Dörfler Viktor[i] - Velencei Jolán[ii]

Tudásrendezés

A döntéstámogatás fogalom alatt sokan döntéshelyettesítést szeretnének érteni. Amíg döntéstámogatásról beszélünk, és közben döntéshelyettesítésre gondolunk, gondjaink lesznek. Az igazi döntéshozó azért szökik, mert nem akarja, hogy egy szoftver lecserélje, a hamisnak meg azért nincs helye, mert éppen ezt akarja.

A döntéstámogatás lejáratott fogalmát tudásrendezésre cseréljük. A szakértő tudásának rendezése egy folyamat, melyet nem egy számítógépes program hajt végre, hanem egy ember, a tudásrendező. Ehhez használ egy programot, amit keretrendszernek hívunk. A nevét onnan kapta, hogy a tudásrendezés folyamatának kereteit adja meg, nem pedig elvégzi a tudásrendezést. A Doctus tudásbázisú keretrendszer (shell) alkalmazási lehetőségeit mutatjuk be a cikkben.

Hallgatólagos

„Az emberiség satnya testi adottságai ellenére életképesnek bizonyult. Aligha engedhette meg magának, hogy túl gyakran hozzon téves döntéseket, mert azt a természetes szelekció hamar megbüntette volna" (Mérő, 1997). Tehát, az ember tud dönteni**. Mi mindent tud egy döntéshozó?** Gazdag ismerethátterrel rendelkezünk az emberi tudásról. Azt is tudjuk, hogy a tudás sokféle lehet. Ebben a cikkben csak a szakértő tudásával foglalkozunk. Ez az a tudásszint, ahol még szavakba lehet önteni a (szak)tudást. Biztos, hogy érdekes az alacsonyabb tudásszinten lévő emberek - a kezdő és a haladó - tudása is. Őket mégsem vizsgáljuk. Feltételezzük, hogy az ő szerepük a vállalati döntések előkészítésében elenyésző, tehát **még nem** szükséges modellezni. Szintén érdekes lehet a nagymester tudásának vizsgálata. Ezt a tudást viszont **már nem** tudjuk modellezni. Minél többet tud valaki, annál nagyobb az a tudás, amit nem képes szavakba önteni.

A szakértő tudása az üzleti életben mindig a döntéshozót (menedzsert és/vagy vezetőt) támogatja. A döntéshozónak mindig a **„Tudod-e...?"** kérdést kell feltennie a szakértőnek. Ez így volt a számítógépek megjelenése előtt is. Amikor megjelentek a számítógépek, a döntéshozó megváltoztatta kérdését: **„Mi mindent tudsz?"**. Erre természetesen a gép rengeteg adatot kínált válaszul, ami haszontalan volt a döntéshozónak. Ma már eljutottunk oda,

[i] Egyetemi tanársegéd, Budapesti Műszaki Egyetem, Ipari Menedzsment és Vállalkozásgazdaságtan Tanszék
[ii] Tudományos munkatárs, Budapesti Műszaki Egyetem, Ipari Menedzsment és Vállalkozásgazdaságtan Tanszék

hogy ismét fel lehet tenni a **„Tudod-e...?”** kérdést. Ehhez persze a gépet meg kell tanítani a szakértő tudására.

A szakértő bízik tapasztalatában. Csak egy gondja van. Hogyan tudja azt előhívni és kimondani. Amit senki sem tud megmagyarázni, azt „jó szimatnak” nevezzük (Baracskai, 1999). Gyakran döntünk szimatunk alapján, mert az azt sugallja, hogy a döntés helyes lesz. Nehéz megindokolni, milyen gondolkodási folyamat szüleménye az ítélet, csak azt tudjuk, hogy megszületik a döntés, és gyakran kielégíti elvárásainkat. A tapasztalt szakértő inkább a megérzéseire támaszkodik, mint a kemény adatokra. A tapasztalat sosem lehet teljes mértékben kifejezhető. A tapasztalatnak van szavakkal leírható (explicit) és van szavakkal nem kifejezhető, hallgatólagos (látens) része (Polányi, 1997). Nem lehet elmagyarázni valakinek, hogy hogyan kell verset írni, vagy megtalálni a fát, amiben Pinocchio lakik. De azt sem, hogy mi alapján dönti el a vezető, hogy melyik fejlesztési javaslatot fogadja el, és hogy kit nevez ki munkatársának.

De előhívható

A szakértő tudását **döntési szempontokkal** és a közöttük lévő **kapcsolatok**kal írjuk le. Ezek a szakértő hosszú távú memóriájában vannak. Ha explicit döntési szempontok között explicit kapcsolatok vannak, az többféleképpen leírható. Az egyetlen probléma, hogy melyik leírást választjuk, de ezzel most nem foglalkozunk. Azt mondjuk, hogy itt nincs probléma. A látens szempontok közötti explicit kapcsolatról értelmetlen volna beszélni, úgyhogy ezzel sincs semmi probléma. Két problémát látunk ezen a területen:

- Nem ismerjük az explicit szempontok közötti látens kapcsolatokat.
- Nem ismerjük a látens szempontok közötti látens kapcsolatokat.

Itt csak az első problémával foglalkozunk. A Doctus keretrendszer segítségével bemutatjuk a tudásbázis felépítésének folyamatát. Ennek során megtaláljuk az explicit szempontok közötti látens kapcsolatok **egy leírását** a sok közül. Részt vesznek benne a tudásrendező és a szakértők. Szakértő az, akinek egy diszciplínában néhány ezer kognitív sémája van. A tudásrendező feladata, hogy kihúzza a szakértőkből a tudást, előhívja tapasztalatukat, és segítsen azt rendszerezni.

A legegyszerűbben megfogalmazva a kérdés a következő**: Hogyan kapcsolódnak össze a tudás elemei egy szakértő fejében egy adott pillanatban?** Mi erre a kérdésre a szimbolikus logika alkalmazásával adunk választ. A döntési szempontokat ***tulajdonságok***kal fejezzük ki. A tulajdonsághoz tartoznak az ***értékek***, amiket felvehet. A tulajdonságok közötti kapcsolatokat **„ha...akkor" szabályok**kal írjuk le. A Doctus keretrendszer a tudás kétféle modellezésére alkalmas. Az **eset alapú következtetés**t **(Case Based Reasoning, CBR)** most nem mutatjuk be, a cikkben a **szabály alapú következtetés**t használjuk **(Rule Based Reasoning, RBR)**.

Amikor a szabályok új tudást szülnek

A tudásbázis felépítése két részből áll: a tudásgyűjtésből (Knowledge Acquisition) és a tudásrendezésből (Knowledge Engineering). A tudásgyűjtés a szakértő tudásának kihúzása, szavakba öntése, a tudásrendezés ennek rendszerezése, formába öntése. A tudásbázis kialakításának első lépése mindenképpen tudásgyűjtés, az utolsó mindenképpen tudásrendezés, de menet közben a kettő összefonódik. Először a tudásrendező szóra bírja a szakértőt, akinek ki kell mondania azokat a döntési szempontokat, tulajdonságokat (Attributes), amiket fontosnak tart, és azokat az értékeket (Values), amiket ezek felvehetnek. Ki kell mondania a tulajdonságok közötti függőségi viszonyokat. Tudásrendezői tapasztalatunk azt mutatja, hogy a szakértő 3-4 tulajdonságnál többet nem képes összevonni „ha...akkor" szabályokkal. A keretrendszer lehetőséget ad a tulajdonságok hierarchikus rendezésére. Az így kapott gráfot ***deduktív gráf***nak (Deductive Graph) nevezzük (1. ábra). Bármelyik tulajdonságra megadhatjuk, hogy melyik másik tulajdonságoktól függjön. Egy tulajdonságtól függhet több másik is. Ha A tulajdonság függ B-től, akkor B nem függhet A-tól, áttételesen sem. Ezt a Doctus keretrendszer nem is engedi meg. Fontos, hogy a szakértő a számítógép előtt ülve beszélgessen a tudásrendezővel (és a számítógéppel is). Ha a tudásrendező képes érzelmileg ráhangolódni a szakértőre, akkor ő könnyedén tudja a megadott formába önteni tudását. A tudásrendező kérdéseket tesz fel a szakértőnek. Ezt nem viheti túlzásba, mert a szakértő ráunhat, és a határidőket is be kell tartani. A kérdésekkel segíti a használt fogalmak tisztázását.

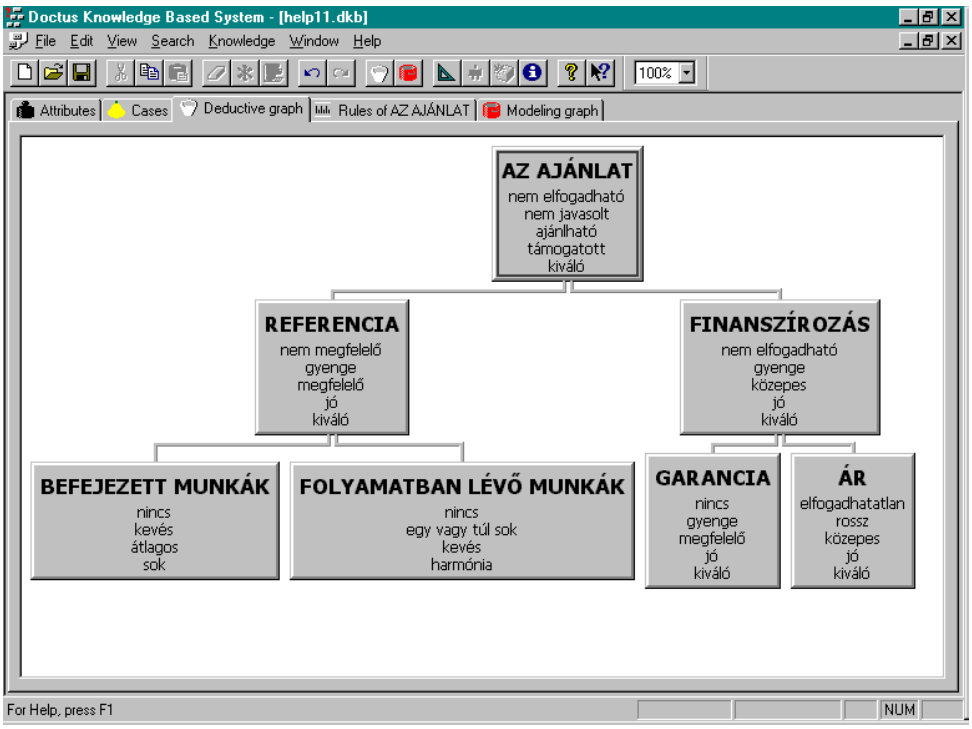
> *Az egyik kutatónak a sikerességet az elfogadott innováció jelentette, a másiknak a legalább négypontos publikáció, a harmadiknak a magas prémium. Csoda, ha nem tudtak megegyezni?*

A függőségi viszonyok alapján a tulajdonságokat három csoportba oszthatjuk (1. ábra):

- Lesz egy, amelyiktől nem függ semmi, egy olyan tulajdonság, amelyik az összes többi függvénye, ezt nevezzük ***végkövetkeztetés***nek, vagy ***kimeneti tulajdonság***nak. (Output Attribute)

- Lesznek olyanok, amik semmitől sem függnek, ezek a ***független tulajdonság***ok, vagy a ***bemeneti tulajdonság***ok. (Input Attributes)

- Lesznek olyan tulajdonságok, amik függenek valamilyen más tulajdonságoktól (ezek kimenetei), és egyúttal tőlük is függnek más tulajdonságok (ezek bemenetei). Ezeket fogjuk ***függő tulajdonságok***nak, vagy ***közbülső tulajdonság***oknak nevezni. (Depending Attributes)

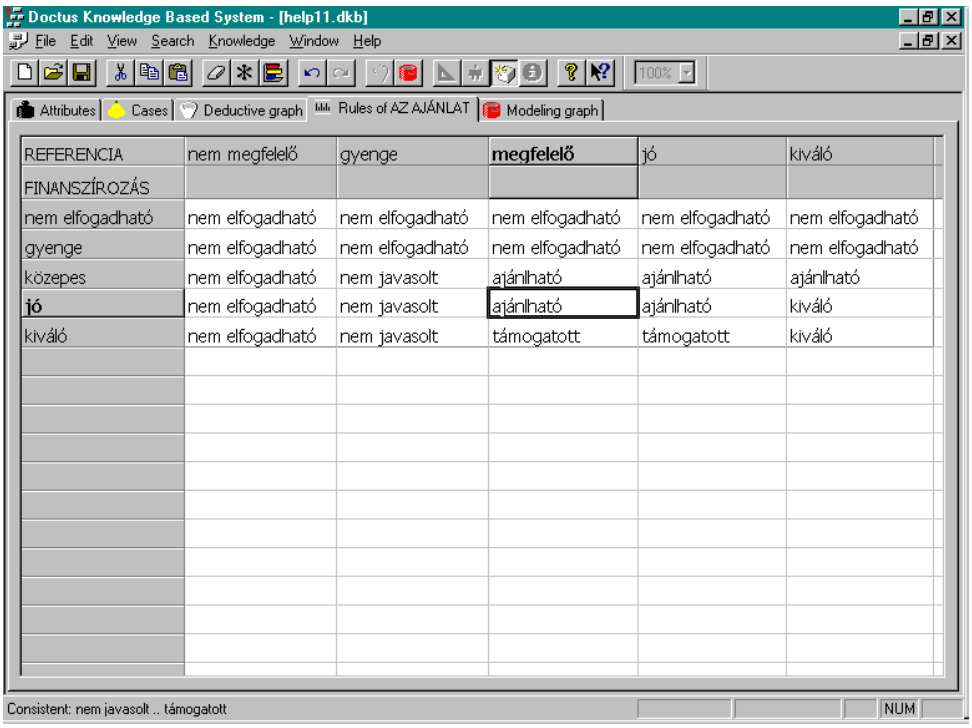


1. ábra: Bemeneti, függő és kimeneti tulajdonságok a deduktív gráfban

Azt már tudjuk, hogy melyik tulajdonság melyektől függ, csak azt nem, hogy hogyan. A függőségi viszonyok „hogyan”-ját ***szabályok***kal (Rules) írhatjuk le, a szabályok meghatározását ***szabálymegadás***nak, vagy ***szabálybevitel***nek nevezzük (2. és 3. ábra).

A deduktív gráf minden csomópontjához meg kell adnunk egy szabályrendszert. Nem muszáj minden szabályt megadni, hiányos rendszerrel is működhet. Akkor lazább eredményt

kapunk. A csomópontban lévő tulajdonság lesz a szabályrendszer kimenő tulajdonsága, a hozzá kapcsolódó tulajdonságok pedig a szabályrendszer bemenő tulajdonságai.



REFERENCIA	nem megfelelő	gyenge	megfelelő	jó	kiváló
FINANSZÍROZÁS					
nem elfogadható	nem elfogadható	nem elfogadható	nem elfogadható	nem elfogadható	nem elfogadható
gyenge	nem elfogadható	nem elfogadható	nem elfogadható	nem elfogadható	nem elfogadható
közepes	nem elfogadható	nem javasolt	ajánlható	ajánlható	ajánlható
jó	nem elfogadható	nem javasolt	ajánlható	ajánlható	kiváló
kiváló	nem elfogadható	nem javasolt	támogatott	támogatott	kiváló

2. ábra: A függőség „hogyan”-jai, a szabályok (2D)

A ***következtetés*** (Reasoning) a szabályok aktivizálása. A szakértő ***esetek***et (Cases) ad meg, és kimondja, hogy a bemeneti tulajdonságok mely értéke(i) jellemzi(k). Ezután a keretrendszer a bevitt szabályok alapján következtet a függő tulajdonságokon keresztül a kimeneti tulajdonságra. Ennek leírására a következő jelöléseket használjuk:

Tulajdonságok nevei: A, B, C, ... (a kimeneti tulajdonság X)

Értékek: $A=\{a_1, a_2, a_3, ...\}$; $B=\{b_1, b_2, b_3, ...\}$; $C=\{c_1, c_2, c_3, ...\}$; ... $X=\{x_1, x_2, x_3, ...\}$

Ha az ***A*** tulajdonság $\boldsymbol{a_1}$ *és* a ***B*** tulajdonság $\boldsymbol{b_2}$ *és* a ... *akkor* az ***X*** tulajdonság $\boldsymbol{x_1}$. Matematikailag a következőképpen lehet leírni:

$$\text{Ha } A=a_1 \wedge B=b_2 \wedge C=c_1 \wedge ... \Rightarrow X=x_1 \quad (1)$$

> *Ha a finanszírozás jó és a referencia megfelelő, akkor az ajánlat ajánlható. (2. ábra)*

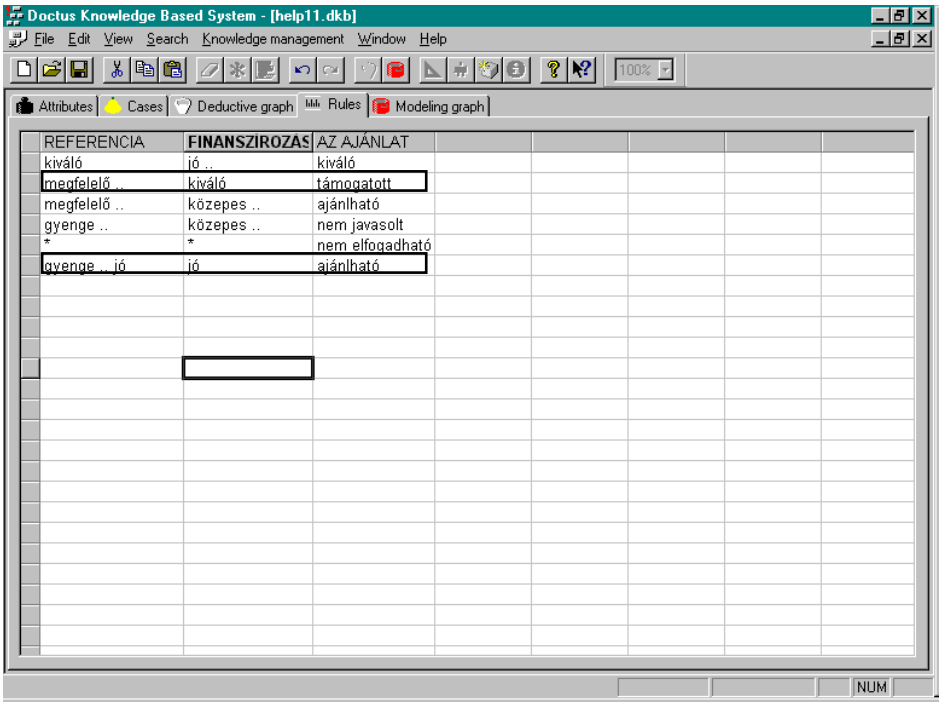
A szabályokat kétféleképpen vihetjük be, nevezzük őket egydimenziós (1D) és kétdimenziós (2D) szabálybevitelnek. A tudásrendező a szakértő gondolkodásmódjától és hozzáállásától függően irányítja és magyarázza a kétféle szabálybevitelt.

1D szabálybevitel

A szabálybevitel legegyszerűbb módja, ha a szabály minden bemeneti tulajdonságára megadunk egy-egy értéket, ami megfelel az (1) képletnek. Egy-egy tulajdonsághoz megadhatunk több értéket is. Amennyiben ezek nem egymás mellett lévő értékek, a shell több szabályként jeleníti meg. Értelmezhetünk szakaszokat is az értékekben, úgy, hogy azt mondjuk, hogy az ***A*** tulajdonság a_2 és a_5 között van, ami matematikailag egy zárt intervallumnak felel meg:

Ha $A \in [a_2, a_5] \wedge B = b_2 \wedge C = c_1 \wedge \ldots \Rightarrow X = x_1$. ..... (2)

> *Ha a referencia gyenge és jó között van és a finanszírozás jó, akkor az ajánlat ajánlható. (3. ábra)*



3. ábra: A függőség „hogyan”-jai, a szabályok (1D)

Amennyiben valamelyik tulajdonságnál az intervallum eléri a szélső értéket, leírhatjuk úgy is, hogy az ***A*** tulajdonság legalább a_2. Matematikailag ez a következőnek felel meg:

Ha $A \geq a_2 \wedge B = b_2 \wedge C = c_1 \wedge \ldots \Rightarrow X = x_1$. ..... (3)

> *Ha a referencia legalább megfelelő és a finanszírozás kiváló, akkor az ajánlat támogatott. (3. ábra)*

Ha egy szabály minden bemeneti tulajdonságához egy-egy érték tartozik, egyszerű szabályról beszélünk, ilyen az (1). Ha legalább egy bemeneti tulajdonsághoz nem egy konkrét értéket adunk meg egy szabályban, komplex szabályról beszélünk, ilyen a (2),(3). Okosko-

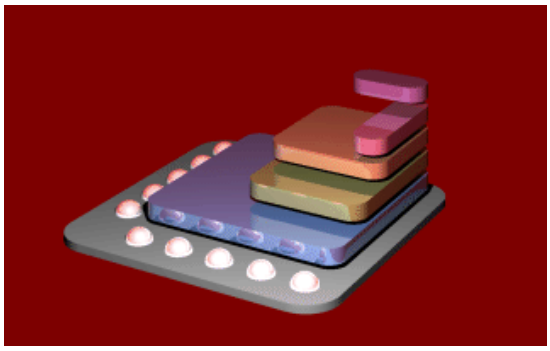
dásunk leírása annál egyszerűbb, minél kevesebb komplex szabállyal tudjuk kifejezni ugyanazt. Egy okoskodás természetesen sokfajta komplex szabályrendszerrel leírható. A kevesebb komplex szabály áttekinthetőbben írja le tudásunkat, de nem az a legfontosabb, hogy minél kevesebb legyen a komplex szabály. A lényeg, hogy olyan komplex szabályokat kapjunk, amelyek elég jónak tűnnek.

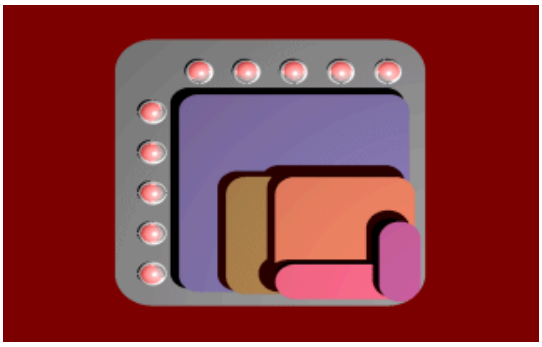
A shell megengedi azt is, hogy a szabályok átfedjék egymást. Mindig az kerül a legfelülre, amit utoljára vittünk be. Ez azért nem vezet ellentmondásra, mert a szoftver a szabályokat „felülről lefelé" olvassa, és abbahagyja a keresést, mihelyt talál egy olyan szabályt, amely megfelel az adott esetnek. Így a lejjebb elhelyezkedő, az esetet szintén leíró, esetleg másmilyen eredményt adó szabályokat nem olvassa el, azt mondjuk, hogy azok a rejtett szabályok (Hidden Rules).

2D szabálybevitel

A tudásrendező egy mátrixot hoz létre, melynek sorait és oszlopait a szakértő által megadott bemeneti tulajdonságok értékei alkotják. Minden lehetséges kombinációnak a mátrix egy-egy mezője felel meg. A mátrix akkor átlátható, ha nincs több mint két-két tulajdonság a sorokban és az oszlopokban. Ez egy újabb érv amellett, hogy 3-4 tulajdonságnál ne akarjunk többet kezelni egy szabályon belül. Egyszerű szabályt egy mező, komplex szabályt több mező kiválasztásával tudunk megadni. Ha a kijelölés nem írható le egyetlen komplex szabállyal, a shell több komplex szabályt fog generálni. Ezek értelmezése a tudásrendező feladata. A tudásrendező változtatja, hogy mely tulajdonság értékei alkossák az oszlopokat és melyek a sorokat, ezzel segíti a szakértőt a komplex szabályok kimondásában. Ami az egyik állapotban, pl. egy oszlop volt, az a másik állapotban néhány különálló mező lesz, vagy egy másik sor vagy oszlop.

> *Tekintsük a táblázatot egy felületnek, melyre, egymás tetejére színes papírokat rakunk. Egy színes papír egy komplex szabálynak felel meg. Ez azt jelenti, hogy egy színes papírhoz a kimeneti tulajdonságnak egy értéke tartozik, de egy kimeneti értékhez tartozhat több színes papír is. Ha felülről merőlegesen ránézünk a felületre, minden mezőn csak a legfelső rétegben lévő papír színét látjuk, a papírok nem átlátszóak. (4. ábra)*





4. ábra: Egy színes papír, egy komplex szabály

A szabályok rendezése

A tudásbázis akkor van készen, ha a szakértő egyetért a szabályokkal, és azokkal a következtetésekkel is, amiket a shell ad az esetekre. Ha egy következtetéssel nem ért egyet a szakértő, meg kell változtatni a hozzá tartozó szabályt. Ez egy új szabály bevitelét jelenti, ami a meglévők fölé kerül. Sok változtatással nagyon sok egymást részben vagy egészében takaró szabályt kaphatunk. Így a szabályrendszer a szakértő számára áttekinthetetlenné válik. Ilyenkor a tudásrendező műveleteket végez a szabályokkal.

Azok a szabályok, melyek teljesen rejtve vannak, soha sem aktivizálódnak. Ezeket a tudásrendező az 1D megjelenítésben meg tudja mutatni a szakértőnek. Ha megegyeznek abban, hogy egy rejtett szabály valóban felesleges, kitörölhetik. A következtetést ez nem befolyásolja. 1D-ben a tudásrendező a részben rejtett szabályok látható részét (Visible Parts) is meg tudja mutatni. Ezek a megjelenítések segítik a tudásrendezőt a szakértő által kimondott szabályok értelmezésében.

A komplex szabályokat szétrobbanthatjuk két vagy több egyszerű vagy komplex szabályra. A szétrobbantás ott történik, ahol egy tulajdonság több értéket vehet fel egy szabályban. Ennek a műveletnek az ellentéte, ha szabályokat vonunk össze.

Ezen lépések kombinációjával vezeti el a tudásrendező a szakértőt egy kevesebb komplex szabályból álló szabályrendszerhez.

Ha vannak preferenciák

A szakértő az egyes tulajdonságokhoz tartozó értékeket megadhatja nominális, vagy ordinális skálán. Nominális skálán nem juthatunk ellentmondásra. Ha az értékek között jósági sorrend van (ordinális skála), ellentmondásra juthatunk. Ha egy szabálynál, semmilyen tu-

lajdonság sem vesz fel rosszabb (jobb) értéket, mint a másik, de a szabálybevitelkor ros--szabb (jobb) kimeneti értéket adunk neki, ellentmondást kapunk. Ezt nevezzük inkonzisztenciának. Ilyenkor azt mondjuk, hogy az általunk adott érték „túl rossz” (túl jó).

Ha a szakértő preferenciákat adott meg, a shell javaslatokkal támogatja a szabályrendszer kialakítását. Néhány szabály bevitele után olyan kimeneti értékeket fog javasolni, amelyek nem vezetnek inkonzisztenciára. Az inkonzisztenciát nem kell minden esetben elkerülni, hiszen ez csak annyit jelent, hogy a szakértő preferenciái nem következetesek. Mivel preferenciáink kusza hierarchiát alkotnak, ez teljesen elképzelhető. Lehet, például, hogy valamelyik tulajdonsághoz tartozó értékek a szakértő fejében nem egy egyenes, hanem egy kör mentén helyezkednek el.

> *Ha egy állásra keresünk valakit, elképzelhető, hogy a nagyon fiatal ugyanolyan rossz, mint a nagyon öreg. Ezt úgy jeleníthetnénk meg, hogy a skálát körré hajlítjuk, így a két vége egy pontba kerül.*

Egyszerű szabályrendszert kapunk, ha alulról építkezünk. A legrosszabb kimeneti értékkel foglalkozunk először. Kijelöljük a teljes táblázatot, és úgy gondolkodunk, hogy az legalább a legrosszabb kimenetet adja. Ezután vesszük a második legrosszabb kimenetet. Kijelölünk egy területet, mely egy, vagy kevés szabállyal leírható, és úgy gondolkodunk, hogy az ilyen és ilyen tulajdonság az legalább a második legrosszabb kimenetet eredményezi. Áttérünk a harmadik legrosszabbra, azokat a mezőket keressük, melyekre legalább a harmadik legrosszabb kimenet lesz igaz, és így tovább, a legjobb kimenetig. Matematikailag ez a következőképpen értelmezhető:

$$\text{Ha } A=a_1 \wedge B=b_2 \wedge C=c_1 \wedge \ldots \Rightarrow X \geq x_1 \quad (4)$$

> *Ha a referencia bármilyen és a finanszírozás bármilyen, akkor az ajánlat (legalább) nem elfogadható. Ha a referencia legalább gyenge és a finanszírozás legalább közepes, akkor az ajánlat (legalább) nem javasolt... (3. ábra)*

Igen, én így gondolom!

A tudásrendező munkája véget ért. A szakértő már le tudja írni az explicit szempontjai közötti látens kapcsolatokat. Így érvelni tud, alá tudja támasztani ítéletét, amit szimata sugallt. Véleménye a döntéshozót támogatja. Aki megértette a tudásrendezés folyamatát, nem várja el egy számítógépes programtól, hogy az Alt+Shift+F2-re döntést hozzon helyette.

A sikeres tudásrendezés eredménye olyan kevésszámú komplex szabály, amellyel a szakértő egyet tud érteni: „Igen, én így gondolom!”

Felhasznált irodalom

1. Baracskai Zoltán: A profi vezető nem használ szakácskönyvet. „Szabolcs-Szatmár-Bereg megyei Könyvtárak” Egyesülés, Nyíregyháza, 1999.
2. Mérő László: Észjárások. Tericum, Budapest, 1997.
3. Polányi Mihály: Tudomány és ember. Argumentum, Budapest, 1997.
4. Tímár András: Doctus felhasználói kézikönyv. Doctus bt.Budapest, 1999.