализация на многозначно наследяване чрез натрупване на свойства.
- Решаване на конфликт при многозначно наследяване чрез дължината на пътя.
- Реализация на алгоритъм за предаване на маркери (с подтискащи връзки и с енергия на маркерите).

8.3 Лабораторни упражнения

- Експерименти със съпоставяне на семантични мрежи (файл match.pl).
- Програма за наследяване на свойства и изключения (фигура 8.1).
- Програма за многозначно наследяване чрез натрупване на свойства.
- Решаване на конфликт при многозначно наследяване чрез дължината на пътя.
- Програма за предаване на маркери (с подтискащи връзки и с енергия на маркерите).

Chapter 9

Представяне на несигурни знания и вероятностни разсъждения

9.1 Лекции

9.1.1 Необходимост от разсъждения с несигурни знания

- Човек обикновено разсъждава в термините на несигурни предположения и твърдения (не 100% истина).
- Класическата логика изисква 100% истинна.
- Директното прилагане на теорията на вероятностите изисква наличието на прекалено много данни (условни вероятности).
- Правилата с фактори за сигурност са прекалено опростен модел.

9.1.2 Представяне на несигурни знания с вероятности

- *Случайна променлива.* Терм в езика за представяне на знания, който може да има няколко (вкл. безкрайно много) възможни стойности.
- *Област на променлива* $dom(x)$ = множеството от възможни стойности на x.
- *Твърдение* – булев израз от *присвоявания на променливи* $(x_i = v_j)$. Например: *(време = дъждовно)* $\vee$ *(болест = грип)* $\vee \neg$ *(температура = повишена)*.
- *Вероятност* (число между 0 и 1) = мярка за увереност в дадено твърдение: $P(A) = 0 \rightarrow 100\%$ увереност, че твърдението A е лъжа. $P(A) = 1 \rightarrow 100\%$ увереност, че твърдението A е истина.
- *Вероятност = мярка за незнание*: Ако твърдението има вероятност между 0 и 1, това не значи, че то е истина "в известна степен", а по-скоро, че ние не знаем със сигурност неговата истинностна стойност.
- *Вероятностното разпределение* задава вероятността на всяка възможна стойност на променливата. Например ако $dom(x) = \{v_1, ..., v_n\}$, то $\sum_{i=1}^{n} P(x = v_i) = 1$.
- *Априорна вероятност* – вероятност при отсъствие на каквато и да е информация.
- *Условна вероятност* – вероятност при наличието на информация за стойностите на други случайни променливи. Например: $P($*температура = повишена* $\mid$ *болест = грип*$)$.
- *Основни формули* (A, B – твърдения):
  - $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$
  - A, B – *независими* (знанието на едното не променя вероятността на другото), $P(A \wedge B) = P(A)P(B)$
  - A, B – *несъвместими* (никога не се случват заедно), $P(A \wedge B) = 0$.
  - Дефиниция на условна вероятност: $P(A|B) = \frac{P(A \wedge B)}{P(B)}$
  - Бейс: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

- Вероятностен модел на проблемната област:
  - *Атомарно събитие* – $(x_1 = v_1) \wedge (x_2 = v_2) \wedge ... \wedge (x_n = v_n)$, където $x_1, ..., x_n$ са случайни променливи. Описва конктретно състояние на проблемната област.
  - *Съвместно разпределение* – n-мерна таблица с m_i клетки ($i = 1, ..., n$) по всяка размерност, ако x_i има m_i възможни стойности. Във всяка клетка се записва вероятността на съответното атомарно събитие. Тъй като атомарните събития са *несъвместими* (взаимно изключващи се) и таблицата съдържа *всички* атомарни събития, сумата от стойностите на всички клетки е 1. Например:

	зъбобол = да	*зъбобол = не*
кариес = да	0.04	0.06
кариес = не	0.01	0.89

  - При дадено съвместно разпределение могат да се изчисляват вероятностите на произволни твърдения. Например (за краткост стойностите на променливите са пропуснати):
    $P(\textit{кариес}) = 0.04 + 0.06 = 0.1$ *(сума на реда)*
    $P(\textit{кариес} \vee \textit{зъбобол}) = 0.04 + 0.01 + 0.06 = 0.11$
    $P(\textit{кариес} \mid \textit{зъбобол}) = P(\textit{кариес} \wedge \textit{зъбобол})\ /\ P(\textit{зъбобол}) = 0.04/(0.04 + 0.01) = 0.8$

9.1.3 Механизми за извод

- Използуване на съвместно разпределение.
- Използуване на формулата на Бейс.
  - Дадени са e – множество от симптоми и $d_1, ..., d_n$ – диагнози. Известни са $P(d_i)$ и $P(e|d_i)$ за $i = 1, ..., n$.
  - Изчисляваме $P(d_i|e)$ за всяко i и намираме най-вероятната диагноза при дадените симтоми e.

$$P(d_i|e) = \frac{P(d_i)P(e|d_i)}{P(e)}$$

  - $P(e)$ може да се изчисли (ако е необходимо) от формулата:

$$\sum_{i=1}^{n} P(d_i|e) = \sum_{i=1}^{n} \frac{P(d_i)P(e|d_i)}{P(e)} = 1$$

$$P(e) = \sum_{i=1}^{n} P(d_i)P(e|d_i)$$

  - Изчисляването на $P(e|d_i)$ изисква изброяване на всички възможни комбинации от стойности на елементарни симтоми (участвуващи в множеството e), които причиняват d_i (*експоненциална сложност*).
- Разсъждения при предположение за независимост на симтомите ("naive" Bayes).
  - Ако приемем, че елементарните симтоми, съставящи e, $e = \{e_1, ..., e_k\}$, са *независими*, то

$$P(e|d_i) = \prod_{j=1}^{k} P(e_j|d_i)$$

  - e_j ($j = 1, ..., k$) могат да се определят лесно от експерт или да се извлекат от експериментални данни.
  - Пример:

Вероятност	*здрав*	*грип*	*алергия*
$P(d)$	0.9	0.05	0.05
$P(\textit{кихане}|d)$	0.1	0.9	0.9
$P(\textit{кашлица}|d)$	0.1	0.8	0.7
$P(\textit{температура}|d)$	0.01	0.7	0.4

    Ако симтомите e са кихане и кашлица без повишена температура, то:
    $P(\textit{здрав}|e) = (0.9)(0.1)(0.1)(0.99)/P(e) = 0.0089/P(e)$
    $P(\textit{грип}|e) = (0.05)(0.9)(0.8)(0.3)/P(e) = 0.01/P(e)$
    $P(\textbf{алергия}|e) = (0.05)(0.9)(0.7)(0.6)/P(e) = 0.019/P(e)$
    $P(e) = 0.0089 + 0.01 + 0.019 = 0.0379$
    $P(\textit{здрав}|e) = 0.23$
    $P(\textit{грип}|e) = 0.26$
    $P(\textbf{алергия}|e) = 0.50$

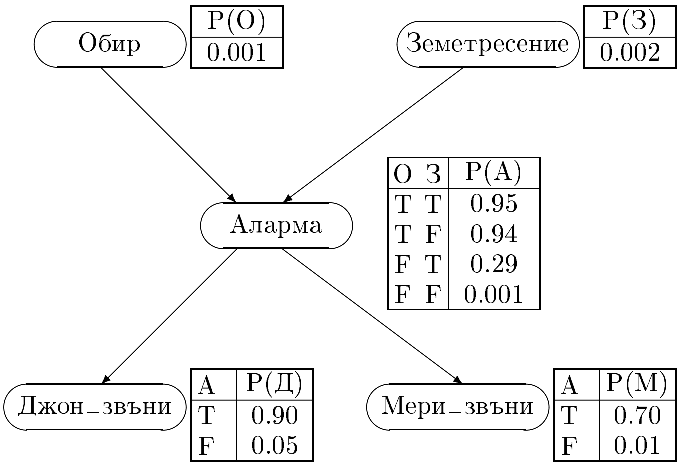


Figure 9.1: Бейсова мрежа

  - Проблем: предположението за независимост на елементарните симтоми е прекалено силно и нереалистично.

- Бейсови мрежи – използуване на *насочен ацикличен граф* за представяне на зависимостите между променливите с цел сбито описание на съвместното им разпределение.

9.1.4 Откъде се появават вероятностите?

- Честотен подход – експерименти.
- Обективен подход – вероятностите са проява на съществуващи свойства на реалния свят.
- Субективен подход – описание на степента на увереност на агента (знанието му за дадено обективно съществуващо явление).
- Избор на клас за сравнение (reference class).
- Пример: каква е вероятността слънцето за изгрее утре? (неопределена, 1, $1-\epsilon$, $\frac{d+1}{d+2}$).

9.1.5 Бейсови мрежи (БМ)

- За всяка случайна променлива – възел.
- Връзките задават директни *причинно-следствени връзки.*
- За всеки възел е дефинирана *таблица с условни вероятности*, която задава вероятността на всяка стойност на променливата във възела в зависимост от всяка една комбинация от стойности на променливите в родителските възли (експоненциален брой редове на таблиците). Пример: фигура 9.1.
- БМ задават неявно съвместното разпределение на променливите си. Нека $x_1, ..., x_n$ са случайни променливи и $P(v_1, ..., v_n)$ е съвместната вероятност те да получат стойностите $v_1, ..., v_n$ съответно. Тогава

$$P(v_1, ..., v_n) = \prod_{i=1}^{n} P(v_i | Parents(x_i)),$$

където $P(v_i|Parents(x_i))$ е условната вероятност за $x_i = v_i$ при условие, че са дадени стойностите на родителските променливи на x_i, $Parents(x_i)$. Например:

$P(\textit{Джон\_звъни, Мери\_звъни, Аларма}, \neg \textit{Обир}, \neg \textit{Земетресение}) = (0.9)(0.7)(0.001)(0.999)(0.998) = 0.000628$

- Условна независимост (използува се при извода). Ако при премахване на подмножество от възлите на мрежата S възлите x и y станат несвързани, то казваме, че x и y са *независими относно* S, т.е. $P(x|y, S) = P(x, S)$.

- Типове връзки:
  - $x \leftarrow y \rightarrow z$: x и z са независими относно y.
  - $x \leftarrow y \leftarrow z$ или $x \rightarrow y \rightarrow z$: x и z са независими относно y.
  - $x \rightarrow y \leftarrow z$: x и z са независими относно y без наличието на информация за стойността на y. При дедена стойност на y (или негови наследници) те стават зависими. Пример: *Обир* $\rightarrow$ *Аларма* $\leftarrow$ *Земетресение*. Ако вече има аларма и се е случило земетресение, вероятността за обир е по-малка.
- Видове извод в БМ. При дадена стойност на подмножество от променливите (наблюдаеми, evidence variables) да се определи вероятността на стойностите на друго подмножество променливи (търсени, query variables).
  - Диагностика – от следствието към причината: $P(\textit{Обир}|\textit{Джон\_звъни})$?
  - Предсказване – от причината към следствието: $P(\textit{Джон\_звъни}|\textit{Обир})$?
  - Междупричинен извод – между причините на дадено следствие: $P(\textit{Обир}|\textit{Земетресение})$?
  - Смесен – комбинация на горните три: $P(\textit{Аларма}|\textit{Джон\_звъни} \wedge \neg \textit{Земетресение})$?
- Алгоритми за извод в БМ.
  - Изчисляване на пълното съвместно разпределение.
  - Разпространение на увереността – изчисляване на вероятността на всички променливи в съответствие с наблюденията. Итеративна модификиция на вероятностните вектори за всяка променлива чрез предаване на съобщения между възлите: "наследници$\rightarrow$родители" и "родители$\rightarrow$наследници", започвайки от наблюденията.
  - Алгоритми, водени от целевите (query) променливи – не изчисляват всички променливи.
  - В общия случай задачата е *експоненциална* по броя на променливите.
  - За едносвързани мрежи (без ненасочени цикли) съществуват *полиномиални* алгоритми.

9.2 Семинарни упражнения

- Изчисляване на априорни и апостериорни вероятности чрез честотен анализ на данни от вида: `example(Id,Diagnosis,[S1=V1,S2=V2,...,Sn=Vn])` (файл tumor.pl).
- Реализация на алгоритъм за класификация (диагностика) "naive Bayes" за работа с данни от горния вид.
- Извод в БМ чрез изчисляване на пълното съвместно разпределение (БМ от фигура 9.1).
- Създаване на БМ за диагностика на автомобили чрез използуване на базата от правила с фактори за сигурност (cars.pl).

9.3 Лабораторни упражнения

- Програмиране на предикати за изчисляване на априорни и апостериорни вероятности чрез честотен анализ на данни (файл tumor.pl).
- Програмиране на алгоритъм за класификация (диагностика) "naive Bayes" за работа с данни от горния вид.
- Експерименти с примерната БМ (bnet_dat.pl) интерпретатора на БМ (bnet.pl).
- Програмиране на БМ от фигура 9.1 и изпълнение с интерпретатора на БМ (bnet.pl).
- Програмиране на БМ за диагностика на автомобили и изпълнене с интерпретатора на БМ (bnet.pl).

Chapter 10

Планиране

10.1 Лекции

10.1.1 Дефиниция на задачата за планиране

- Частен случай на търсене в пространството на състоянията при което се използува логическо представяне на целите и операторите.
- Състоянията имат логическа структура (не са черни кутии, както при търсенето), което дава възможност за ефективно планиране.
- Пример: планиране на действия на робот в "света на кубчетата" (фигура 10.1).

10.1.2 Situation Calculus

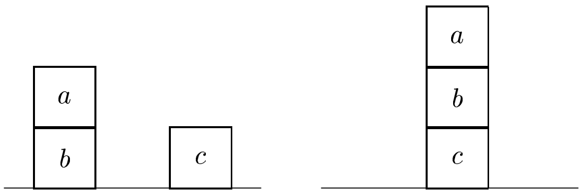
- Използуване на предикатното смятане за формализация на *ситуациите* и *действията* (McCarthy& Hayes, 1969).
- Действия и факти се представят чрез термове, например: $puton(A, B)$, $on(A, B)$.
- Верните факти в дадена ситуация се описват с предиката $holds$. Например $holds(on(A, B), S)$ означава, че A е върху B в ситуацията S.
- *Ситуациите* са термове, описващи състоянията на света. При зададена начална ситуация, например $s0$, всяка следваща се получава чрез функцията $result$. Например $result(puton(a, b), s0)$ представя ситуацията, получена при прилагане на действие $puton(a, b)$ в ситуация $s0$. По този начин можем да записване и по-сложни ситуации: $result(puton(c, a), result(puton(a, b), s0))$.
- Удобно представяне в Пролог: начална ситуация [], действието $puton(A, B)$ в ситуация S води до ситуация $[puton(A, B)|S]$. По този начин всяка ситуация директно описва *плана* (действията са в обратен ред) за достигането си.
- *Аксиомите* се представят с импликации, свързващи предусловия и ефекти, изразени чрез предиката $holds$. Например:

$$holds(clear(A), S) \wedge holds(clear(B), S) \rightarrow holds(on(A, B), result(puton(A, B), S))$$

- Тъй като по този начин ситуацията се променя, необходимо е да се постулира и кой факти остават в сила и в новата ситуация, т.е. *какво не се променя* при извършване на действието. Това става чрез т.нар. *фреймови аксиоми*. Например:

$$holds(clear(X), S) \wedge \neg eq(X, B) \rightarrow holds(clear(X), result(puton(A, B), S))$$

- За всеки един факт, който се използува при представяненто трябва да се зададе съответна фреймова аксиома (frame problem).
- Неефективност при търсенето.



Начално състояние	Целево състояние
$on(a,b)$	$on(a,b)$
$on(b,table)$	$on(b,c)$
$on(c,table)$	$on(c,table)$
$clear(a)$	$clear(a)$
$clear(c)$	

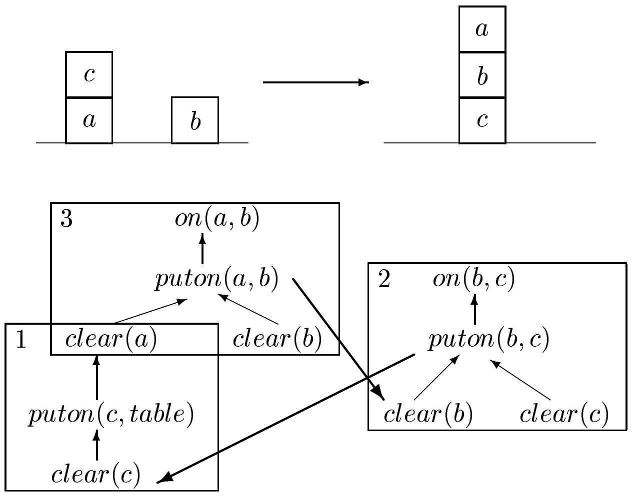
План: $puton(a,table), puton(b,c), puton(a,b)$

Figure 10.1: Планиране в "света на кубчетата"

10.1.3 Продход за планиране STRIPS

- Опит за решаване на фреймовия проблем (Fikes&Nilsson, 1971).
- Всяко действие се представя с три компонента:
  - *Предусловие*: формула, която трябва да е вярна за да се изпълни действието.
  - *Списък за добавяне*: формули, които стават верни след изпълнение на действието.
  - *Списък за изтриване*: формули, които стават невалидни след изпълнение на действието.
- Пример: $puton(X,Y)$
  - Precondition: $clear(X) \wedge clear(Y) \wedge on(X,Z)$
  - Add list: $\{on(X,Y), clear(Z)\}$
  - Delete list: $\{clear(Y), on(X,Z)\}$
- *Предположение*: всяка формула, която е била истина преди действието и не принадлежи на списъка за изтриване, е истина и след изпълнение на действието.
- Независими подцели: цели, които могат да се достигнат независимо една от друга в произволен ред. Пример: начално състояние $= \{on(a,table), on(b,table), on(c,table), on(d,table), clear(a), clear(b), clear(c), clear(d)\}$, цел $= \{on(a,b), on(c,d)\}$.
- Зависими (взаимодестващи си) цели: зависимост от реда на изпълнение на целите. Пример: начално състояние $= \{on(a,table), on(b,table), on(c,table), clear(a), clear(b), clear(c)\}$, цел $= \{on(a,b), on(b,c)\}$. Ако първо се изпълни $puton(b,c)$, а след това $puton(a,b)$ целта е удовлетворена. Какво става при обратен ред?
- Аномалия на Сузман (взаимно развалящи се цели): начално състояние $= \{on(a,table), on(b,table), on(c,a), clear(b), clear(c)\}$, цел $= \{on(a,b), on(b,c)\}$.
- Основен алгоритъм на STRIPS ($+$ и $-$ са операции върху множества):

```
strips_plan(State,Goals,NewState,Plan) :-
%   [Goal|Rest_of_unsatisfied_goals] = Goals - State
    planning_rule(Action,Precondition,Add_list,Delete_list),
    member(Goal,Add_list),
    strips_plan(State,Precondition,State1,Plan1),
%   State2 = State1 + Add_list - Delete_list
    strips_plan(State2,Rest_of_unsatisfied_goals,NewState,Plan2),
%   Plan = Plan1 + [Action|Plan2]
strips_plan(State,_,State,[]).
```



Оптимален план: $puton(c, table), puton(b, c), puton(a, b)$

Figure 10.2: Планиране чрез наредба на частични планове

- Удовлетворяване на зависими цели: многократно изпълнение на основния алгоритъм.
- Оптималност на плановете (минимален брой действия) и ефективност на планиращите системи:
  - Пренареждане на цели.
  - Избор на правило с `Add_list`, включващ следващи цели (начално състояние = $\{on(a, table), on(b, table), on(c, a), clear(b), clear(c)\}$, цел = $\{on(a, c), on(c, b)\}$).
  - Подход на частично-наредени планове: търсене в пространсвото от частични планове (фигура 10.2):
    - план1 < план3 (действието на 1 осигурява предусловието на 3).
    - план1 < план2 (действието на 2 разваля предусловието на 1).
    - план2 < план3 (действието на 3 разваля предусловието на 2).
    - Следователно: план = {план1, план2, план3}.

10.2 Семинарни упражнения

- Алгоритъм за дедуктивно планиране на Пролог.
- Алгоритъм STRIPS на Пролог.
- Въвеждане на ограничения в "света на кубчетата" (напр. размер на кубчето и ограничение за поставяне на по-голямо кубче върху по-малко).
- Формализация на задачата за ханойските кули като задача за планиране.

10.3 Лабораторни упражнения

- Експерименти с дедуктивния планировчик (scalc.pl). Решаване на задачи с 4 и повече кубчета. Разширение на програмата за интерактивна работа (задаване на началното състояние от въпроса) и реализация на алгоритъм за итеративно търсене по нива.
- Сравнителни експерименти с дедуктивния планировчик (scalc.pl) и STRIPS (planner.pl). Сравнение по бързодействие и оптималност на плановете.

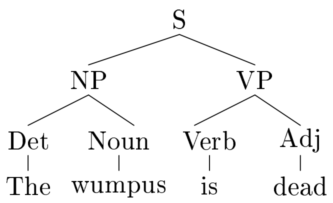
Chapter 11

Обработка на естествен език

11.1 Лекции

11.1.1 Комуникация между интелигентни агенти

1. Намерение $(a1)$: $knows(a2, \neg alive(wumpus, t_n))$.
2. Генериране на текст $(a1)$: The wumpus is dead.
3. Синтезиране на говор $(a1)$: "...thewampusisdead...".
4. Разпознаване на говор $(a2)$: The wumpus is dead.
5. Синтактичен анализ $(a2)$:



6. Семантична интерпретация $(a2)$: $\neg alive(wumpus, now) \vee tired(wumpus, now)$.
7. Прагматична интерпретация (свързване с текущия контекст на $a2$): $\neg alive(wumpus, t_n) \vee tired(wumpus, t_n)$.
8. Премахване на многозначността $(a2)$: $\neg alive(wumpus, t_n)$.
9. Използуване (проверка за съвместимост и включване към знанията на $a2$): $tell(KB, \neg alive(wumpus, t_n))$.

11.1.2 Синтактичен анализ

- Граматика $G = \langle S, T, N, R \rangle$:
  - Множество от терминални символи (думи от езика), $T = \{wumpus, dead, the, is\}$.
  - Множество от нетерминални символи (граматични категории), $N = \{S, NP, VP, Det, Noun, Adj\}$.
  - Начален символ $S \in N$.
  - Граматични правила (rewrite rules) $R : N \to N \cup T$. Пример (контекстно-свободна граматика – един символ отляво):
    $S \to NP\ VP$
    $NP \to Det\ Noun$
    $VP \to Verb\ Adj$
    $Det \to the$
    $Noun \to wumpus$
    $Verb \to is$
    $Adj \to dead$

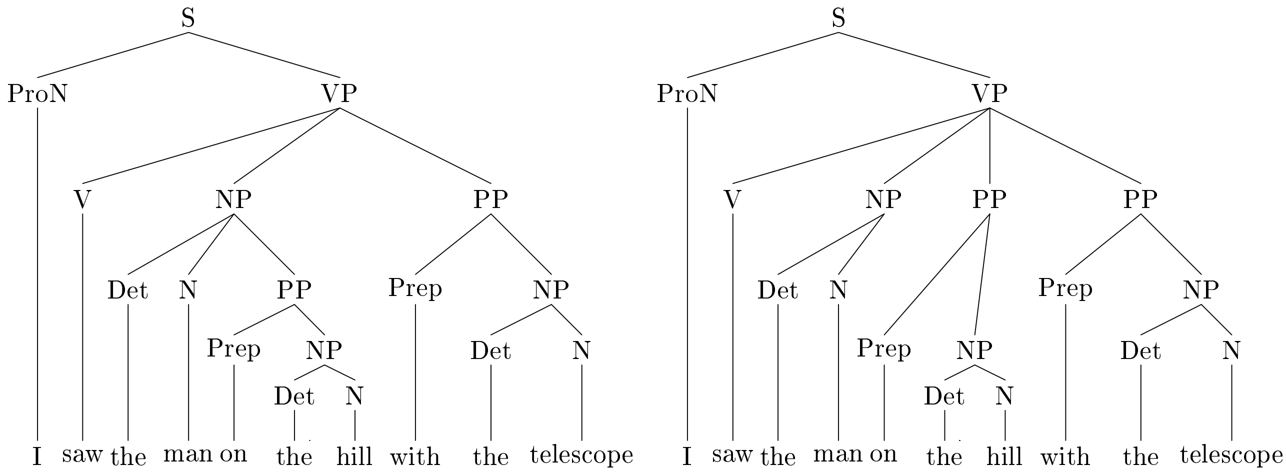


Figure 11.1: Дървета на разбора на многозначно изречение

- Дърво на разбора (фигура 11.1).
- Синтактична многозночност (фигура 11.1).
- Алгоритми за синтактичен анализ - отдолу нагоре (bottom-up parsing) и отгоре надолу (top-down parsing).

```
the wumpus is dead                            S
Det wumpus is dead   (Det -> the)             NP VP                  (S -> NP VP)
Det Noun is dead     (Noun -> wumpus)         Det Noun VP            (NP -> Det Noun)
NP is dead           (NP -> Det Noun)         the Noun VP            (Det -> the)
NP Verb dead         (Verb -> is)             the wumpus VP          (Noun -> wumpus)
NP Verb Adj          (Adj -> dead)            the wumpus Verb Adj    (VP -> Verb Adj)
NP VP                (VP -> Verb Adj)         the wumpus is Adj      (Verb -> is)
S                    (S -> NP VP)             the wumpus is dead     (Adj -> dead)
```

11.1.3 Семантичен анализ

- Представяне на семантиката и семантични ограничения чрез case-frames. Например чрез съпоставяне на синтактичното дърво, `sem(Syntactic_tree,Semantics)`:

```
sem(s(np(Ag),vp(V,np(_,Obj))),[action=Act,agent=Ag,object=Obj]) :-
  meaning(V,Act,_).
sem(s(np(Ag),vp(vp(V,np(_,Obj)),pp(to,np(_,To)))),[action=Act,agent=Ag,object=Obj,Prop=To]) :-
  meaning(V,Act,Prop).
sem(s(np(Ag),vp(vp(V,np(_,Obj)),pp(for,np(For)))),[action=sponsor,agent=Ag,object=Obj,Prop=For]) :-
  meaning(V,sponsor,Prop).

meaning(threw,propel,direction).
meaning(threw,sponsor,recipient).
meaning(gave,giving,subject).
```

- Премахване на многозначността:

```
[tom,threw,a,ball]               -> s(np(tom),vp(threw,np(a,ball)))
                                 -> [action=propel,agent=tom,object=ball]
[tom,threw,a,ball,for,charity]   -> s(np(tom),vp(vp(threw,np(a,ball)),pp(for,np(charity))))
                                 -> [action=sponsor,agent=tom,object=ball,recipient=charity]
[tom,threw,a,ball,to,the,dog]    -> s(np(tom),vp(vp(threw,np(a,ball)),pp(to,np(the,dog))))
                                 -> [action=propel,agent=tom,object=ball,direction=dog]
```

- Координация на частите на изречението (примери от програмата nlp.ncl):

```
?- parse([tom,takes,the,cake,from,the,table,and,eats,it).
pre_location_of(cake) = table
verb_action = eating
subject_of(eating) = tom
d_object_of(eating) = cake
verb_action = taking
subject_of(taking) = tom
d_object_of(taking) = cake

?- parse([tom,takes,the,cake,from,the,table,and,washes,it]).
pre_location_of(cake) = table
verb_action = washing
subject_of(washing) = tom
d_object_of(washing) = table
verb_action = taking
subject_of(taking) = tom
d_object_of(taking) = cake
```

- Превод на изречения във формули от предикатното смятане (пример с програмата qa.pl):

```
?- sentence([every,man,that,lives,loves,a,woman],[],S).
S = all(_A,man(_A)&lives(_A)->exists(_B,woman(_B)&loves(_A,_B)))
```

11.1.4 Система за въпроси и отговори на естествен език

- Превод на изреченията във формули
- Превод в клаузна форма
- Дедуктивен извод чрез клаузна резолюция

11.2 Семинарни упражнения

- Реализация на граматики чрез недетерминиран append и чрез разлика на списъци (например граматика, която разпознава изреченията "john loves mary" и "john eats apples").
- Реализация на граматика, която генерира синтактично дърво. Например изречението "john loves mary" се превежда в терма s(np(john),vp(loves,np(mary))).
- Добавяне на семантични проверки (напр. съгласуване по число или избягване на безмислени изречения като "john eats mary").
- Основни принципи за реализацията на граматика, превеждаща изречения във формули (предиката sentence от програмата qa.pl).

11.3 Лабораторни упражнения

- Програмиране на граматики чрез недетерминиран append и чрез разлика на списъци (например граматика за изреченията "john loves mary" и "john eats apples").
- Реализация на граматика, която генерира синтактично дърво. Добавяне на семантични проверки и генериране на семантика чрез съпоставяне на семантичното дърво.
- Експериметни с предикатите sentence и translate от програмата qa.pl. Кога изреченията се превеждат в повече от една клауза и кога в не-хорнова клауза?
- Генериране на изречения от формули (предиката sentence от qa.pl). Кога програмата се зацикля?
- Експерименти с програмите qa.pl и nlp.ncl.

11.4 Задание за домашна работа

Напишете граматика, която на базата на съпоставяне на синтактичното дърво с case-frames и чрез зададена семантика на необходимите думи да разпознава следните изречения като генерира и съответната им семантика:

```
[tom,threw,a,ball]              -> [action=propel,agent=tom,object=ball]
[tom,threw,a,ball,for,charity] -> [action=sponsor,agent=tom,object=ball,recipient=charity]
[tom,threw,a,ball,to,the,dog]  -> [action=propel,agent=tom,object=ball,direction=dog]
[tom,gave,a,ball,to,the,dog]   -> [action=giving,agent=tom,object=ball,subject=dog]
[tom,gave,a,ball,to,bob]       -> [action=giving,agent=tom,object=ball,subject=bob]
[tom,gave,the,dog,to,bob]      -> [action=giving,agent=tom,object=dog,subject=bob]
[tom,threw,a,ball,for,bob]     -> [action=sponsor,agent=tom,object=ball,recipient=bob]
[tom,threw,a,ball,for,the,dog] -> [action=sponsor,agent=tom,object=ball,recipient=dog]
```

Граматиката трябва да разпознава не само горните изречения, но и всички възможни такива с горните думи, като не трябва да допуска следните конструкции:

- Неправилно използуване на член, например "the,tom", "dog" (без член), "the,charity", и т.н.
- "dog" и "ball" като извършители на действие, или "tom" и "bob" като обекти на действието (напр. "the dog gave tom to bob").
- "charity" като извършител или обект на действие, например "tom gave charity to bob".
- Повторения като "tom gave the dog to the dog".
- "gave a ball" в смисъл на "организирам бал".

Chapter 12

Машинно самообучение (1)

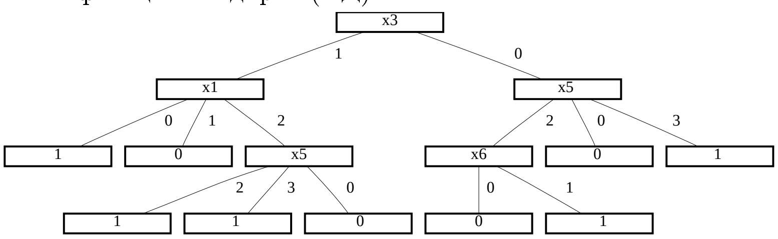
12.1 Лекции

12.1.1 Основни понятия на машинното самообучение

- *Пространство на примерите*: $x_1, x_2, x_4 \in \{0,1,2\}, x_3, x_6 \in \{0,1\}, x_5 \in \{0,1,2,3\}$, $|\{(x_1,x_2,x_3,x_4,x_5,x_6)\}| = 432$
- *Обучаваща/тестова извадка*:

No.	x_1	x_2	x_3	x_4	x_5	x_6	y
1	0	0	0	2	3	1	1
2	2	1	1	1	0	0	0
3	2	1	1	1	3	0	1
4	1	0	0	0	0	1	0
5	2	2	1	2	2	1	1
6	1	2	1	0	2	1	0
7	1	1	0	2	2	1	1
8	0	1	0	1	2	0	0
9	0	0	1	0	0	0	1
10	2	0	1	0	0	0	0
11	1	1	1	0	1	0	?

- *Формиране на понятия/регресия*: $f?$, $y = f(X)$
  - Пропозиционални правила:
    $x_1 = 0 \wedge x_2 = 0 \rightarrow y = 1$
    $x_1 = 1 \wedge x_2 = 1 \rightarrow y = 1$
    $x_1 = 2 \wedge x_2 = 2 \rightarrow y = 1$
    $x_5 = 3 \rightarrow y = 1$
    $x_1 = 0 \wedge x_2 = 1 \wedge x_5 = 0 \rightarrow y = 0$
    $x_1 = 0 \wedge x_2 = 2 \wedge x_5 = 0 \rightarrow y = 0$
    $x_1 = 0 \wedge x_2 = 1 \wedge x_5 = 1 \rightarrow y = 0$
  - Класификационно дърво (КД):



  - Релационно представяне (променливи, релации: $=, \neq$):
    $x_1 = x_2 \vee x_5 = 3 \rightarrow y = 1$
    $x_1 \neq x_2 \wedge x_5 \neq 3 \rightarrow y = 0$
- *Предсказване*: $f(1,1,1,0,1,0) = ?$
- *Кластеризация* (y – неизвестно): $\{(x_1,...,x_6)\} = \{(x_1,...,x_6)|y=0\} \cup \{(x_1,...,x_6)|y=1\}$
- Оценка на грешката: $Error(f) = \frac{|\{(x_1,...,x_6)|f(x_1,...,x_6) \neq y\}|}{432}$

12.1.2 Индуктивна задача

- *Формална система*: *език* L с три подмножества – L_B (*език на основните знания*), L_E (*език на примерите*) и L_H (*език на хипотезите*), и *релация за изводимост* "$\rightarrow$" – съответствие между елементи от езика L.
- *Индуктивна задача*: При дадени *основни знания* $B \in L_B$, *положителни примери* $E^+ \in L_E$ и *отрицателни примери* $E^- \in L_E$, да се намери хипотеза $h \in L_H$, при следните условия:
  1. $B \not\rightarrow E^+$ (*необходимост*);
  2. $B \not\rightarrow E^-$ (*непротиворечивост на основните знания*);
  3. $B \cup h \rightarrow E^+$ (*достатъчност*);
  4. $B \cup h \not\rightarrow E^-$ (*непротиворечивост на хипотезата*).
- Наредба между хипотези: $h \geq h' \Leftrightarrow \{e|e \in L_E, h \rightarrow e\} \supseteq \{e|e \in L_E, h' \rightarrow e\}$.
- Най-обща хипотеза $\top$: $e \in L_E \Rightarrow \top \rightarrow e$.
- Най-специфична хипотеза $\perp$: $e \in E^+ \Leftrightarrow \perp \rightarrow e$ (например $\perp = E^+$).
- Пространство на хипотезите: $\{h|h \geq \perp, h \leq \top\}$
- Оператори за обобщение/специализация: $h' = \rho(h)$, $h' \geq h$ $(h' \leq h)$
- Най-специфично обобщение (lgg): $h = lgg(h_1, h_2)$ ако $h \geq h_1$, $h \geq h_2$ и $\forall h'$, $h' \geq h_1$, $h' \geq h_2 \Rightarrow h' \geq h$.

12.1.3 Език атрибут-стойност

- $L = \{a_1 = v_1, ..., a_n = v_n | v_1 \in V_{a_1}, ..., v_n \in V_{a_n}\}$. V_{a_i} е множество от възможните стойности на атрибута a_i. Например {`цвят=зелен, форма=правоъгълна`}.
- Типове атрибути и дефиниции на релацията *поглъщане* ($\geq$):
  - *Номинални* (няма наредба): $X \in L$, $Y \in L$, $X \geq Y$, ако $X \subseteq Y$; $lgg(X,Y) = X \cap Y$.
  - *Линейни* (пълна наредба) и *структурни* (частична наредба): $X = \{a_1 = x_1, ..., a_n = x_n\}$, $Y = \{a_1 = y_1, ..., a_n = y_n\}$, $X \geq Y$, ако $x_i \geq y_i$ (наредба между стойности) $(i = 1, ..., n)$.
- Език на хипотезите: $L_H = \{C_1 \vee C_2 \vee ... \vee C_n | C_i \in L, i \geq 1\}$ (множество от правила); $h \rightarrow e \Leftrightarrow \exists C_i \in h, C_i \geq e$.
- Индуктивна задача за повече от едно понятие (клас): $E = \cup_{i=1}^{k} E^i$. Положителните примери за класа i са E^i, а отрицателните $E \backslash E^i$.
- Представяне на основните знания: $B = \emptyset$ (използват се *неявно* в релациите за изводимост и наредбите между примерите).

12.1.4 Покриваща стратегия

- Общ алгоритъм ($h = \{h_1, h_2, ..., h_n\}$):
  1. Намери h_i, т. че $\nexists e \in E^-$, $h_i \not\rightarrow e$;
  2. $E^+ = E^+ \setminus \{e|h_i \rightarrow e\}$;
  3. Ако $E^+ \neq \emptyset$ то продължи с 1, в противен случай край.
- Търсене в графа на обобщенията/специализациите (фигура 12.1) – евристики, условие за край.
- Търсене отдолу нагоре чрез lgg: задача за k класа, $E = \cup_{i=1}^{k} E^i$, $B = \emptyset$, $h_i \in L_E$.
  1. Избери $e_i, e_j \in E^k$ за някое k;
  2. $h_{ij} = lgg(e_i, e_j)$;
  3. Ако $\nexists e \in E^l$, за $l \neq k$, такова че $h_{ij} \rightarrow e$ то $E^k = (E^k \setminus \{e|h_{ij} \rightarrow e\}) \cup \{h_{ij}\}$;
  4. Ако стъпки 1 или 3 са невъзможни то край, в противен случай продължи с 1.
  5. Окончателната хипотеза е $h = \cup_{i=1}^{k} E^i$.

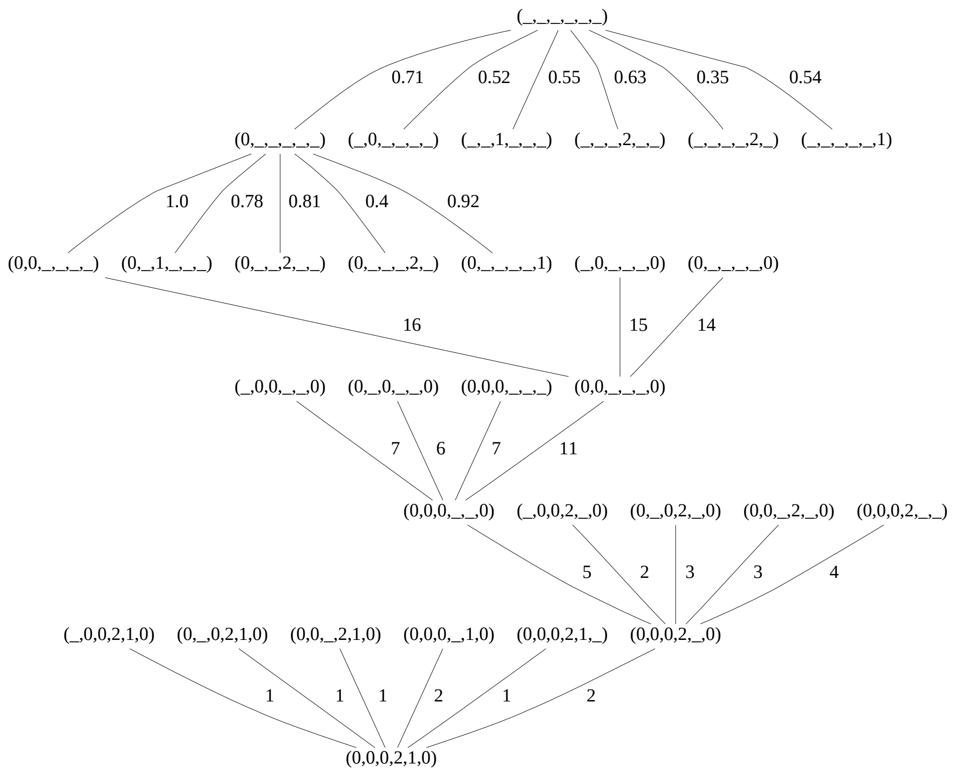


Figure 12.1: Граф на обобщенията/специализациите

12.1.5 Разделяща стратегия

- Базов алгоритъм за построяване на КД (E – множество от примери от различни класове):

  1. Ако всички примери в E са от клас $k \Rightarrow$ листо от тип k. В противен случай се избира атрибут a със стойности $v_1, v_2, ..., v_n$ и се генерира разделящ възел:

     $\boxed{a}$

     $v_1 \quad v_2 \quad \ldots \quad v_n$

  2. $E = \cup_{i=1}^{n} E_i$, т. че всяко E_i съдържа примерите от E, които имат стойност v_i на атрибута a.
  3. Алгоритъмът се прилага рекурсивно за всяко E_i, $i = 1, ..., n$.

- Информационен критерии за избор на атрибут [Quinlan,ID3]:

  – Ентропия (оценка за нивото на "изненада" или "неопределеност" при получаването на n различни съобщения): $I = -\sum_{i=1}^{n} P_i \log_2 P_i$.

  – Информационно съдържание (ентропия) на множество от примери от k класа ($E = \cup_{i=1}^{k} E^i$): $I(E) = \sum_{i=1}^{k} \frac{|E_i|}{|E|} log_2 \frac{|E_i|}{|E|}$.

  – Информация, необходима за довършване на дървото след избора на a: $R(a) = \sum_{i=1}^{n} \frac{|E_i|}{|E|} I(E_i)$.

  – Печалба при избор на атрибут a: $gain(a) = I(E) - R(a)$.

  – Преспециализация и подрязване на дървото.

12.2 Семинарни упражнения

- Рализация на Пролог на оператори за обобщение, специализация и lgg при номинални атрибути.

- Реализация на алгоритъм за търсене чрез обобщение/специализация (фигура 12.1).

- Реализация на алгоритъм за търсене чрез lgg.

- Сравнение на КД за $E = E^+ \cup E^-$, $E^+ = \{2, 5, 6\}$, $E^- = \{1, 3, 4\}$ с първи атрибути форма и цвят:
  1. `[цвят=червен, форма=окръжност]` $-$
  2. `[цвят=червен, форма=квадрат]` $+$
  3. `[цвят=червен, форма=триъгълник]` $-$
  4. `[цвят=син, форма=триъгълник]` $-$
  5. `[цвят=син, форма=квадрат]` $+$
  6. `[цвят=син, форма=окръжност]` $+$

- Изчисляване на $gain$(`форма`) и $gain$(`цвят`) за E.

- Тестване на алгоритми за ИМС чрез обучаваща и тестова извадка. Графика на предсказаната точност при нарастваща обучаваща извадка.

12.3 Лабораторни упражнения

- Експерименти с алгоритмите за търсене чрез обобщение/специализация (search.pl), търсене чрез lgg (simple.pl) и КД (id3.pl) върху данните animals.pl и monks1.pl.

- Сравнение на алгоритмите за търсене чрез обобщение/специализация (search.pl) и търсене чрез lgg (simple.pl) на базата на предсказващата им точност (изчислена съответно чрез предикатите test/3 и test/2).

- Сравнение на вида на правилата, получавани чрез покриващите и разделящите алгоритми.

Chapter 13

Машинно самообучение (2)

13.1 Лекции

13.1.1 Релационен език

- Антиунификация (lgg на термове, $t \geq t_1 \Leftrightarrow \exists \theta, t\theta = t_1$): $t = lgg(t_1, t_2)$, ако $t \geq t_1$, $t \geq t_2$ и $\forall t'$, $t' \geq t_1$, $t' \geq t_2 \Rightarrow t' \geq t$.
- θ-Поглъщане: $C_1 \geq_\theta C_2 \Leftrightarrow \exists \theta \; C_1\theta \subseteq C_2$.
- $lgg_\theta(C_1, C_2) = \{L | L = lgg(L_1, L_2), L_i \in C_i, i = 1, 2\}$
- Релативно lgg: $C = rlgg(C_1, C_2, B) \Leftrightarrow B \cup C \models C_1, B \cup C \models C_2$. Ако C_1 и C_2 са атоми, а B – множество от основни атоми, то $rlgg(C_1, C_2, B) = lgg_\theta((C_1 : -B), (C_2 : -B))$.

13.1.2 Индуктивни алгоритми за извод на клаузи

- Търсене отдолу нагоре:
  1. Избери $e_i, e_j \in E^+$;
  2. $C_{ij} = rlgg(e_i, e_j, B)$;
  3. Редуцирай C_{ij} (θ-обобщение докато C_{ij} не покрива примери от E^-);
  4. $E^+ = E^+ \setminus \{e | B \cup C_{ij} \models e\}$;
  5. Ако $E^+ = \emptyset$ то край, в противен случай продължи с 1.
- Търсене отгоре надолу (фигура 13.1): $E^+ = \{$member(a,[a,b]), member(b,[b]), member(b,[a,b])$\}$, $E^- = \{$member(c,[a,b])$\}$, оператор за специализация:
  - $\rho(C) = C\theta$, $|\theta| = 1$
  - $\rho(C) = C \cup \neg L$

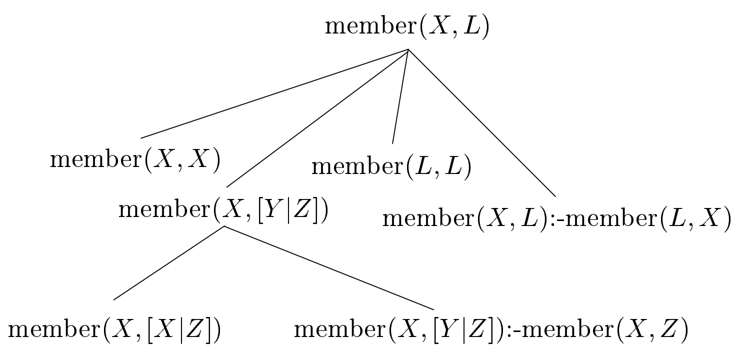


Figure 13.1: Граф на специализациите

Край на специализацията (листа в дървото):

- Успешно листо (добавя се към хипотезата): $\forall e, C \models e \Rightarrow e \in E^+$
- Неуспешно: $\nexists e \in E^+, C \models e$.

13.1.3 Предсказване

- Използуване формулата на Бейс при независимост на атрибутите (naive Bayes).

- Алгоритъм на най-близкия съсед:

No.	x_1	x_2	x_3	x_4	x_5	x_6	y
1	0	0	0	2	3	1	1
2	2	1	1	1	0	0	0
3	2	1	1	1	3	0	1
4	1	0	0	0	0	1	0
5	2	2	1	2	2	1	1
6	1	2	1	0	2	1	0
7	1	1	0	2	2	1	1
8	0	1	0	1	2	0	0
9	0	0	1	0	0	0	1
10	2	0	1	0	0	0	0
X	1	1	1	0	1	0	?

  - Разстояние: $D(X,Y) = \sum_{i=1}^{6} d(x_i, y_i), \quad d(x,y) = \begin{cases} 0, & x = y \\ 1, & x \neq y \end{cases}$

Y	1	2	3	4	5	6	7	8	9	10
$D(X,Y)$	6	3	3	4	5	3	4	4	3	3

  - Сортиране по разстояние:

$D(X,Y)$	3	3	3	3	3	4	4	4	5	6
X	2	3	6	9	10	4	7	8	6	1

  - Избор на преобладаващ клас от първите k примера:

k	1	2	3	4	5	6	7
#1/#0	0/1	1/1	1/2	2/2	2/3	2/4	3/4
$y(X)$	0	?	0	?	0	0	0

13.1.4 Разстояние между термове

- $size(A)$ = брой символи в A − брой различни променливи в A

- Примери: $size(p(X,Y)) = 1$, $size(p(X,X)) = 2$

- $A \geq B \Leftrightarrow size(A) \leq size(B)$

- Разстояние: $D(A,B) = size(A) + size(B) - 2 * size(lgg(A,B))$

- Пример: $D(p(a,a),p(b,b)) = size(p(a,a)) + size(p(b,b)) - 2 * size(p(X,X)) = 3 + 3 - 2 * 2 = 2$

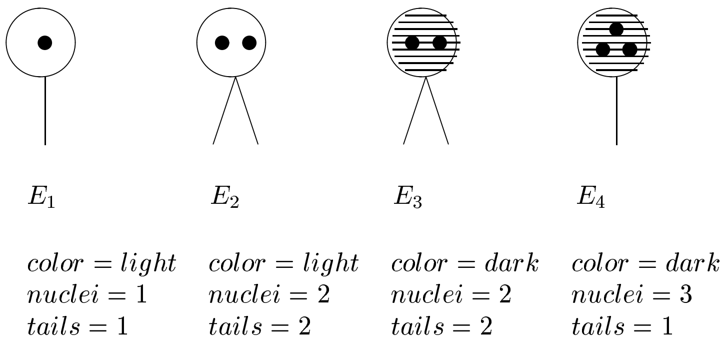
13.1.5 Кластеризация

- Разстояния между множества от примери (min, max, Hausdorff).

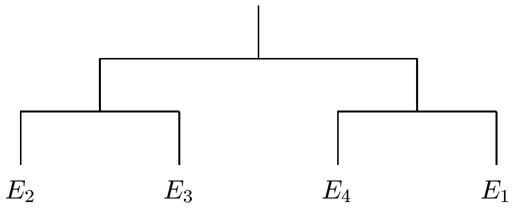
- Общ алгоритъм (E – множество от примери):

  1. Избери $e_i, e_j \in E$, т.че $d(e_i, e_j)$ е минимално;
  2. Обедини e_i и e_j в група и ги замести с нея в E.
  3. Ако езика позволява, построй h_{ij} – обобщение (lgg) на e_i и e_j и изключи от E примерите, които то покрива. Построй поддърво с корен h_{ij} и наследници – всички елементи на E, покривани от h_{ij}.
  4. Продължи с 1 докато в E остане само една група.

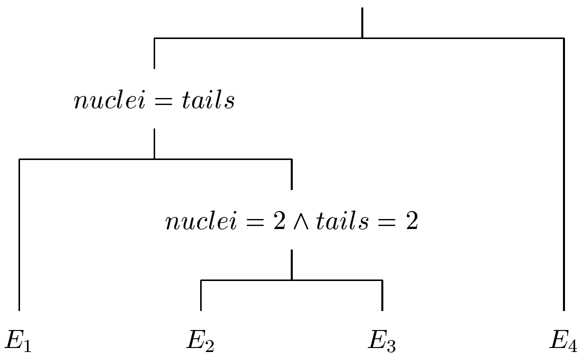
- Пример:



13.1.6 Агломеративна кластеризация



13.1.7 Концептуална кластеризация (стъпка 3 от алгоритъма)



13.2 Семинарни упражнения

- Алгоритъм за редукция на клаузи (стъпка 3 от алгоритъма за търсене чрез lgg).
- Евристики за избор на субституция и литерал при прилагане на оператора за специализация – тип на аргументите и начин на използуване (входни/изходни).
- Построяване на кластеризационни дървета (агломеративно и концептуално) за животните от файла animals.pl.

13.3 Лабораторни упражнения

- Експерименти с rlgg на клаузи (предиката rlgg/4 от ilp.pl).
- Експерименти с алгоритмите за извод на клаузи induce_rlgg/2 и induce_spec/2 (от файла ilp.pl) върху данните за element/2 и append/3 (файлове bot_up1.pl, bot_up2.pl, top_dow1.pl и top_dow2.pl). Експерименти с промяна на броя на положителните (отрицателните) примери.
- Сравнение на бейсовия класификатор (bayes.pl) с алгоритъма за най-близкия съсед (knn.pl) върху данните monks1.pl.

Chapter 14

Критика на силния тезис в ИИ

14.1 Лекции

14.1.1 Силен и слаб тезис в ИИ

14.1.2 Хипотеза за символната обработка (А. Нюел, Х. Саймън, 61)

- Биологично предположение – невроните на най-долното си ниво обработват дискретни символи.
- Психологическо предположение – мисленето е процес на преработка на информация, независимо от процесора (мозък, компютър и т.н.)
- Епистемиологично предположение – това, което човек може да разбере (възприеме от реалния свят), може да се опише формално.
- Онтологично предположение – реалният свят може да се опише формално (например като множество от факти).

14.1.3 Тест за силния тезис – тест на Тюринг

14.1.4 Китайската стая на Сърл

Компютърът може и да е способен да издържи успешно теста на Тюринг, но той не ”разбира” това което върши. Критика на Сърл:

- Системен отговор – системата от човека, програмата и стаята ”разбира”.
- Роботски отговор – ако дадем на системата сензори и ефектори, то тя ще ”разбира”. Символите ще получат интерпретация (grounding) чрез връзката им с реалния свят посредством сензорите и ефекторите.
- Симулация на мозъка – какво ще стане, ако инкрементално подменяме невроните в мозъка с изкуствени техни еквиваленти?
- Липса на обективен тест за ”разбирането” – как да докажем, че една система (естествена или изкуствена) разбира, само чрез наблюдение отвън на нейното поведение.

14.1.5 Аргументи на Р. Пенроуз

- Всяка компютърна програма е еквивалентна на формална система от аксиоми и следователно непълна (теоремата на Гьодел за пълнотата).
- Мисленето е формализируем, но неизчислим (алгоритмично неразрешим) процес.
- Ако компютърът е ”безгрешен” (разсъждава по законите на една формална система), то той не може да бъде интелигентен, не ако не е (например реализира напълно недетерминирано поведение или някакъв непрекъснат физически процес), то доколко интелигентен може да бъде той?