10. Principy expertních systémů.

Expertní systém (ES) je programový systém, který simuluje činnost expertů při řešení složitých problémově zaměřených úloh.

Použití ES předpokládá splnění následujících podmínek:

- Problémová oblast musí být dostatečně úzká, aby do ES bylo možné uložit všechny relevantní znalosti.
- Problémová oblast musí být dostatečně složitá, aby expertíza měla smysl.
- Musí existovat experti v dané problematice a musí být ochotni poskytnout své znalosti.
- Přínos expertízy musí převyšovat náklady spojené s tvorbou a provozováním ES.

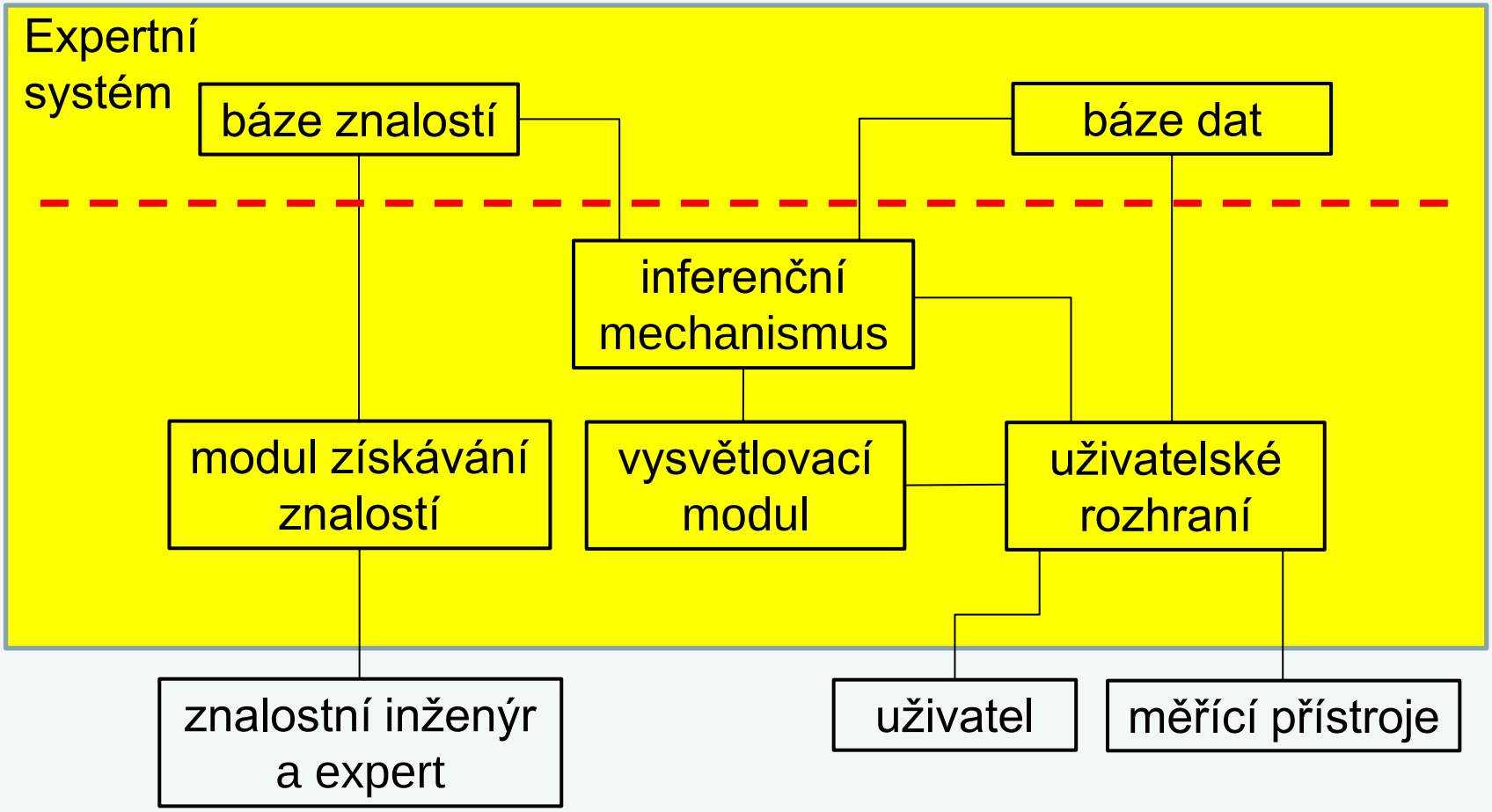
Typické problémové oblasti vhodné pro ES:

- interpretace (rozpoznávání situací, např. havarijních)
- predikce (předpověď důsledků dané situace)
- diagnostika (určování stavu systému)
- konstrukce (skládání/kompozice objektu)
- plánování (stanovení posloupnosti akcí)
- monitorování (sledování údajů)
- řízení (vyhodnocení údajů + akční zásahy)
- učení (speciální typ řízení)

Rozdělení ES podle charakteru řešených úloh:

- diagnostické (jejich úkolem je určit, která z předem daných hypotéz nejlépe koresponduje s daty dané konkrétní úlohy).
- Plánovací (jejich úkolem je nalézt posloupnost kroků, kterými lze z daného stavu dosáhnout cílového stavu).

Blokové schéma ES:



Základní charakteristikou každého ES je oddělení znalostí a dat od mechanismu jejich využívání. Prázdným (shell) ES se pak nazývá systém s prázdnými bázemi znalostí i dat, tedy systém, který lze po naplnění těchto bází použít k řešení úloh z různých problémových oblastí.

Do báze znalostí se ukládají expertní znalosti, které ES musí být schopen získat od expertů prostřednictvím modulu získávání znalostí. Při získávání znalosti od pomáhá expertům znalostní inženýr, který musí být dobře seznámen jak s příslušným ES, tak i s řešenou problematikou.

Do báze dat (pracovní paměti) ES se ukládají znalosti jedinečné, které se týkají pouze konkrétních řešených úloh. Tato data poskytuje systému uživatel nebo různé měřící přístroje (přes uživatelské rozhraní) a v průběhu výpočtu jsou tyto znalosti doplňovány fakty/výroky, resp. hypotézami odvozenými inferenčním mechanismem.

Důležitou částí ES je vysvětlovací modul, který musí být schopen vysvětlit uživateli postup při řešení dané úlohy. Právě tento modul odlišuje ES od jiných znalostních systémů.

Znalosti, které se ukládají do bází ES se rozdělují na:

- deklarativní (poznatky) - co je, resp. co má být poznáno/odvozeno,
- procedurální (pravidla) - jak se bude poznávat/odvozovat.

Deklarativní znalosti se nacházejí obvykle v bázi dat, zatímco procedurální znalosti v bázi znalostí.

ES jejichž báze znalostí jsou tvořeny pravidly se nazývají pravidlové ES (Rule-based expert systems).

Princip činnosti pravidlových ES vychází z principu činnosti produkčního systému (Newell and Simon, 1972), který má tři základní části:

- bázi znalostí (pravidel)
- bázi dat (pracovní paměť)
- inferenční mechanismus

Pravidla se zapisují obvykle ve tvaru P $\rightarrow$ A (if P then A)
a interpretují se takto:

jestliže je splněna podmínka P pak je možné vykonat akci A

resp.

jestliže je splněna podmínka P pak je možné odvodit A

Pozn.: produkční systém nemá vysvětlovací modul, a proto poskytuje závěry bez možného ověření postupu odvozování.

Během činnosti produkčního systému se často v jeho bázi znalostí nachází více aktuálně aplikovatelných pravidel. Tato pravidla představují tzv. konfliktní množinu, ze které se pak k aplikaci vybere jedno pravidlo, a to:

- podle pořadí v bázi znalostí,
- podle priority,
- podle časových známek objektů v podmínkové části,
- podle počtu objektů v podmínkové části,
- náhodně.

Obvykle se pro stejnou množinu dat aplikuje pravidlo pouze jednou.

Inferenční mechanismus může pracovat dvěma směry:

- dopředným, řízeným daty (forward chaining / data driven),
- zpětným, řízeným cílem (backward chaining / goal driven);
  v tomto případě používá produkční systém ještě zásobník, do kterého ukládá dosud nesplněné (pod)cíle.

Příklad – odvozování řízené daty (data driven / forward chaining)

Báze znalostí:

1. $y \wedge d \rightarrow z$
2. $x \wedge b \wedge c \rightarrow y$
3. $a \rightarrow x$
4. $e \rightarrow u$
5. $f \wedge u \rightarrow v$

Nechť báze dat obsahuje fakty *a*, *b*, *c*, *d*, *e* a nechť cílem odvozování (inference) je *z*:

Iterace	báze dat (pracovní paměť)	konfliktní množina	vybrané pravidlo
0	*a b c d e*	3, 4	3
1	*a b c d e x*	4, 2	4
2	*a b c d e x u*	2	2
3	*a b c d e x u y*	1	1
4	*a b c d e x u y z*		

Příklad – odvozování řízené cílem (goal driven / backward chaining)

Báze znalostí:

1. $y \wedge d \rightarrow z$
2. $x \wedge b \wedge c \rightarrow y$
3. $a \rightarrow x$
4. $e \rightarrow u$
5. $f \wedge u \rightarrow v$

Předpokládejme stejnou počáteční bázi dat i stejný cíl:

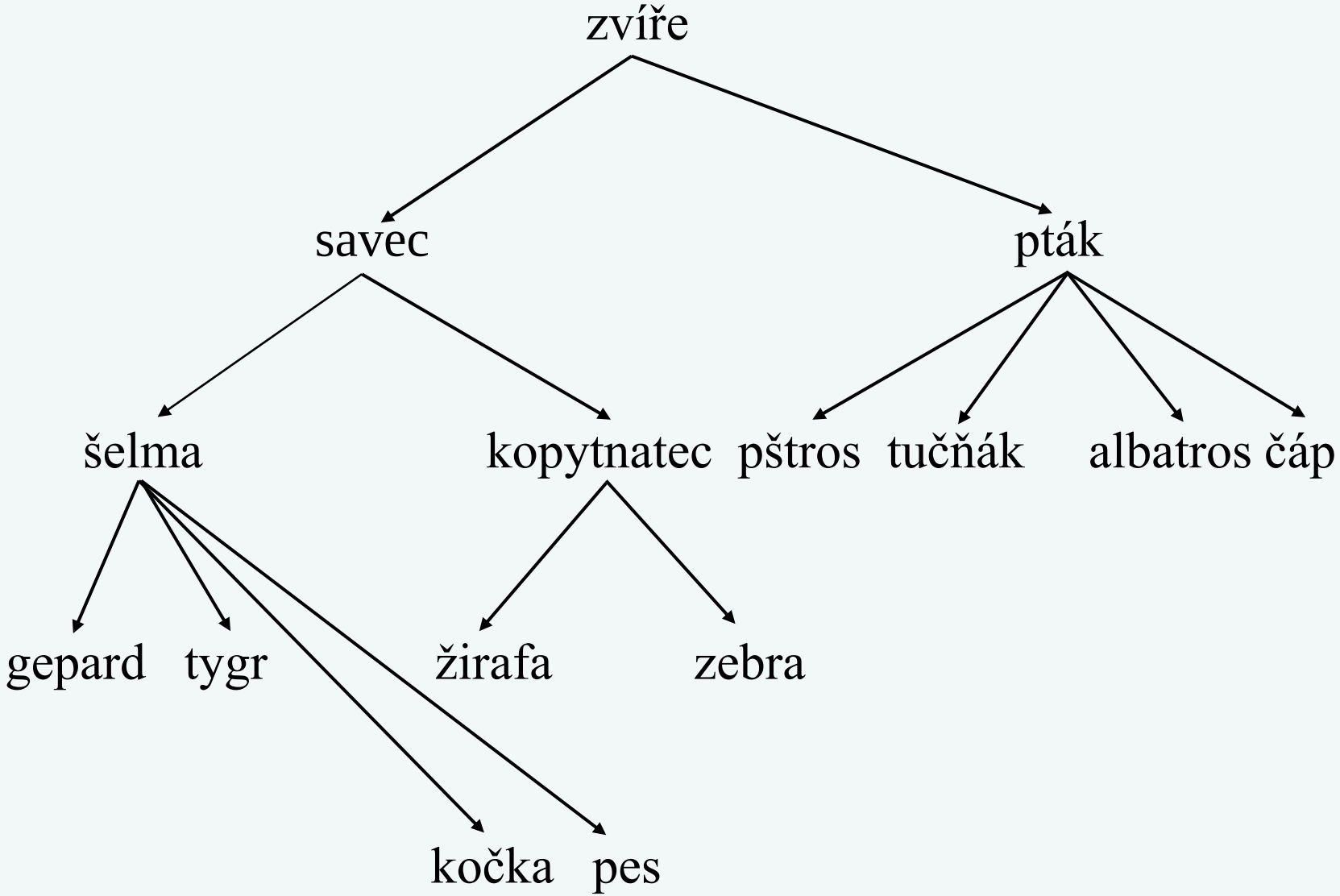
Iterace	báze dat (pracovní paměť)	konfliktní množina	vybrané pravidlo	zásobník (pod)cílů
0	*a b c d e*			*z*
				z y
		3	3	*z y x*
1	*a b c d e x*	2	2	*z y*
2	*a b c d e x y*	1	1	*z*
3	*a b c d e x y z*			

Činnost produkčního sytému končí buď odvozením cíle (úspěchem), nebo prázdnou konfliktní množinou aplikovatelných pravidel (neúspěchem).

Všimněte si, že v předchozím příkladu byl při dopředném řetězení odvozován fakt *u*, který na odvození cíle *z* neměl žádný vliv. Je zřejmé, že v rozsáhlých bázích dat by pak tímto inferenčním postupem bylo zbytečně odvozováno mnoho podobných faktů.

Proto se dopředné řetězení prakticky používá v případech, kdy cíle odvozování nejsou známé (například při analýze, nebo interpretaci) a v ostatních případech, především pak v diagnostice, se používá výhradně řetězení zpětné.

Příklad: Jednoduchý pravidlový ES pro rozpoznávání zvířat:



Příklad – pokračování: ES v jazyku PROLOG:

/* Báze znalostí */

```
rule(1,zvire,savec,[c1]).          % if c1 or c2
rule(2,zvire,savec,[c2]).          % then zvíře je savec
rule(3,zvire,ptak,[c3]).
rule(4,zvire,ptak,[c4,c5]).        % if c4 and c5 then zvíře je pták
rule(5,savec,selma,[c6]).          % if c6 or (c7 and c8 and c9)
rule(6,savec,selma,[c7,c8,c9]).    % then zvíře je šelma
…
rule(9,selma,gepard,[c12,c13]).
rule(10,selma,tygr,[c12,c14]).
…
rule(19,selma,pes,[c22,c24]).
rule(20,selma,pes,[c22,c26]).
…
```

/* Otázky uživatelům při tvorbě báze dat/faktů */

```
ask(c1):- write('Ma srst ? ').
ask(c2):- write('Dava mleko ? ').
ask(c3):- write('Ma peri ? ').
ask(c4):- write('Leta ? ').
ask(c5):- write('Klade vejce ? ').
ask(c6):- write('Zivi se masem ? ').
ask(c7):- write('Ma spicate zuby ? ').
ask(c8):- write('Ma drapy ? ').
…
ask(c24):-write('Steka ? ').
ask(c25):-write('Mnouka ? ').
ask(c26):-write('Chodi na voditku ? ').
```

```
/* Inferenční mechanismus (na úloze nezávislý) */

e:-  retrfakt,
     write('Expertni system pro rozpoznavani zvirat. '), nl,
     write('Na otazky odpovidejte ano/ne: '), nl, nl,
     recognition(zvire,0).

retrfakt:- retract(fakt(_,_)), fail.        % Vyprázdnění báze dat
retrfakt.

recognition(X,I):- rule(N,X,Y,Z), conditions(Z), conclusion(X,Y,N),
                   J is I + 1, recognition(Y,J).
recognition(_,I):-  I > 0, write('Vic nevim.'), nl.
recognition(_,_):-  write('Takove zvire neznam.'), nl.
```

```
/* Inferenční mechanismus - pokračování */

conditions([]).
conditions([X|Y]):- condition(X),conditions(Y).

condition(X):- fakt(X,ano), !.
condition(X):- fakt(X,_), !, fail.
condition(X):- ask(X), read(Y), assertz(fakt(X,Y)), Y=ano.

conclusion(X,Y,N):- write('Je to '), write(X), write('-'), write(Y),
                    write(' podle pravidla '), write(N), write('.'),
                    nl, nl.
```

Pravidlové expertní systémy pracující s neurčitostí

Zásadní problém, se kterým se lze setkat při expertíze pomocí ES, spočívá v nedostatku exaktních znalostí, které by byly potřebné pro odvozování naprosto spolehlivých závěrů.

Zmíněný nedostatek (neurčitost/neúplnost/nejasnost) bývá způsoben nejčastěji těmito příčinami:

- neznámými daty,
- nepřesně definovanými pravidly,
- nepřesně vyjadřovanými závěry expertů („často“, „někdy“ atd.),
- neshodami/odlišnými názory expertů na různá pravidla.

Existuje několik různých přístupů k práci s neurčitostí v ES. Pro získání základní představy se dále zaměříme na jediný z nich, a to na přístup který spočívá v práci s tzv. faktorem jistoty.

Podmínkovou část pravidla budeme dále označovat jako předpoklad *E* (Evidence), odvozovaný závěr jako hypotézu *H* (Hypothesis) a důvěru ve správnost pravidla (tj. důvěru v platnost hypotézy *H*) jako faktor jistoty cf_r (Certainty Factor):

$$\text{if } E \text{ then } H \ \{cf_r\}$$

Faktor jistoty je formálně definován vztahem

$$cf_r = \frac{MB(H,E) - MD(H,E)}{1 - \min(MB(H,E), MD(H,E))}$$

kde $MB(H,E)$ značí tzv. míru důvěry (Measure of Belief) a $MD(H,E)$ míru nedůvěry (Measure of Distrust) v hypotézu *H*, pro jistý, tj. pravdivý, předpoklad *E*. Pro takový předpoklad *E* tedy platí

$$cf(H, E) = cf_r$$

Pro míry důvěry a nedůvěry pak platí vztahy

$$MB(H,E)=\begin{cases}1 & pro \quad p(H)=1\\ \dfrac{\max(p(H|E),p(H))-p(H)}{1-p(H)} & jinak\end{cases}$$

$$MD(H,E)=\begin{cases}1 & pro \quad p(H)=0\\ \dfrac{\min(p(H|E),p(H))-p(H)}{-p(H)} & jinak\end{cases}$$

kde $p(H)$, resp. $p(H|E)$ značí nepodmíněnou, resp. podmíněnou pravděpodobnost hypotézy H.

Pro míru důvěry MB, míru nedůvěry MD a faktor jistoty cf_r platí:

Rozsah	$0 \leq MB \leq 1$ $0 \leq MD \leq 1$ $-1 \leq cf_r \leq 1$
Jistě pravdivá hypotéza $p(H\|E) = 1$	$MB = 1$ $MD = 0$ $cf_r = 1$
Jistě nepravdivá hypotéza $p(H\|E) = 0$ $p(\neg H\|E) = 1$	$MB = 0$ $MD = 1$ $cf_r = -1$
Neznámý předpoklad $p(H\|E) = p(H)$	$MB = 0$ $MD = 0$ $cf_r = 0$

Faktor jistoty ve slovním vyjádření pak může znamenat například:

cf	Slovní význam
-1	jistě/určitě ne
-0.8	téměř jistě ne
-0.6	pravděpodobně ne
-0.4	možná ne
-0.2 až 0.2	nevím
0.4	možná
0.6	pravděpodobně
0.8	téměř jistě
1	jistě/určitě

Hodnoty faktorů jistoty se pomocí předchozích vztahů obvykle nepočítají. Buď jsou stanovovány experty ad hoc, nebo jsou v pravidlech if E then H $\{cf_r\}$ s nejistými předpoklady E, tj. s $\{cf(E)\}$, vyhodnocovány pomocí následujících jednoduchých operací:

- Pro negaci nejistého předpokladu

$$cf(\neg E) = -cf(E)$$

- Pro nejistý předpoklad E

$$cf(H, E) = cf_r \cdot cf(E)$$

- Pro konjunkci nejistých předpokladů E_1, E_2, … v podmínkové části pravidla:

$$cf(H, E_1 \wedge E_2 \wedge \ldots) = cf_r \cdot \min(cf(E_1), cf(E_2), \ldots)$$

- Pro disjunkci nejistých předpokladů E_1, E_2, … v podmínkové části pravidla:

$$cf(H, E_1 \vee E_2 \vee \ldots) = cf_r \cdot \max(cf(E_1), cf(E_2), \ldots)$$

- Pro dvě pravidla se stejnou hypotézou:

if E_1 then H $\{cf_{r1}\}$ $\quad cf_1 = cf_{r1} \cdot cf(E_1)$
if E_2 then H $\{cf_{r2}\}$ $\quad cf_2 = cf_{r2} \cdot cf(E_2)$

$$cf_{r1r2}\left(H, E_{1,} E_2\right) = \begin{cases} cf_1 + cf_2 \cdot \left(1 - cf_1\right) & \text{if } cf_1 > 0 \text{ and } cf_2 > 0 \\ \dfrac{cf_1 + cf_2}{1 - \min\left(|cf_1|, |cf_2|\right)} & \text{if } \left(cf_1 \cdot cf_2\right) < 0 \\ cf_1 + cf_2 \cdot \left(1 + cf_1\right) & \text{if } cf_1 < 0 \text{ and } cf_2 < 0 \end{cases}$$

- Pro více pravidel se stejnou hypotézou se postupuje podobně: nejprve se určí cf_{r1r2} pro dvě pravidla, pak se spočítá cf_{r1r2r3} z cf_{r1r2} a z cf_{r3}, atd.

Příklad (tři pravidla se stejnou hypotézou):

if $(E_1 \wedge E_2 \wedge E_3) \vee (E_4 \wedge \neg E_5)$ then H $\{cf_{r1}\}$
if $E_6 \wedge E_7$ then H $\{cf_{r2}\}$
if E_8 then H $\{cf_{r3}\}$

Nechť $cf(E_1) = 0.9$, $cf(E_2) = 0.8$, $cf(E_3) = 0.3$, $cf(E_4) = -0.5$,
$cf(E_5) = -0.4$, $cf(E_6) = 0.7$, $cf(E_7) = -0.1$, $cf(E_8) = 0.6$,
$cf_{r1} = 0.9$, $cf_{r2} = 0.7$, $cf_{r3} = 0.8$

Pak $cf(H_{\text{rule1}}) = 0{,}9 \cdot \max(\min(0.9,0.8,0.3), \min(-0.5,0.4)) = 0.27$
$cf(H_{\text{rule2}}) = 0.7 \cdot \min(0.7,-0.1) = -0.07$
$cf(H_{\text{rule3}}) = 0.8 \cdot 0.6 = 0.48$
$\text{cf}(H_{\text{r1+r2}}) = (0.27 + (-0.07)) / (1- \min(0.27,0.07)) = 0.215$
$cf(H_{\text{(r1+r2)+r3}}) = 0.215 + 0.48 \cdot (1- 0.215) = 0.59$

Z praktických účelů je vhodné upravit pravidla tak, aby jejich podmínkové časti byly tvořeny pouze konjunkcemi, nebo disjunkcemi předpokladů (jde o podobnou úpravu, jako byla úprava obecných grafů na AND/OR grafy ve třetí přednášce). Například první pravidlo v předcházejícím příkladu lze upravit pomocí nových pravidel takto:

if E_5 then E_{5n} $\{cf_{r5n} = -1\}$
if $E_4 \wedge E_{5n}$ then E_{45n} $\{cf_{r45n} = 1\}$
if $E_1 \wedge E_2 \wedge E_3$ then E_{123} $\{cf_{r123} = 1\}$
if $E_{123} \vee E_{45n}$ then H $\{cf_{r1}\}$

tj.

$cf(E_{5n}) = cf_{r5n} \cdot cf(E_5) = -1 \cdot (-0.4) = 0.4$
$cf(E_{45n}) = cf_{r45n} \cdot \min(cf(E_4), cf(E_{5n})) = 1 \cdot \min(-0.5, 0.4) = -0.5$
$cf(E_{123}) = cf_{r123} \cdot \min(cf(E_1), cf(E_2), cf(E_3)) =$
$= 1 \cdot \min(0.9, 0.8, 0.3) = 0.3$
$cf(H, E_{123}, E_{45n}) = cf_{r1} \cdot \max(cf(E_{123}), cf(E_{45n})) =$
$= 0.9 \cdot \max(-0.5, 0.3) = 0.27$

Bázi znalostí ES pracujícího s neurčitostí pak lze zobrazit jako AND/OR graf, který má několik kořenových hypotéz a celou řadu vnitřních a listových hypotéz/předpokladů. Předpoklady, resp. hypotézy ohodnocené explicitně uživatelem se stávají fakty.

Práce ES pracujícího s neurčitostí probíhá obvykle takto:

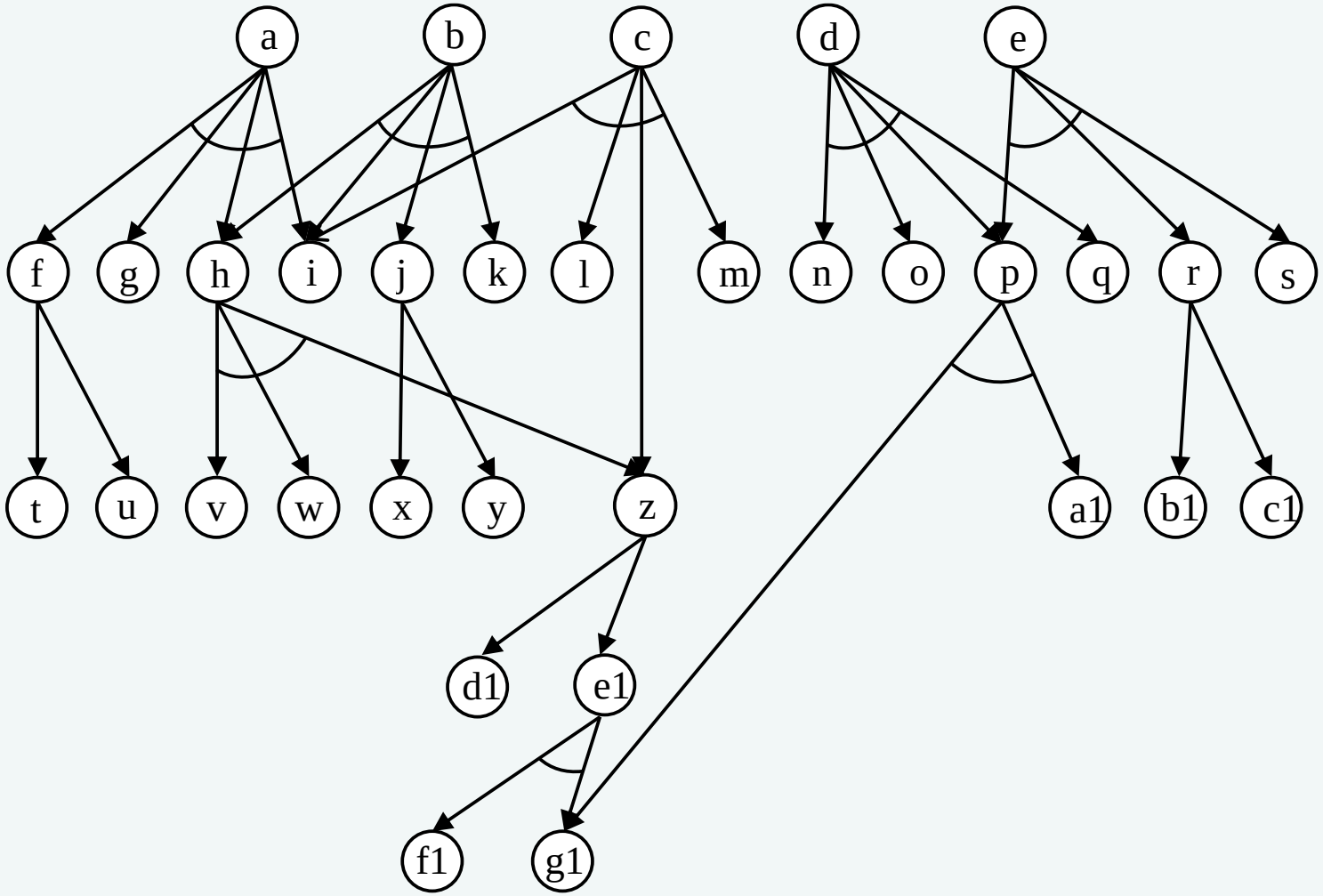
1. Uživatel oznámí systému fakt, který musí odpovídat některému z uzlů AND/OR (tj. některému z předpokladů některého pravidla).
2. Systém hledá kořenové hypotézy, které zadaný fakt ovlivňuje.
3. Systém oznámí uživateli zjištěné kořenové hypotézy a jejich aktuální platnosti (faktory jistoty *cf*) a nastaví práh ceny verifikace (ověřování) hypotéz na zvolenou minimální hodnotu.
4. Je-li uživatel spokojen, činnost systému končí. Jinak systém zvýší práh pro ceny verifikace a prohledává graf pro každý relevantní kořen (hypotézu) směrem k listům, dokud nenarazí na uzel který:

- odpovídá faktu zodpovězenému dříve uživatelem,
- představuje hypotézu s cenou verifikace vyšší, než je aktuální práh - pak systém použije buď apriorní platnost této hypotézy (pokud je zadána), nebo platnost průměrnou, tj. $cf = 0$.
- představuje hypotézu s cenou verifikace nižší nebo stejnou, než je aktuální práh - pak se prohledávání grafu rekurzivně zanořuje a vyčíslí se tak *cf* této hypotézy.
- je dosud neznámým listovým předpokladem. Systém se dotáže na jeho platnost a tím se z tohoto předpokladu stane fakt.

5. Systém postupuje zpět ke kořenům a přepočítává platnosti všech vnitřních a kořenových hypotéz.
6. Systém oznámí uživateli nové platnosti relevantních kořenových hypotéz a vrací se k bodu 4.

Příklad - diagnostika místa krvácení z trávící trubice

(převzato z Brůha, I., Jelínek, J.: Expertní konzultační systémy, Praha, 1984):



Význam jednotlivých hypotéz/faktů:

a	Pacient má krvácení žaludečního vředu
b	Pacient má krvácení duodenálního (dvanáctníkového) vředu
c	Pacient má jícnové varixy (křečové žíly jícnu)
d	Pacient má hemeroidy
e	Pacient má vředovitý zánět v tlustém střevu
f	Vložený uzel, bez slovního popisu
g	Pacient má bolesti při jídle, které po vyprázdnění žaludku mizí
h	Pacient má krvácení vředu
i	Pacient má pozitivní vyšetření RTG s kontrastní látkou
j	Vložený uzel, bez slovního popisu
k	Pacient má hyperaciditu (zvýšenou kyselost žaludeční šťávy)
l	Pacient má ztížené polykání
m	Pacient má pozitivní endoskopické vyšetření
n	Pacient má bolesti při defekaci
o	Pacient má svědění konečníku
p	Pacient má krvácení z dolní části trávící trubice
q	Pacient má identifikovány hemeroidy při rektoskopickém vyšetření

r	Pacient má potíže se stolicí
s	Pacient má příměs hlenu ve stolici
t	Pacient má hypoaciditu (nedostatek žaludeční šťávy)
u	Pacient má anaciditu (sníženou kyselost žaludeční šťávy)
v	Pacient má sezónní bolesti v oblasti žaludku
w	Pacient má pálení žáhy
x	Pacient má bolesti 2-3 hodiny po jídle
y	Pacient má noční bolesti
z	Pacient má krvácení z horní části trávící trubice
a1	Krev je čerstvá
b1	Pacient má nucení na stolici
c1	Pacient má průjem
d1	Pacient zvrací krev
e1	Vložený uzel, bez slovního popisu
f1	Krev je černá
g1	Pacient má krev ve stolici

Báze znalostí/pravidel:

if $(f \wedge g \wedge h \wedge i)$ then a {0.9}

if $(h \wedge i \wedge j \wedge k)$ then b {0.9}

if $(i \wedge l \wedge m \wedge z)$ then c {0.8}

if $(n \wedge o \wedge p \wedge q)$ then d {0.9}

if $(p \wedge r \wedge s)$ then e {0.7}

if $(t \vee u)$ then f {0.85}

if $(v \wedge w \wedge z)$ then h {0.9}

if $(x \vee y)$ then j {0.75}

if $(a1 \wedge g1)$ then p {0.9}

if $(b1 \vee c1)$ then r {0.8}

if $(d1 \vee e1)$ then z {0.8}

if $(f1 \wedge g1)$ then $e1$ {0.9}

Příklad – pokračování: ES v jazyku PROLOG:

/* Báze znalostí */

```
rule(and,[f,g,h,i],a,0.9).
rule(and,[h,i,j,k],b,0.9).
…
rule(or,[t,u],f,0.85).
rule(and,[v,w,z],h,0.7).
rule(or,[x,y],j,0.75).
…
apriori(a,0.4).
apriori(b,-0.2).
…
price(i,50).
price(m,30).
…
```

/* Inferenční mechanismus (na úloze nezávislý) */

```
expert(Hyp,CF):- retr_all, asserta(fakt(Hyp,CF)),
                 find_roots(Hyp,Roots),
                 asserta(threshold(10)),
                 repeat,
                 calculate(Roots,Roots),
                 test_end.

retr_all:- retrfakt, retrfound, retrnotroot.

retrfakt:- retract(fakt(_,_)), fail.
retrfakt.

retrfound:- retract(found(_)), fail.
retrfound.

retrnotroot:- retract(not_root(_)), fail.
retrnotroot.
```

```
find_roots(Hyp,_):- search_up(Hyp,UpperHyp),
                    assert_1(found(UpperHyp)), fail.
find_roots(_,L):- collect([],L), !.

search_up(Hyp,UH):- rule(_,List,H,_), member(Hyp,List),
                    assert_1(not_root(Hyp)), search_up(H,UH).
search_up(Hyp,Hyp).

assert_1(X):- X, !.
assert_1(X):- asserta(X).

collect(Tmp,L):- retract(found(X)), !, search_root(Tmp,X,L).
collect(L,L).

search_root(Tmp,X,L):- not_root(X), retract(not_root(X)), collect(Tmp,L).
search_root(Tmp,X,L):- collect([X|Tmp],L).
```

```
calculate([],_).
calculate([H|T],Roots):- calc(H,Roots,_), calculate(T,Roots).

calc(H,Roots,CF_H):- rule(and,L,H,CF_R), calc_and(L,1,CF_E,Roots),
                     CF_H is CF_E * CF_R,
                     write_calc(H,CF_H,Roots).
calc(H,Roots,CF_H):- rule(or,L,H,CF_R), calc_or(L,-1,CF_E,Roots),
                    CF_H is CF_E * CF_R,
                     write_calc(H,CF_H,Roots).
calc(H,Roots,CF_H):- rule(comb,L,H,CF_R), calc_comb(L,CF_E),
                    CF_H is CF_E * CF,
                     write_calc(H,CF_H,Roots).

write_calc(H,CF_H,Roots):- member(H,Roots),!,
                           write('The root hypothesis '), write(H),
                           write(' has the validity '), write(CF_H), nl,nl.
write_calc(H,CF_H,_):- write('The inner hypothesis '), write(H),
                       write(' has the validity '), write(CF_H), nl.
```

```
calc_and([F|T],CF_in,CF_out,Roots):- test(F,CF_F,Roots),
                                     min(CF_in,CF_F,CF_tmp),
                                     calc_and(T,CF_tmp,CF_out,Roots).
calc_and([],CF,CF,_).

calc_or([F|T],CF_in,CF_out,Roots):- test(F,CF_F,Roots),
                                    max(CF_in,CF_F,CF_tmp),
                                    calc_or(T,CF_tmp,CF_out,Roots).
calc_or([],CF,CF,_).

calc_comb([H1,H2],CF_out,Roots):- test(H1,CF1,Roots),
                                  test(H2,CF2,Roots),
                                  calc_v(CF1,CF2,CF_out).
calc_comb([H1,H2|T],CF_out,Roots):- test(H1,CF1,Roots),
                                    calc_comb([H2|T],CF2,Roots),
                                    calc_v(CF1,CF2,CF_out).
```

```
calc_v(CF1,CF2,CF):- CF1 >= 0, CF2 >= 0,
                     CF is CF1 + CF2 * (1 - CF1).
calc_v(CF1,CF2,CF):- (CF1 * CF2) < 0, abs(CF1,C1), abs(CF2,C2),
                     min(C1,C2,C), CF is (CF1 + CF2) / (1 - C).
calc_v(CF1,CF2,CF):- CF1 < 0, CF2 < 0,
                     CF is CF1 + CF2 * (1 + CF1).

test(Hyp,CF,_):- fakt(Hyp,CF), !.
test(Hyp,CF,Roots):- threshold(T), price(Hyp,P), P>T, !, val(Hyp,CF),
                     write('The hypothesis '), write(Hyp),
                     write(' has the default value '), write(CF),
                     write(' and price of its verification is '), write(P), nl.
test(Hyp,CF,_):- calc(Hyp,CF), !.
test(Hyp,CF,_):- write('Type the validity of  '),write(Hyp),
                 write(':  '), read(CF), asserta(fakt(Hyp,CF)), !.

val(Hyp,CF):- apriori(Hyp,CF), !.
val(_,0).
```

```
min(X,Y,X):- X<Y, !.
min(_,Y,Y).

max(X,Y,X):- X>Y,!.
max(_,Y,Y).

abs(X,X):- X >= 0.
abs(X,Y):- X < 0, Y is -X.

test_end:- retract(threshold(T)),
           write('Price threshold was '),
           write(T),nl,
           nl,write('Are you satisfied (y/n): '),
           read(X),test_e(X,T).

test_e(y,_):- !.
test_e(_,T):- T>=100, !, write('No other solution.'), nl.
test_e(_,T):- Tnew is T+10, asserta(threshold(Tnew)),fail.
```