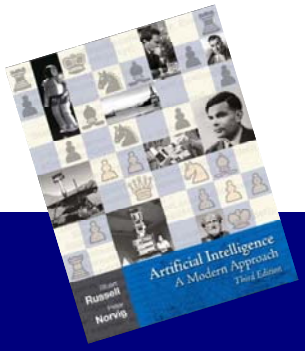


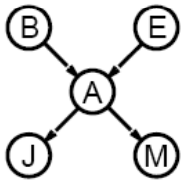
Umělá inteligence II

Roman Barták, KTIML

`roman.bartak@mff.cuni.cz`
`http://ktiml.mff.cuni.cz/~bartak`

3

Pro zopakování



- **Bayesovská síť** zachycuje závislosti mezi náhodnými proměnnými
  - orientovaný acyklický graf (DAG), kde uzly odpovídají náhodným proměnným a mají přiřazenu tabulku **P**(X | Parents(X))
  - kompaktním způsobem reprezentuje úplnou sdruženou distribuci

    $P(x_1,\ldots,x_n) = \Pi_i P(x_i \mid parents(X_i))$
  - umíme sítě konstruovat pro zvolené pořadí proměnných
- **Dnešní program**
  - odvozování v Bayesovských sítích
    - exaktní metody (enumerace, eliminace proměnných)
    - aproximační metody (vzorkovací techniky)

Odvozování enumerací

- Připomeňme, k čemu mají Bayesovské sítě sloužit – zjistit pravděpodobnostní distribuce náhodných proměnných X v dotazu za předpokladu znalosti hodnot e proměnných z pozorování (ostatní proměnné Y jsou skryté).

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha\, \mathbf{P}(X,\mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X,\mathbf{e},\mathbf{y})$$

- hodnotu $\mathbf{P}(X,\mathbf{e},\mathbf{y})$ zjistíme z Bayesovské sítě

$$P(x_1,\ldots,x_n) = \prod_i P(x_i \mid \text{parents}(X_i))$$

- můžeme ještě vhodně přesunout některé členy $P(x_i \mid \text{parents}(X_i))$ před součty

Odvozování enumerací

příklad

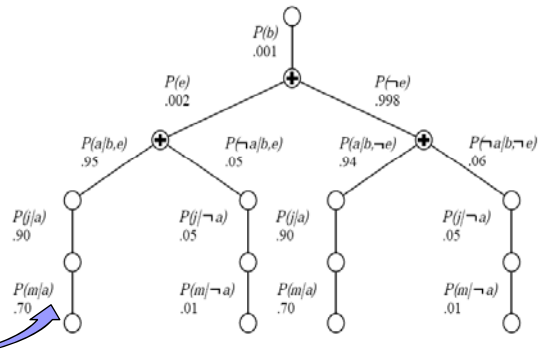
- Nechť máme dotaz, zda došlo k vloupání, pokud Marry i John volají

$$P(b \mid j,m)$$
$$= \alpha \sum_e \sum_a P(b)P(e)P(a|b,e)P(j|a)P(m|a)$$
$$= \alpha\, P(b) \sum_e P(e) \sum_a P(a|b,e)P(j|a)P(m|a)$$

Strukturu výpočtu můžeme zachytit stromovou strukturou.

- je to hodně podobné řešení CSP a SAT

- Všimněme si, že některé části výpočtu se opakují!



Odvozování enumerací

algoritmus

```
function ENUMERATION-ASK(X, e, bn) returns a distribution over X
  inputs: X,  the query variable
          e, observed values for variables E
          bn, a Bayesian network with variables {X} ∪ E ∪ Y

  Q(X) ← a distribution over X, initially empty
  for each value x_i of X do
      extend e with value x_i for X
      Q(x_i) ← ENUMERATE-ALL(VARS[bn], e)
  return NORMALIZE(Q(X))

function ENUMERATE-ALL(vars, e) returns a real number
  if EMPTY?(vars) then return 1.0
  Y ← FIRST(vars)
  if Y has value y in e
      then return P(y | Pa(Y)) × ENUMERATE-ALL(REST(vars), e)
      else return Σ_y P(y | Pa(Y)) × ENUMERATE-ALL(REST(vars), e_y)
          where e_y is e extended with Y = y
```


Eliminace proměnných

- Enumerační metoda zbytečně opakuje některé výpočty.
- Stačí si výsledek zapamatovat a následně použít.

$\mathbf{P}(B \mid j,m)$

$= \alpha\ \mathbf{P}(B)\ \Sigma_e\ P(e)\ \Sigma_a\ \mathbf{P}(a|B,e)P(j|a)P(m|a)$

$= \alpha\ \mathbf{f_1}(B)\ \Sigma_e\ \mathbf{f_2}(E)\ \Sigma_a\ \mathbf{f_3}(A,B,E)\ \mathbf{f_4}(A)\ \mathbf{f_5}(A)$

- Činitelé $\mathbf{f_i}$ jsou matice (tabulky) pro dané proměnné.
- Vyhodnocení provedeme **zprava doleva**
  - **násobení činitelů** je násobení po prvcích (ne násobení matic)
  - vysčítáním činitelů **eliminujeme** příslušnou proměnnou

Eliminace proměnných

operace s tabulkami

- Máme-li dvě tabulky, potom jejich **součin** je tabulka nad sjednocením proměnných z obou tabulek.

$f(X_1,...,X_j,Y_1,...,Y_k,Z_1,...Z_l) = f(X_1,...,X_j,Y_1,...,Y_k) \cdot f(Y_1,...,Y_k,Z_1,...Z_l)$

A	B	$f_1(A,B)$	B	C	$f_2(B,C)$	A	B	C	$f_3(A,B,C)$
T	T	0.3	T	T	0.2	T	T	T	0.06 = 0.3*0.2
T	F	0.7	T	F	0.8	T	T	F	0.24 = 0.3*0.8
F	T	0.9	F	T	0.6	T	F	T	0.42 = 0.7*0.6
F	F	0.1	F	F	0.4	T	F	F	0.28 = 0.7*0.4
						F	T	T	0.18 = 0.9*0.2
						F	T	F	0.72 = 0.9*0.8
						F	F	T	0.06 = 0.1*0.6
						F	F	F	0.04 = 0.1*0.4

- Při **vysčítání** dojde k eliminaci proměnné

$\Sigma_a f(A,B,C) = f(B,C)$

$$\begin{bmatrix} 0.06 & 0.24 \\ 0.42 & 0.28 \end{bmatrix} + \begin{bmatrix} 0.18 & 0.72 \\ 0.06 & 0.04 \end{bmatrix} = \begin{bmatrix} 0.24 & 0.96 \\ 0.48 & 0.32 \end{bmatrix}$$

Eliminace proměnných

algoritmus

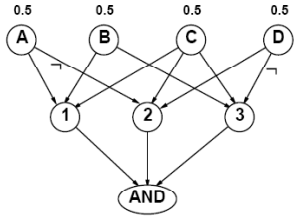
```
function ELIMINATION-ASK(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, evidence specified as an event
          bn, a belief network specifying joint distribution P(X_1, ..., X_n)
  factors ← [ ]; vars ← REVERSE(VARS[bn])
  for each var in vars do
      factors ← [MAKE-FACTOR(var, e)|factors]
      if var is a hidden variable then factors ← SUM-OUT(var, factors)
  return NORMALIZE(POINTWISE-PRODUCT(factors))
```

- Algoritmus funguje pro libovolné uspořádání proměnných.
- Složitost je dána velikostí největšího činitele (tabulky) v průběhu výpočtu.
- Vhodné je proto pro eliminaci vybrat proměnnou, jejíž eliminací vznikne nejmenší tabulka.

Složitost problému

- Eliminace proměnných urychluje odvozování, ale jak moc?
- Pokud je Bayseovská síť **poly-strom** (mezi každými dvěma vrcholy vede maximálně jedna neorientovaná cesta), potom časová a prostorová složitost odvozování eliminací proměnných odpovídá velikosti tabulek $\mathbf{O(n.d^k)}$.
- Pro **více propojené sítě**, je to horší:
  - 3SAT lze redukovat na odvození v Bayesovské síti, takže odvození je NP-těžké

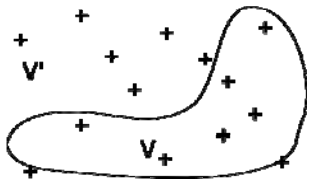
1. A v B v C
2. C v D v ¬A
3. B v C v ¬D



  - odvozování v Bayesovské síti je ekvivalentní zjištění počtu řešení SAT-formule, je tedy **#P-těžké**

Vzorkovací metody

- Exaktní odvozování je výpočtově náročné, můžeme ale použít také aproximační techniky založené na **metodě Monte Carlo**.
- Monte Carlo algoritmy slouží pro odhad hodnot, které je těžké spočítat exaktně.
  - vygeneruje se množství vzorků
  - hledaná hodnota se zjistí statisticky
  - více vzorků = větší přesnost



- **Pro Bayesovské sítě** ukážeme dva přístupy
  - přímé vzorkování
  - vzorkování s Markovskými řetězci

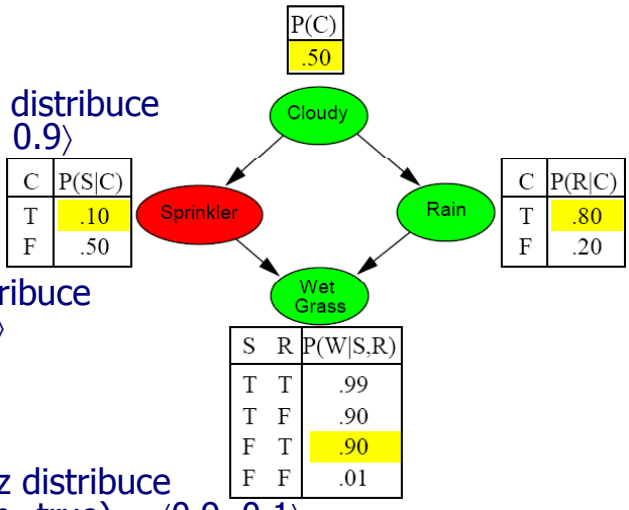
Přímé vzorkování

- Vzorkem pro nás bude ohodnocení náhodných proměnných.
- Vzorek je potřeba generovat tak, aby „odpovídal" tabulkám v Bayesovské síti.
  - uzly (proměnné) bereme v topologickém uspořádání
  - ohodnocení rodičů nám dá pravděpodobnostní distribuci hodnot aktuální náhodné proměnné
  - náhodně vybereme hodnotu podle této distribuce
- Nechť N je počet vzorků a $N(x_1,\ldots,x_n)$ je počet výskytů jevu $x_1,\ldots,x_n$, potom
  $P(x_1,\ldots,x_n) = \lim_{N\to\infty} (N(x_1,\ldots,x_n)/N)$

function PRIOR-SAMPLE(bn) **returns** an event sampled from bn
  inputs: bn, a belief network specifying joint distribution $\mathbf{P}(X_1,\ldots,X_n)$
  $\mathbf{x} \leftarrow$ an event with n elements
  for $i = 1$ **to** n **do**
    $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
      given the values of $Parents(X_i)$ in $\mathbf{x}$
  return $\mathbf{x}$

Přímé vzorkování
příklad

- Vybereme hodnotu pro Cloudy z distribuce
  P(Cloudy) = ⟨0.5, 0.5⟩
  nechť je to **true**
- Vybereme hodnotu pro Sprinkler z distribuce
  P(Sprinkler | Cloudy=true) = ⟨0.1, 0.9⟩
  nechť je to **false**
- Vybereme hodnotu pro Rain z distribuce
  P(Rain | Cloudy=true) = ⟨0.8, 0.2⟩
  nechť je to **true**
- Vybereme hodnotu pro WetGrass z distribuce
  P(WetGrass | Sprinkler=false, Rain=true) = ⟨0.9, 0.1⟩
  nechť je to **true**



Získali jsme vzorek Cloudy=true, Sprinkler=false, Rain=true, WetGrass=true

Pravděpodobnost jeho získání je zřejmě 0.5 * 0.9 * 0.8 * 0.9 = 0.324

Vzorkování se zamítáním

- Nás ale zajímá **P**(X | e)!
- Ze vzorků, které vygenerujeme, vezmeme jen ty, které jsou kompatibilní s e (ostatní zamítneme).

  P(X | e) ≈ **N**(X,e) / N(e)

```
function REJECTION-SAMPLING(X, e, bn, N) returns an estimate of P(X|e)
  local variables: N, a vector of counts over X, initially zero
  for j = 1 to N do
      x ← PRIOR-SAMPLE(bn)
      if x is consistent with e then
          N[x] ← N[x]+1 where x is the value of X in x
  return NORMALIZE(N[X])
```

- Nechť v našem příklad vygenerujeme 100 vzorků, z toho u 27 platí Sprinkler= true a z nich u 8 je Rain = true a u 19 je Rain = false. Potom

  P(Rain | Sprinkler=true) ≈ Normalize(⟨8, 19⟩) = ⟨0.296, 0.704⟩
- Hlavní nevýhoda metody je **zamítání příliš mnoha vzorků**!

Vážení věrohodností

- Místo zamítání vzorků je efektivnější generovat pouze vzorky vyhovující pozorování e.
  - zafixujeme hodnoty z pozorování e a vzorkujeme pouze ostatní proměnné
  - pravděpodobnost získání vzorku je $P(z,e) = \Pi_i P(z_i \mid parents(z_i))$
  - To ale není to, co potřebujeme! Ještě nám chybí $w(z,e) = \Pi_j P(e_j \mid parents(e_j))$.
  - Každý vzorek tedy doplníme o příslušnou **váhu**:

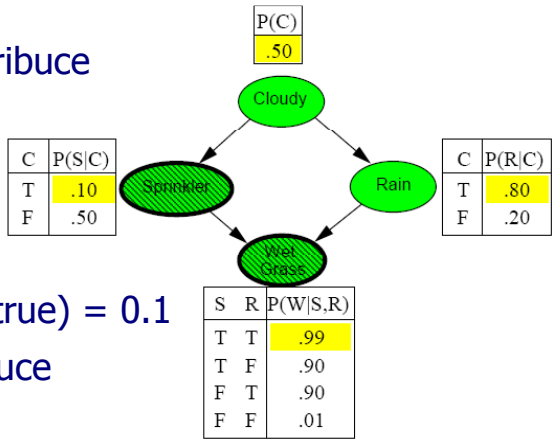
    P(X | e) ≈ α **N**(X,e) w(X,e)

Vážení věrohodností

příklad

Nechť zpracováváme dotaz **P**(Rain | Sprinkler=true,WetGrass=true))

- Počáteční váha vzorku w = 1.0
- Vybereme hodnotu pro Cloudy z distribuce
  P(Cloudy) = ⟨0.5, 0.5⟩
  nechť je to **true**
- Hodnotu Sprinkler=**true** známe,
  ale upravíme váhu
  w ← w * P(Sprinker=true | Cloudy=true) = 0.1
- Vybereme hodnotu pro Rain z distribuce
  P(Rain | Cloudy=true) = ⟨0.8, 0.2⟩
  nechť je to **true**
- Hodnotu WetGrass=**true** známe, ale upravíme váhu
  w ← w * P(WetGrass=true | Sprinker=true,Rain=true) = 0.099



Získali jsme vzorek
Cloudy=true, Sprinkler=false, Rain=true, WetGrass=true,
který má váhu 0.099

Vážení věrohodností

algoritmus

function LIKELIHOOD-WEIGHTING($X, \mathbf{e}, bn, N$) **returns** an estimate of $P(X|\mathbf{e})$
  local variables: $\mathbf{W}$, a vector of weighted counts over X, initially zero
  for $j = 1$ to N **do**
    $\mathbf{x}, w \leftarrow$ WEIGHTED-SAMPLE(bn)
    $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ where x is the value of X in $\mathbf{x}$
  return NORMALIZE($\mathbf{W}[X]$)

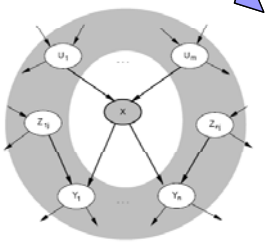
---

function WEIGHTED-SAMPLE($bn, \mathbf{e}$) **returns** an event and a weight
  $\mathbf{x} \leftarrow$ an event with n elements; $w \leftarrow 1$
  for $i = 1$ **to** n **do**
    if X_i has a value x_i in $\mathbf{e}$
      then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
      else $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return $\mathbf{x}$, w

Metoda MCMC

- Přímé vzorkování generuje každý vzorek „od nuly"
- Můžeme to dělat i jinak:
  - začneme od náhodně vygenerovaného vzorku, který odpovídá pozorování e
  - pro vybranou proměnnou X (mimo pozorování e) vybereme hodnotu vzorkováním podle jejího Markovského obalu
    P(x | mb(X)) = P(x | parents(X)) $\Pi_{Z \in Children(X)}$ P(z | parents(Z))
  - získáme tzv. **Markovský řetězec**, odtud se metoda jmenuje **Markov Chain Monte Carlo** (MCMC)
  - vzorky zpracujeme podobně jako u přímého vzorkování

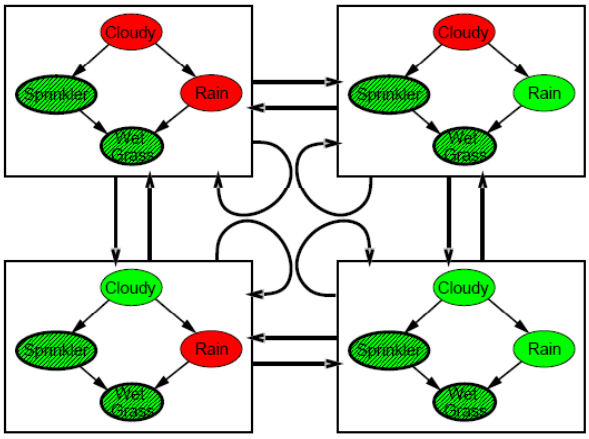
```
function MCMC-Ask(X, e, bn, N) returns an estimate of P(X|e)
  local variables: N[X], a vector of counts over X, initially zero
                   Z, the nonevidence variables in bn
                   x, the current state of the network, initially copied from e
  initialize x with random values for the variables in Y
  for j = 1 to N do
      for each Z_i in Z do
          sample the value of Z_i in x from P(Z_i|mb(Z_i))
              given the values of MB(Z_i) in x
          N[x] ← N[x] + 1 where x is the value of X in x
  return NORMALIZE(N[X])
```



Markovské řetězce

příklad

- Uvažujme pozorování Sprinkler=true, WetGrass=true
- Dostaneme čtyři stavy, přes které Markovský řetězec prochází.



- Uvažujme, že projdeme 100 stavů,
  u 31 stavů platí Rain=true
  a 69 stavů platí Rain=false
- Potom dostaneme:
  P(Rain | Sprinkler=true, WetGrass=true)
  = Normalize(⟨31, 69⟩) = ⟨0.31, 0.69⟩

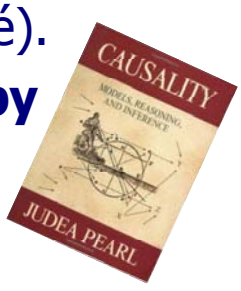
Jak MCMC funguje?

- Při dostatečné délce Markovský řetězec konverguje ke **stacionární distribuci** – počet výskytů stavu/vzorku je proporciální jeho posteriorní pravděpodobnosti.
- zavedeme označení:
  - $q(\mathbf{x}\rightarrow\mathbf{x}')$ pro pravděpodobnost přechodu z x do x' (určuje Markovský řetězec)
  - $\pi_t(\mathbf{x})$ pro pravděpodobnost dosažení stavu x po t krocích
- obecně platí:
  - $\pi_{t+1}(\mathbf{x}') = \Sigma_{\mathbf{x}}\, \pi_t(\mathbf{x})\, q(\mathbf{x}\rightarrow\mathbf{x}')$
- pro stacionární distribuci potom chceme
  - $\pi(\mathbf{x}') = \Sigma_{\mathbf{x}}\, \pi(\mathbf{x})\, q(\mathbf{x}\rightarrow\mathbf{x}')$
  - což platí například pokud $\pi(\mathbf{x})\, q(\mathbf{x}\rightarrow\mathbf{x}') = \pi(\mathbf{x}')\, q(\mathbf{x}'\rightarrow\mathbf{x})$
- předpokládejme, že měníme hodnotu proměnné X_i z x_i na x_i', ostatní proměnné označme $\mathbf{Y}_i$ a jejich aktuální hodnoty $\mathbf{y}_i$
  - $q(\mathbf{x}\rightarrow\mathbf{x}') = q((x_i,\mathbf{y}_i)\rightarrow(x_i',\mathbf{y}_i)) = P(x_i'|\, \mathbf{y}_i,\mathbf{e}) = P(x_i'|\, mb(X_i))$
  - hovoříme o tzv. **Gibssově vzorkování**
  - $\pi(\mathbf{x})\, q(\mathbf{x}\rightarrow\mathbf{x}') = P(\mathbf{x}|\mathbf{e})\, P(x_i'|\, \mathbf{y}_i,e) = P(x_i,\mathbf{y}_i|\mathbf{e})\, P(x_i'|\, \mathbf{y}_i,e)$
    $= P(x_i|\mathbf{y}_i,\mathbf{e})\, P(\mathbf{y}_i|\mathbf{e})\, P(x_i'|\, \mathbf{y}_i,e)$
    $= P(x_i|\mathbf{y}_i,\mathbf{e})\, P(x_i',\mathbf{y}_i|\, e) = q(\mathbf{x}'\rightarrow\mathbf{x})\, \pi(\mathbf{x}')$ **řetězcové pravidlo**

Neurčitost a UI

krátké historické shrnutí

- Pro modelování **neurčitosti** jsme použiti **teorii pravděpodobnosti** (podobně jako ve fyzice či ekonomii), ale bylo tomu tak vždy?
- První expertní systémy (kolem roku **1970**) používali **čistě logický přístup** bez práce s neurčitostí, což se ukázalo nepraktické.
- Další generace ES zkusila **pravděpodobnostní techniky**, ale ty měly **problém** se **škálovatelností** (efektivní odvozování založené na Bayesovských sítích ještě nebylo známé).
- Proto byly zkoušeny **alternativní přístupy** (**1975-1988**) pro modelování neurčitosti, ignorance a vágnosti.



Alternativní přístupy

- **Pravidlově orientované systémy** staví na úspěchu logického odvozování, které obohacují o prvek **nejistoty**.
- **Dempster-Shaferova teorie** se soustředí na otázku ignorance a pracuje s popisem stupně **domnění** (belief).
- Logika i pravděpodobnost pracují s faktem, že tvrzení je buď pravdivé nebo není. **Fuzzy logika** umožňuje vyjadřovat **vágnost** tvrzení.

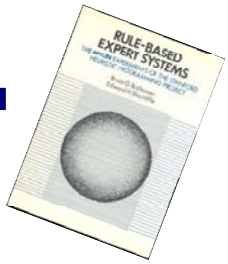
Pravidlové systémy

základní vlastnosti

- Klasické **logické pravidlové systémy** staví na třech vlastnostech:
  - **lokálnost**: z A a A⇒B odvodíme B nezávisle na ostatních pravidlech (v pravděpodobnosti musíme uvažovat všechna pozorování)
  - **oddělenost**: jakmile dokážeme, že platí B, můžeme s tímto faktem pracovat odděleně od důkazu (v pravděpodobnosti je zdroj pozorování důležitý pro další zpracování)
  - **pravdivostní skládání**: pravdivost složených tvrzení se odvozuje od pravdivosti jejich komponent (kombinace pravděpodobnosti tímto způsobem nepracuje, pokud se nejedná o nezávislost)

Pravidlové systémy
problémy

- **Vlastnosti** pravidlových systémů z pohledu neurčitosti
  - uvažujme jevy
    H_1 „při prvním hodu mincí padne hlava“, O_1 „při prvním hodu mincí pane orel“, H_2 „při druhém hodu padne hlava“
    - pravděpodobnost každého z těchto jevů je 0.5
    - $P(H_1 \vee H_1) = 0.5$, $P(H_1 \vee O_1) = 1$, $P(H_1 \vee H_2) = 0.75$, což je ve sporu se skládáním pravdivosti dílčích tvrzení
  - uvažujme pravidla
    $Sprinkler \rightarrow_{0.9} WetGrass$ a $WetGrass \rightarrow_{0.9} Rain$
    - pokud je rozprašovač zapnutý, zvýší to víru, že trávník je mokrý, což dále zvýší víru, že pršelo; nemělo by to být opačně?
    - systém totiž věří, že $Sprinkler \rightarrow Rain$
- **Jak to tedy může fungovat?**
  - systémy používají **pravidla** jen **v jednou směru**
  - **pozorování** se vkládají pouze **na začátek**
  - typickým reprezentantem byl systém **MYCIN**



Dempster-Shafer

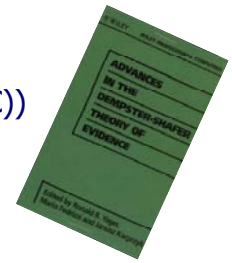
- zobecnění teorie pravděpodobnosti o **modelování ignorance**
  - uvažujme hod mincí
    - pokud je mince pravá, je pravděpodobnost hlavy 0.5 a orla 0.5
    - pokud je mince „falešná“, ale nevíme jak, tak je pravděpodobnost hlavy také 0.5 a orla 0.5 (ignorance)
- Dempster-Shafer teorie používá **míru domnění** (belief) a **míru plauzibility**
  - $bel(X) \leq pl(X)$
  - $pl(X) = 1 - bel(\neg X)$

  pracujeme tak vlastně s intervalem, kde se nachází pravděpodobnost
  interval popisuje míru nejistoty o daném tvrzení
  - $bel(X) \leq P(X) \leq pl(X)$
  - uvažujme tvrzení A „ve sklepě je mrtvá myš“, kde $bel(A) = 0.5$ a $pl(A) = 0.8$
    - naše víra v pravdivost tvrzení je $bel(A) = 0.5$
    - naše víra v nepravdivost tvrzení je $bel(\neg A) = 0.2$ (=1-0.8)

Dempster-Shafer

jak to funguje?

- Podobně jako v pravděpodobnosti pracujeme s množinou elementárních tvrzení X, kde máme dánu **míru m** na podmnožinách
  - $m: 2^X \rightarrow [0,1]$
  - $m(\emptyset) = 0$
  - $\Sigma_A\, m(A) = 1$
- **Míru domnění a plauzibility** potom definujeme takto:
  - $bel(A) = \Sigma_{B \subseteq A}\, m(B)$
  - $pl(A) = \Sigma_{B \cap A \neq \emptyset}\, m(B)$
- Spojování různých vstupů se provádí pomocí **Dempsterova pravidla**:
  - $m_{1,2}(\emptyset) = 0$
  - $m_{1,2}(A) = (m_1 \oplus m_2)(A) = \alpha\, \Sigma_{B \cap C = A \neq \emptyset}\, m_1(B)\, m_2(C)$
    - α je normalizační konstanta $\alpha = 1 / (1 - \Sigma_{B \cap C = \emptyset}\, m_1(B)\, m_2(C))$
    - slouží pro ignorování konfliktních vstupů, což může dávat neintuitivní výsledky



Fuzzy množiny

- **Teorie fuzzy množin** se týká zachycení **vágnosti** pojmů.
  - Karel měří 180 cm. Je Karel vysoký?
  - **fuzzy množina** definuje predikát „vysoký" implicitně množinou svých prvků – množina **nemá přesný okraj**
- **Fuzzy logika** je metoda pro uvažování o fuzzy množinách.
  - je založena na pravdivostním skládání T-norem
    - $T(A \wedge B) = \min(T(A),T(B))$
    - $T(A \vee B) = \max(T(A),T(B))$
    - $T(\neg A) = 1 - T(A)$

Problém: nechť T(Vysoký(Karel)) = 0.6, potom $T(\text{Vysoký(Karel)} \wedge \neg\, \text{Vysoký(Karel)}) = 0.4$

Neberou se v úvahu vztahy mezi složkami tvrzení!

Teorie fuzzy množin není metodou pro práci s neurčitostí!