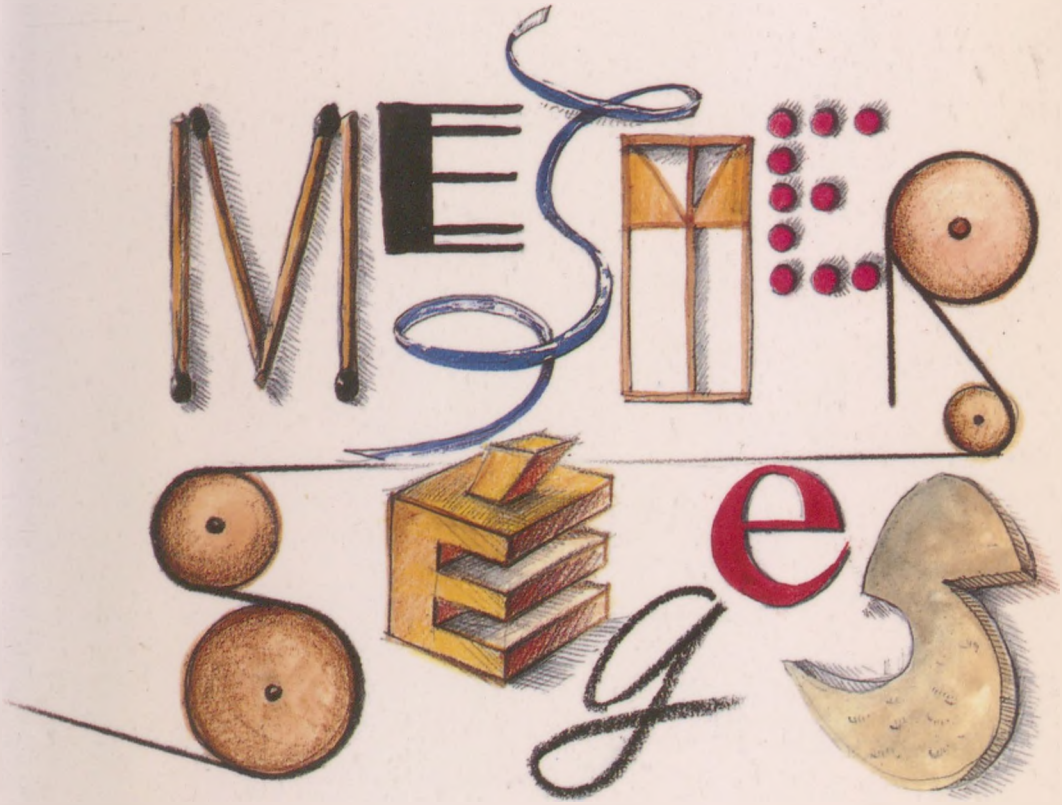
Jenny Raggett – William Bains

MESTERSÉGES INTELLIGENCIA

A–Z



AKADÉMIAI KIADÓ, BUDAPEST

MESTERSÉGES INTELLIGENCIA A–Z

Jenny Raggett – William Bains

MESTERSÉGES INTELLIGENCIA A–Z

Akadémiai Kiadó, Budapest

A fordítás alapját képező kiadás:
Jenny Raggett–William Bains:
Artificial Intelligence from A to Z
Chapman and Hall, London, 1992

Fordította
Kepes János

A fordítást az eredetivel egybevetette
Rátz Miklós

ISBN 963 05 6784 9

Kiadja az Akadémiai Kiadó
1117 Budapest, Prielle Kornélia u. 19–35.

Első magyar nyelvű kiadás: 1994

Printed in Hungary

Tartalomjegyzék

Bevezetés

Mi az ördög lehet az a „szakértői rendszer"? Hogyan működnek a „logikai programnyelvek"? Mit jelent az „elemző" és a „természetes nyelvi felület"? Vagy éppenséggel a „neuronhálózat"? A *Mesterséges Intelligencia A-tól Z-ig* a mesterséges intelligenciakutatásban és rokon területeken használatos mintegy 200 kifejezéshez ad rövid, nem szakmai jellegű magyarázatot.

A Mesterséges Intelligencia A-tól Z-ig nem tudományos kézikönyv, és nem is annak szól, aki kimerítő, terjedelmes definíciókra vágyik. Ez a könyv ügyvédek, újságírók, könyvtárosok, szaktanácsadók, nyelvészek, orvosok és még ezer szakma képviselőinek íródott, akik munkájuk során találkozhatnak egy-egy MI-kifejezéssel, és szeretnék a lehető legegyszerűbben megtudni, mit is jelent.

A könyv emellett arra is használható, hogy valamelyik adott MI-problémával kapcsolatban elmélyítsük ismereteinket: ebben segítenek a témát körbejáró kereszthivatkozások és olvasmánymegjelölések. Aki eredetileg a „szakértői rendszer" címszót lapozta fel, azon keresztül el fog érkezni a „tudástervezés"-hez, „hátra láncoláshoz", „magyarázó rendszerhez", „együttműködő rendszerhez", és így tovább. Hasonlóképpen, aki a „szintaxist" keresi, egyúttal a „szemantikáról", „természetesnyelvfeldolgozásról" és „pragmatikáról" is megtudhat valamit.

Minden címszót egy apró ikonnal is megjelöltünk, amely elárulja, hogy az adott kifejezés az MI melyik területére tartozik. A különböző ikonok számából (összesen12), az olvasó arról is képet alkothat, hogy az MI hány különböző területet fog át, és valóban milyen nehéz felmérni, hol kezdődik, és meddig is terjed. A szakértőirendszer–technológia, a számítógépes látás, a filozófia, a programozási technikák, a nyelvészet és még számos tudományág területeivel részben át is fedik egymást. (Az AI [artificial intel-

ligence] szimbólum a nemzetközileg elfogadott MI angol megfelelője. – *A ford.*)

Noha igyekeztünk a leggyakrabban fellelhető kifejezéseket összegyűjteni a könyvben, nem vettünk bele a kereskedelemben kapható, MI-technikákon alapuló számítógépes programokat. Ennek oka egyrészt, hogy ezáltal nagyobb általánosságot kívántunk biztosítani a könyvnek, másrészt el akartuk kerülni, hogy a forgalmazott szoftverrel kapcsolatban kicsinyes vitákba és rágalmazásokba keveredjünk. Ezért ezek többnyire a 70-es évekből és a 80-as évek elejéről, az MI-kutatás aranykorából származó, tudományos célú rendszerek.

Az információk keresését megkönnyítendő, a magyar kiadáshoz csatoljuk a címszavak angol nyelvű mutatóját. Minden címszó után megadjuk a hozzá kapcsolódó kifejezések utaló szavait is, hogy az olvasó kedvére kalandozhasson a rokon témák között.

Előszó

A könyv könnyen érthető bevezetést nyújt az Olvasónak a Mesterséges Intelligencia-kutatás (MI) és a rokon területek számos kulcsproblémájához. A rövid, lényegre törő szócikkek garantálják az ábécérendbe szedett kulcsszavak gyors megértését mind a szakmabeli, mind az érdeklődő olvasó számára. Ebben rajzok és ábrák, valamint számos ismert és kevésbé ismert példa is segíti.

A szerzők nem kerülték ki a nehezebb témákat sem – például a neuronhálózatok erősen matematikai jellegű témáját is friss, intuitív előadásmódban közvetítik.

Aki nem elégszik meg a szokásos szótári definíciókkal, ám idejéből nem futja szakfolyóiratok tanulmányozására, épp a megfelelő könyvet tartja a kezében.

Ian Alexander,
Logica

Köszönetnyilvánítás

A könyv leegyszerűsítve közelíti meg ezt a nehéz szakterületet, mégis törekszik arra, hogy helytálló legyen. Ez sokkal kevésbé sikerülhetett volna Martin Lam (konzultáns), David Raggett és Steven Knight (Hewlett Packard), Mike Gray (IBM), Peter-Fred Thomson (Inmos), Ian Alexander (Logica), David Anderson (PA Consulting Group), Jeremy Bennett, Harry Collins és Geoff Smith (University of Bath) támogatása és közreműködése nélkül. A megmaradt hibákról már nem ők tehetnek. Hálásak vagyunk továbbá Sheenagh Orchardnak és a szerkesztőknek, akik a lehető legjobban szerkesztették a szerkesztendőket, a felülmúlhatatlan Piers Burnettnek, akárcsak családunknak és gyermekeinknek, hogy minden provokációnk ellenére sem szaladtak világgá.

CÍMSZAVAK A–Z

CÍMSZAVAK
A–Z

ADATFOLYAMAT

Általános
számítógépes
kifejezés

Az adatfolyamat problémák elemzésére és megoldására szolgáló módszer. Arra fordít nagy figyelmet, hogy a probléma adott bitjei milyen más bitek által generált adatoktól függenek. Egy közönséges számítógépnél a soron következő utasítás kiválasztása a legutoljára végrehajtott utasítástól függ. Az adatfolyamatgépnél a választás azon múlik, hogy az adott pillanatban milyen adatok állnak rendelkezésre a feldolgozásra. Az utasítások végrehajtását nem a program logikai folyamata vezérli, hanem a rendelkezésre álló feldolgozandó adatok.

Miután minden számítási rendszerben egyidejűleg számos adatelem állhat feldolgozásra készen, az adatfolyamat-nyelvek jól alkalmazkodnak a párhuzamos feldolgozási módszerek elméletéhez, ahol egyszerre több processzor dolgoz fel különböző adatrészleteket. Nagyméretű problémákat azonban a gyakorlatban nehéz adatfolyamat-módszerekkel megoldani, mivel az adatfolyamat-processzorok számának növelésével meredeken emelkedik a szükséges kommunikáció mennyisége, és így ez egyre nagyobb részt igényel a teljes feldolgozási időből.

Lásd még: Párhuzamos feldolgozás

ALAPFELTEVÉSES KÖVETKEZTETÉS

Általános
MI-kifejezés

Az „alapfeltevéses következtetés" kifejezés olyan számítógépes következtetési formákat jelöl, amelyek szigorú értelemben véve nem tartoznak a klasszikus logikához.

Számos MI rendszer valamilyen logika alkalmazásával próbál következtetéseket levonni a rendelkezésére álló tényekből és szabályokból. Noha az ilyen rendszerek logikai értelemben mindig helyesen járnak el, egy zárt világban mozognak: igen korlátozott számú tényt „ismernek", és ezekkel is csak a megadott logikai szabályok alapján tudnak operálni.

Azoknál a rendszereknél, amelyek valóságos fogalmakkal dolgoznak, a tiszta logika a maga hamisítatlan formájában gyakran nem használható. Vannak olyan következtetések, amelyek hétköznapi értelemben helytállóak, noha logikailag nem igazolhatóak. Az alapfeltevéses következtetés alapján általánosítással tudunk következtetni bizonyos dolgok igazságára, de esetleg csak az adott gondolatmenetre érvényes módon. Tegyük fel például, hogy van egy adatbázisunk az autókról. Ebben a következő szabályt alkalmazhatjuk:

Benzinnel működik, ha nem gázolajjal.

Ez azt jelenti, hogy ha az adatbázis nem adja meg explicite, hogy egy autó gázolajjal működik, akkor általában azt mondhatjuk, hogy benzinnel működik, noha logikai értelemben erre semmiféle alapunk nincs.

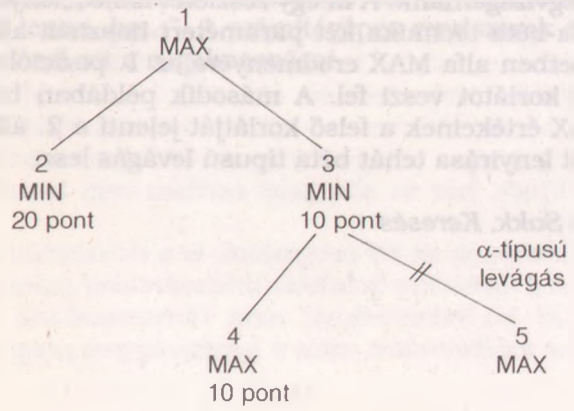
Lásd még: Zártvilág-feltevés

ALFA-BÉTA NYÍRÁS

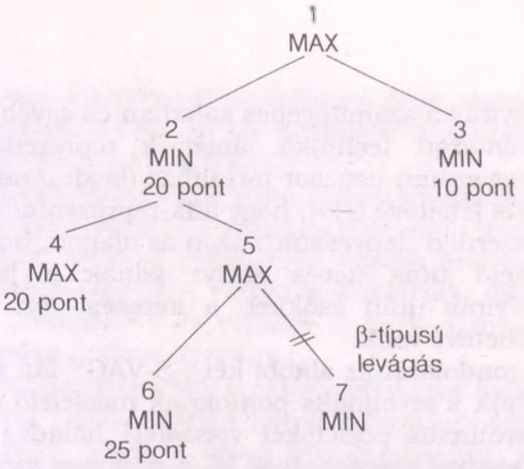
Keresés

Az alfa-béta nyírás a számítógépes sakkban és egyéb játékprogramokban alkalmazott technika, amelyek reprezentációjában a keresési fa nagyszámú csúcsot tartalmaz (lásd: *Játékok fája*). Az alfa-béta nyírás lehetővé teszi, hogy a fa-reprezentáció tekintélyes darabjait egyszerűen „lenyessük", azon az alapon, hogy az ezeken keresztül vezető utak kevés esélyt adnak a játékosnak a győzelemre. Nyírás után csökken a keresési tér, tehát könnyebben kezelhetővé válik.

A nyírás gondolatát az alábbi két ÉS-VAGY fán illusztrálhatjuk. Ha ismerjük a terminális pontoknak megfelelő állások értékeit, a nemterminális pozíciókét visszafelé haladva a minimax módszer segítségével számíthatjuk ki. A minimax módszer – ha a fát MAX játékos nézőpontjából rajzoljuk fel – azt írja elő, hogy az 1. állásból kiindulva a 2. és 3. pozíciók közül abba lépjen, amelyiknek magasabb értéke van (az ő szempontjából). Közben MIN ugyanezzel a stratégiával mindig a MAX szempontjából legkisebb (a sajátjából a legmagasabb) értékű pozíciót választja.



A fenti példában MAX az 1. pozícióból kiindulva csak két kiértékelt állást lát. Ha a 2. pozícióba lép, állásának becsülhető értéke 20 lesz. Ha azonban a 3. állást választja, MIN 10 pont körüli értékre szoríthatja vissza. A 3. pozíció értéke így legfeljebb 10, a helyes lépés tehát a 2. állásba vezet. A lényeg az, hogy ezt a döntést anélkül is meg lehet hozni, hogy az 5. állást vagy alatta bármelyiket kiértékelnénk. A fa egy részét mint érdektelen részletet már a keresés kezdeti szakaszában „lenyírhatjuk".



A második ábrán a 4. pozíció becsült értéke 20. Mihelyt kiszámítottuk, hogy a 6. állás 25 pontot ér, világossá válik, hogy MIN játékosnak, aki mindig a minimális értéket keresi, kerülnie kell az 5. csomópontot. Ez azt jelenti, hogy a 2. álláshoz 20 pont értéket rendelhetünk anélkül, hogy a 7. állást, vagy bármelyik utódját megvizsgálnánk. A fa egy részletét ismét „lenyírhattuk".

Az alfa-béta technika két paramétert használ, alfát és bétát. Az első esetben alfa MAX eredményére az 1. pozícióból kiindulva a 20 alsó korlátot veszi fel. A második példában béta 20 pont értéke MAX értékeinek a felső korlátját jelenti a 2. állásban. A 7. csomópont lenyírása tehát béta típusú levágás lesz.

Lásd még: Sakk, Keresés

ALGEBRA

Általános
MI-kifejezés

A számítógépes algebraprogramok az MI technikák legelső sikeres alkalmazásai között szerepeltek. Ezek a programok és általában a szimbólumfeldolgozó programok számos MI-technikát alkalmaznak, elsősorban a logikai programozás technikáját. A komputeralgebra-programok atyja a Macsyma, a 60-as évek végén az MIT-n elindított hatalmas projekt, amely mind a mai napig bővül. Ennek során matematikai szabályokból óriási adatbázist építettek fel, amely azután hozzáférhető (a szabályalapú rendszerek következtető gépéhez hasonló) viszonylag egyszerű „szabálykezelő", de a szakértői rendszerek módszereivel felhalmozott magas szintű matematikai szaktudás számára is.

Számítógépes matematikai programok számos sikert arattak, köztük az egyik legnagyobb a négyszínsejtés bizonyítása volt (mely szerint minden síkbeli térkép kiszínezhető négy színnel úgy, hogy két szomszédos ország soha ne legyen azonos színű). Sok eredményük (különösen a négyszínsejtés említett példája) igen sok és időigényes kézi munkát igényelt. A matematikusok átválthatnak az egyik szabályrendszerről a másikra, valahogy így: „Ez az algebrai probléma emlékeztet az egyik halmazelméleti kérdésre. Mi lenne, ha...". A számítógépes rendszerek csak újabban kezdik elérni ezt a rugalmasságot.

ALGORITMUS

Általános
MI-kifejezés

Az algoritmus lépésenként végrehajtható módszert ad valamely feladat elvégzésére. Például: „gondolj egy számra, és szorozd meg kettővel", vagy: „tégy egy teászacskót a csészébe, és önts rá előbb forró vizet, majd tejet".

Az úgynevezett hagyományos programozási nyelveknél, például Basicben vagy Cobolban, a programozó először a probléma algoritmusát alkotja meg. Ezt aztán lefordítja a programozási nyelvre, hogy a gép meg tudja oldani.

Az MI-programok általában nem fektetnek ilyen nagy hangsúlyt az algoritmusokra. A legtöbb olyan probléma esetében, amelyet MI-technikával oldottak meg, ha algoritmussal keresték volna a megoldást, az valószínűleg „exponenciális ideig" futott volna. Ez azt jelenti, hogy kisméretű problémáknál (például órarendkészítés kevés osztályos iskolában) az algoritmus gyorsan működhet, de nagyobb méretű problémáknál (amilyet például hasonló órarend jelentene nagy iskola és 50 tantárgy esetén) már használhatatlanul lassú lenne.

Az emberek a legtöbb probléma megoldásában nem algoritmusokra, hanem kevésbé szigorú módszerekre támaszkodnak. Ez azért lehetséges, mert alkalmanként megengedhető, hogy hibázzanak. Ily módon sokszor ésszerű az algoritmusokat heurisztikus ökölszabályokkal helyettesíteni, akkor is, ha ezek időnként hibás eredményt adnak. Az orvos heurisztikus szabályok segítségével állapítja meg, hogy a beteg náthás, és többnyire a helyes diagnózisra számítunk, bár az ellenkező esetben sem lennénk különösen meglepve.

Hasonlóképpen az MI-programok is a heurisztikákat részesítik előnyben a lépésekre bontható módszerekkel szemben.

Noha az MI-technikáknál nem támaszkodunk annyira az algoritmikus módszerekre, mint a hagyományos programozásban,

ez nem jelenti azt, hogy egyáltalán nem alkalmazunk algoritmusokat. A keresési problémáknál, ahol a programnak egy probléma nagyszámú különböző megoldását kell végignéznie, algoritmusok biztosíthatják, hogy szisztematikusan és hatékonyan minden lehetséges megoldási módot végignézünk, amíg a megoldást meg nem találjuk.

Lásd még: Kombinatorikai robbanás

ALVEY

Finanszírozás

Alveynek annyi köze van Wellingtonhoz, Broughamhez és Mae Westhez, hogy mindegyikük neve egy-egy alkotásukkal kapcsolódott össze. Az Alvey-bizottság (amely elnökéről, John Alveyről kapta nevét) 1983-ban Angliában a japán Ötödik Generációs Projekt kezdeményezésre reagálva kooperatív kutatási-fejlesztési programot állított össze. Nagy-Britannia kormánya finanszírozásával az ALVEY 350 millió fontos programmá nőtte ki magát, amely a szoftvertervezés, nagyon nagy mértékű integráció (VLSI), intelligens tudásalapú rendszerek (IKBS) és ember–gép kapcso-lat (MMI) területeit ölelte fel.

Az ALVEY hatalmas lökést adott a nagy-britanniai kutatók közösségének, és egyes elemei külön-külön is hozzájárultak az új technikák elterjesztéséhez. Az angol ipar azonban nem volt a japánhoz hasonlóan lelkes, ezért a kormány a folytatás ellen döntött, és inkább a Kereskedelmi és Ipari Minisztérium, valamint a Tudományos-Műszaki Kutatás Tanácsa normális kutatási költségvetésére támaszkodott, az ESPRIT angol résztvevőinek várható támogatása mellett.

Lásd még: Ötödik generációs projektek, ESPRIT

ARGUMENTUM

Általános
számítógépes
kifejezés

Az MI-ben az „argumentum" szót az alábbi jelentésekben szokták használni:

1. A predikátum-kalkulusban az argumentum egy-egy elemre utal, amelyről a kijelentés valamit állít. Például a

   BENN (MACSKA, SZOBA)

   kijelentésben a MACSKA, SZOBA elemek a BENN predikátum argumentumai.
2. Az „argumentum" szót a „paraméter"-hez hasonló értelemben is használhatjuk. Például a Lisp nyelvben az eljárások az argumentumokkal dolgoznak. A PRINT eljárás argumentumai egy mondat szavai lehetnek, amelyeket a program kinyomtat papírra.
3. Az „argumentum" harmadik értelme a matematikai logikában szerepel. Amennyiben bizonyos tények vagy kijelentések együttesen fennállnak, ebből más kijelentések igazságára is lehet következtetnünk. Azokat a szabályokat, amelyek alapján ismert kijelentésekből továbbiakat vezetünk le, „argumentumformáknak" nevezzük. Az egyik ilyen klasszikus argumentumforma a *modus ponens* (lásd *Dedukció*).

ATOM

Általános számítógépes kifejezés

Az MI-be eredetileg Lisp-kifejezésként került be, de másutt (például a Prologban) is használják, egy programnyelv legkisebb, tovább nem osztható kifejezésére. A Lispben az atom a legegyszerűbb elem, amellyel dolgozhatunk: például az 1 számot jelentő „1" szimbólum is egy atom.

Az atomok sorozata listát alkot. Egy lista azonban nemcsak atomokat, hanem más listákat is tartalmazhat. Például a

(Macdonald tanya (disznók tehenek birkák tyúkok))

lista három elemet tartalmaz, de ebből csak kettő atom („Macdonald" és „tanya"), a többi elem egy újabb listát alkot. A (disznók tehenek birkák tyúkok) lista viszont négy atomból áll.

Lásd még: Lisp

AUTOMATA

Elmélet/
Filozófia

A számítógép-tudományban az automata a számítógép rendkívül egyszerű modelljét jelenti. Van egy memóriája, amely a „belső állapotait" tárolja (sokszor mindössze néhány biten), valamint egy potenciálisan végtelen, négyzetekre osztott input-„szalag" bemenő jelekkel, amelyre a „számítások során" maga is képes jeleket írni. Ezenkívül van valahány szabálya, amely meghatározza, hogy a belső állapotától függően mihez kezdjen az éppen olvasott inputjellel. A szabályok jellegétől függően a automaták hierarchiába rendeződnek. A legáltalánosabb az úgynevezett *Turing-gép*, amely az olvasott jeltől és a belső állapotától függően új jelet ír a szalag éppen vizsgált helyére, és egy négyzettel jobbra vagy balra lép a szalagon. A legegyszerűbb típusú automatát „véges állapotú automatának" nevezik, ez csak egy irányban képes lépkedni.

Az automatákat „gondolatkísérletekben" használjuk, hogy megtudjuk, mire képes a számítógép és mire nem. Turing megmutatta, hogy a Turing-gépek egyenértékűek bármely számítógéppel, ha elég bonyolult programot írunk a szalagjukra. Ily módon a nagyobb, valóságos számítógépek matematikai modelljét alkotják. Azt mondjuk, „teljes számítási képességgel" bírnak. Egy teljes számítási képességű automatával megoldható probléma mindig megoldható alkalmas számítógéppel is, és fordítva, ami nem oldható meg ilyen automatával, az semmilyen számítógéppel sem.

Lásd még: Turing-gép

AUTOMATIKUS PROGRAMOZÁS

Általános
MI-kifejezés

Az automatikus programozás (AP) azt jelenti, hogy a számítógép segítségével írunk programokat. A felhasználó valamilyen magasszintű nyelven írhatja le a programot, és a leírásnak nem is kell mindig teljesen egyértelműnek lennie. Ha egy programot már egész pontosan leírtunk, az átfordítás valamely ismert számítógépes nyelvre már viszonylag mechanikus dolog. A leírás a következő módszerekkel történhet:

1. *Formálisan.* Ez analóg a hagyományos programozással (csak még szigorúbb); a leírás valamely formális nyelvét, például a predikátum-kalkulus nyelvét használja. Ez lehetővé teszi a program logikai struktúrájának tömör megadását, anélkül, hogy a program működésének mechanikus részleteivel kellene törődnünk.
2. *Példa alapján.* Néhány példával adjuk meg, hogy mit kell a programnak végeznie, és a számítógép írja meg az erre szolgáló programot. Ez a leírás azonban ritkán teljes. Az inputtól sokféle út vezet az outputig, de teljesen különböző inputtal csak egy fog helyes eredményt adni. Ez a probléma az indukció minden formájában fennáll.
3. *Természetes nyelven.* „Nem programozók" számára ez a programírás legvonzóbb lehetősége. Sokszor interaktív módon működik, a program megpróbálja kiküszöbölni a többértelműséget, és kérdésekkel pótolni a kezdeti leírás hiányosságait.
4. *Grafikus leírással.* A program elemeit a képernyőn látható kis képek („ikonok") segítségével írjuk le. Az ikonok kiválasztására, és közéjük vonalak berajzolására WIMPS-interfészt használunk.

Az AP sok tekintetben még csak kutatási terület, noha egyes eredményeit már általánosan használják compiler-interpreter kombinációkban MI-környezetekben. Számos módszere ismeretes, amelyek logikai programozást, tudástervezést, problémamegoldó módszereket alkalmaznak. Olyan problémákat is képesek kezelni, mint a hiányos információ, inkonzisztencia (ellentmondásos leírás) vagy a végső program hatékonysága. Az automatikus programozás egy viszonylag új megközelítésében elhagyunk minden szigorú definíciót, és genetikus algoritmussal statisztikusan fejlesztjük ki a programot.

Lásd még: Tanulás

BAYESI LOGIKA

Logika

Bayes tétele arra szolgál, hogy valaminek a legvalószínűbb okát ki tudjuk deríteni. A következményeket annak alapján tudjuk bizonyos okokhoz kapcsolni, hogy az okok milyen valószínűséggel járnak az adott következményekkel. Ez az *a posteriori* logika: valaminek az okát keressük, miután már ismerjük a következményt.

Ez a fajta következtetési mód nagyon jól alkalmazható szakértői rendszerekben, ahol gyakran ismerjük, hogy egy adott ok *általában* milyen eséllyel jár az adott következménnyel, és arra vagyunk kíváncsiak, hogy a *konkrét* esetben mi a valószínűsége, hogy ez az ok magyarázza az adott következményt. (A gyakorlatban ez sokszor nagyon nehéz.) Bayes tétele alapján olyankor is kiszámíthatjuk a valószínűséget, ha több tényező is közrejátszik a következmény létrejöttében: a bayesi logika épp ilyen kombinációkon alapul. A bayesi logikát alkalmazó szakértői rendszer a valószínűségek listáját tartalmazó adatbázis alapján ad numerikus és logikai válaszokat. Az ONCOCIN, QMR, INTERNIST és MYCIN orvosi diagnosztikai rendszerek ilyesféle logika különböző változatait alkalmazzák, számos kisebb rendszerhez hasonlóan, amely a megfigyelt hatásokból kiindulva diagnosztizálja a problémák okait. A bayesi logika valószínűségei bizonyossági tényezők alakjában gyakran felbukkannak egyes tudástervezési kontextusokban is.

Lásd még: Bizonyosság, Dedukció, Szakértői rendszerek

BESZÉD-FELISMERÉS

Általános számítógépes kifejezés

Ez a beszéd dekódolását jelenti számítógépes inputnak alkalmas formába. Nem a beszédhangot, hanem tartalmát, a beszédet próbáljuk felismerni.

A probléma két részre bomlik: akusztikai-fonetika illetve nyelvészeti feladatra. Az utóbbi már a *természetesnyelv-feldolgozás* területére tartozik. Az előbbi azzal foglalkozik, hogyan lehet a beszédet hangok sorozatára bontani, a természetesnyelv-feldolgozó rendszerekbe betáplált írott szöveghez hasonlóan. A dekódolást az bonyolítja, hogy az élő beszédben a szótagok egymásra torlódnak, a hangokat nem külön, kis szünettel elválasztva ejtjük ki, hanem egymással összemosva, a szomszédos hangok hatására kissé meg is változtatva. Így például a szóhatárok azonosítása is meglehetősen nehéz feladat. Az ember ezt úgy végzi, hogy folyamatosan lefordítja magának a hallott szavakat. A hibás szegmentálás persze helytelen fordításhoz is vezet.

BESZÉD-SZINTÉZIS

Általános
MI-kifejezés

A beszédszintézis azt jelenti, hogy a számítógép nem akusztikus kódolt inputot (többnyire írott szöveget) hangos, „beszélő" outputtá alakít. Sok MI-alkalmazással ellentétben ez a feladat nagy sikerrel elvégezhető.

A legtöbb beszédszintetizáló rendszer adott nyelven írt szöveget „ejt ki". Ez egyáltalán nem triviális feladat, rengeteg szabályt, köztük egészen önkényes szabályokat is be kell vetni, hogy a hasonló hangalakú, de különbözőképpen kiejtendő szavak helyesen hangozzanak el. (A szokásos angol nyelvű tesztmondat a következő: „The rough wind brought the bough down": vajon felismeri-e a gép, hogy a rOUGH, brOUGHt és bOUGH szavak kiejtése teljesen különböző, a BOUGH és DOWN viszont majdnem egyformán hangzik?) A jobb rendszerek még a mondat intonációjára is ügyelnek: tehát a tudományos-fantasztikus filmekből megszokott furcsa, érdes, monoton zümmögő számítógépes dünnyögés egyáltalán nem a valóságos programok tapasztalataiból ered.

BETANÍTÓ PÉLDA

Neuron-
hálózatok

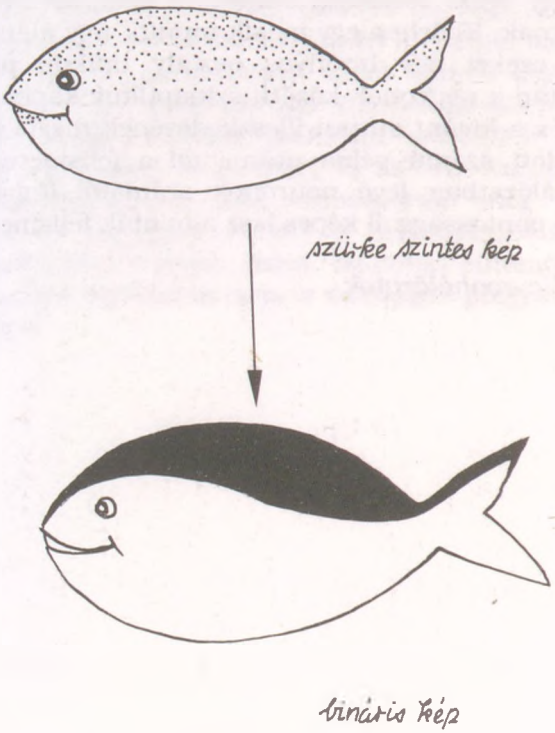
A betanító példák a neuronhálózatok „adatai". A legtöbb neuronhálózat szimuláció rugalmas, amennyiben a program indulásakor nem adottak a neuronkapcsolatok súlyai. Ahhoz, hogy egy neuronhálózat képes legyen felismerni egy mintát (mert ez a leggyakoribb és legsikeresebb alkalmazásuk), először meg kell tanítani, hogyan néz ki ez a minta. Ezt úgy végezzük el, hogy a célmintából egy egész sorozatot mutatunk neki, ezt nevezzük betanító példáknak. Eközben egy másik inputot úgy állítunk be, hogy a hálózat ezeket egy bizonyos osztály tipikus példányaiként kezeli. Ezután a neuronok közötti szinaptikus kapcsolatok súlyai az output és a kívánt output illeszkedésének megfelelően állítódnak be. Adott számú példa után (ami a felismerendő minták, illetve a hálózatban levő neuronok számától függ), a hálózat (valamilyen pontossággal) képes lesz a minták felismerésére.

Lásd még: Neuronhálózatok

BINÁRIS KÉP

Látás

Egy bináris kép csak két színt, rendszerint fehéret és feketét tartalmaz. Ez azt jelenti, hogy minden képpont közvetlenül bináris számokkal ábrázolható: 1 felel meg a fehérnek és 0 a feketének (vagy fordítva).



BIZONYOSSÁG

Általános
MI-kifejezés

A bizonyossági tényező annak a mértéke, hogy mennyire vagyunk biztosak egy tény fennállásában. Az automatikus következtetésnél használjuk, például szakértői rendszerekben, ahol bizonytalan információval kell dolgoznunk. Egyes szakértői rendszerekben bizonyos kijelentésekhez hihetőségi fokokat társítunk, és a bizonyossági tényezők ennek mértékét jelzik. Általában a –1-től 1-ig terjedő skálán helyezkednek el, (1 jelenti a teljes bizonyosságot, hogy a tény fennáll, 0 a teljes bizonytalanságot és –1 a teljes bizonyosságot, hogy a tény nem áll fenn).

A bizonyossági tényező a tény fennállásának valószínűségével függ össze. Ha így értelmezzük, statisztikai számításokat végezhetünk vele (a bayesi logika számításaihoz hasonlóan), és független állításokból (például az alábbiak), ellentmondásmentes, (bár csak statisztikailag megbízható) állításokat kaphatunk.

Állítás	*Bizonyosság*
A nagynénim városban él	0.7
A bikák vidéken élnek	0.9
A nagynénim épp most látott egy bikát	0.5

(Meglehetősen biztosan vagyunk benne, hogy minden bika vidéken él, kevésbé abban, hogy a nagynénim városban lakik, és egyáltalán nem vagyunk meggyőződve arról, hogy meg tudja különböztetni a bikát bölömbikától.)

Lásd még: Bayesi logika, Szakértői rendszerek

BOOLE-ALGEBRA

Elmélet/
Filozófia

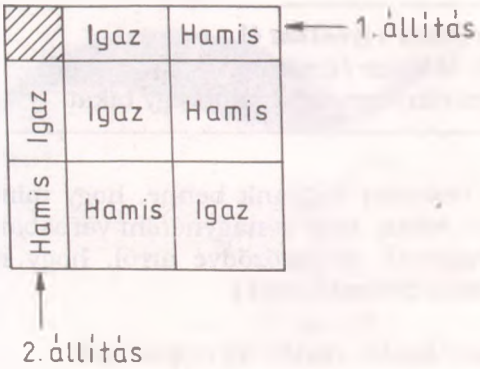
„A gondolkodás törvényei, amelyen a logika és a valószínűség matematikai elméletei alapulnak" című könyvében George Boole 1854-ben kifejtette a logikai műveletek szabályait az ÉS (AND), VAGY (OR) és NEM (NOT) kötőszavak alapján. Kimutatta, hogyan lehet egyes kijelentések eredendő igazságát vagy hamisságát bizonyítani pusztán a kijelentéseket képviselő szimbólumok segítségével. Ebből keletkezett azután a szimbolikus logika: ugyanúgy, ahogy az

$$a + b = c$$

szimbolikus összeadásban *a*, *b* és *c* számokat jelentenek, az

a AND NOT *b*

szimbolikus kijelentésben, például *b* egy olyasfajta logikai állítást jelenthet, mint „A tehén barna" vagy „Minden ember halandó".



A Boole-algebra alapvető segédeszköze az ún. *igazságtábla.* Ahogy két szám szorzatát a szorzótáblából olvashatjuk ki, kijelentésekkel végzett logikai műveletek „igazság"-értékét az igazságtábla adja meg. Ebben a táblázatban az „igaz" értéket hagyományosan 1-es, a „hamis" értéket 0 ábrázolja.

A Boole-algebrát széles körűen használják a számítógépes logikában, különösen a *kijelentés-kalkulus* alkalmazásainál.

Lásd még: Dedukció, Kijelentés-kalkulus

COMPILER

Általános
számítógépes
kifejezés

A compiler olyan program, amely valamely magas szintű programnyelvet (mint például a Fortran vagy a Cobol) gépi utasításokra fordit le. A gépi kódú fordítást *végrehajtható kódnak* is szokás nevezni. Általában compilerrel fordítják le a gép számára a Fortran, Pascal és C nyelvű programokat.

Hagyományos módon a compiler a teljes programot tekinti, és abból készíti el a fordítást gépi kódra. Ha a programban hiba van, vissza kell térni az eredeti programhoz (a *forráskódhoz*), ott kijavítani a hibát, és újra le kell fordítani. (Ebben különbözik az *interpretertől*, amely az utasításokat egyenként fordítja le, sorban egymás után, ahogy találkozik velük.) A compilerrel való fordítás előnye, hogy a keletkező gépikód-program gyorsabban fog futni, mint az interpretált programok.

Az MI-ben elhomályosult a különbség compiler és interpreter között, mivel olyan hibrid compiler-interpretereket hoznak létre, mint például a SmallTalk nyelv fordítóprogramja. Ily módon a SmallTalk gyors, és mégis könnyű a programokat javítani benne. Egyes hasonló rendszerek több lépésben fordítják le a programokat, először egy közbülső szintre, amelyet gyorsabban lehet interpretálni, mint az eredeti nyelvet, de javítani mégis könnyebb, mint a gép kódot.

Lásd még: Interpreter

CONNECTION-ELV

Neuron-
hálózatok

Ez a „connection-hálózat" típusú neuronhálózatok vizsgálatát jelenti. Sokszor használják általában a neuronhálózatok szinonimájaként. (Ez nem egészen helytálló. A connection-elv olyan számítórendszerek vizsgálata, ahol az egyszerű elemek közötti összeköttetések döntő szerepet játszanak a rendszer által végzett számítások szempontjából. A neuronhálózatok viszont olyan elemek hálózatai, amelyek az idegsejtekhez hasonlóan működnek. A valóságos idegsejtek pedig maguk is jelentős mennyiségű számítást képesek végezni.)

1. A hálózatok elemei (a „neuronok") kevés számú állapottal rendelkeznek, vagyis nagyon kicsi a memóriájuk. (Ezzel szemben egy mikrokomputernek óriási számú belső állapotai lehetnek, amelyek mind különböző, a memóriában tárolt dolognak felelnek meg.)
2. A neuronokat akármennyi összeköttetés kötheti a többi elemhez. Ezek serkentő vagy gátló összeköttetések lehetnek: a serkentő összeköttetések növelik, a gátlók pedig csökkentik annak az esélyét, hogy a cél-neuron a maga részéről „kisül".
3. Ezek a összeköttetések nem szállítanak kódolt információt.

Az információ kódolását a hálózat elemei között létező összeköttetések adják, nem maguk a rajtuk végighaladó üzenetek vagy az elemek memóriaállapotai: innen a „connection (összeköttetés)-elv" elnevezés. Ez azt jelenti, hogy sokszor nehéz „dekódolni" a hálózat outputját, végeredményét. Meg lehet azonban szerkeszteni a hálózatot úgy is, hogy egy adott neuron adott összeköttetése jelezzen egy speciális eredményt vagy állapotot.

A connection-hálózatoknak több olyan tulajdonsága van, amely bizonyos feladatok elvégzésére hatékonyabbá teszi őket a

hagyományos komputereknél. Ilyenek többek között az alábbi feladatok:

1. *Tanulás.* A *visszatáplálás* és visszaforgatás módszerével a connection-hálózatok nagyon gyorsan képesek tanulni.
2. *Stabilitás és biztonság.* Ha egy elem inputját vagy outputját kis mértékben megváltoztatjuk, az csak kismértékben érinti az egész rendszer outputját.
3. *Gyors konvergencia.* A connection-hálózatok sokszor igen gyorsan eredményt adnak valamilyen problémára, amellyel a hagyományos számítógépek csak nagyon lassan tudnának megbirkózni. Az ilyen problémákra jellemző, hogy sok lehetséges megoldásuk van, és erősen összefüggők, azaz a probléma egyik részével kapcsolatos döntések nagymértékben befolyásolják a többit. Tipikus példa a *Feladatkiosztás* és *Az utazó ügynök* problémája.

A connection-hálózat a neuronhálózatok egyik típusa, és így jól modellezhető párhuzamos működésű számítógépeken. Párhuzamos hardver hiányában pedig több szoftveres szimuláció áll rendelkezésre mikrokomputerekre. Néhány connection-hálózatot azonban ténylegesen is megépítettek, (nemcsak szoftverrel szimuláltak): jó példa erre a Silicon Retina, és a WIZARD képosztályozó gép.

Filmfőcímek mintájára azt mondhatnánk: a connection–hálózatok a biológiai idegrendszer „ötletén alapulnak". Sok neuronhálózat szimulációt egy teljesen hagyományos mátrix-kezelő programként képzelhetünk el, ahol a hálózatot egy mátrix reprezentálja, amelynek elemei a lehetséges neuron-kapcsolatok erősségét írja le. Tehát maguk a connection-hálózatok a lényegesek, és nem az, ahogyan elképzeljük őket.

Lásd még: Neuronhálózatok, Hopfield-háló, Szilícium-retina

CONNECTION MACHINE

Hardver

A Connection Machine, a Danny Hillis által feltalált, kereskedelemben kapható párhuzamos számítógép 65 536 (azaz 64K) processzort tartalmaz, amelyek egy hiperkocka-struktúrába kapcsolódnak össze. Ez egy MIMD (több utasítás – több adat) jellegű gép, vagyis mindegyik processzor minden pillanatban teljesen mással foglalkozhat, mint a szomszédai. A viszonylag kis (egyenként mindössze 512 byte memóriával rendelkező) processzorok együtt sokkal gyorsabban dolgoznak a leggyorsabb hagyományos számítógépnél. A programozó a processzorok összekapcsolásáról is dönthet: innen a gép elnevezése (Összeköttetés-gép), ami nemcsak a processzorok nagy számára utal, hanem a tetszőleges, a problémamegoldásnak megfelelő összekapcsolhatóságra is. Ennek a rugalmasságnak és nagyfokú párhuzamosságnak köszönhetően a Connection Machine alkalmas számos connection-elvű hálózat szoftveres szimulációjára, noha ő maga nem neuronhálózat.

Lásd még: Párhuzamos feldolgozás

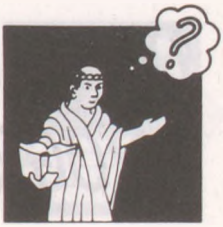
DARPA

Finanszírozás

A DARPA (Defense Advance Research Projects Agency: Magas szintű Honvédelmi Kutatási Projektek Hivatala) az Egyesült Államok hadügyminisztériuma alá tartozó hivatal, és olyan „tiszta" kutatási témákkal foglalkozik, amelyek hosszú távon honvédelmi jelentőséggel bírhatnak. A DARPA speciális területeken indít és támogat kutatásokat, tart fenn speciális kutatási eszközöket, így például az ARPANET elektronikusposta-hálózatot. A DARPA támogat többek között számos beszédfelismeréssel, neuronhálózatokkal, szakértői rendszerekkel és vizuális feladatokkal kapcsolatos kutatást.

Lásd még: ESPRIT, Ötödik generációs projektek

DEDUKCIÓ



Elmélet/
Filozófia

A dedukció: „logikailag helyes következtetés", vagyis olyan módszer, amellyel kiinduló állításokból szükségszerű következtetésekre juthatunk. A szabályok megengedik, hogy ismert kijelentések sorrendjét felcseréljük, de további feltevésekkel nem élhetünk. A kiinduló állítások az axiómák, vagy számítógépes rendszereknél a formulák, kijelentések, egy tudásbázis vagy adatbázis elemei. Általában a dedukció általános kijelentésekből speciálisakat állít elő, és alapvetően két típusa van:

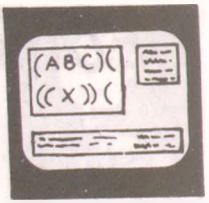
1. Modus ponens
   Axióma: Ha sült krumplit eszem, boldog vagyok
   Axióma: Most sült krumplit eszem
   Konklúzió: Tehát boldog vagyok
   [Ha A-ból következik B, és A teljesül, akkor B is teljesül.]

2. Modus tollens
   Axióma: Ha spenótot eszem, boldogtalan vagyok
   Axióma: Most nem vagyok boldogtalan
   Konklúzió: Tehát most nem eszem spenótot
   [Ha A-ból következik B, és B nem teljesül, akkor A sem teljesül.]

A dedukció logikai fogalmán alapul a *kijelentés-kalkulus*. Biztosan és előrelátható módon működik, így bizonyos programnyelvek, például a Prolog alapjául is szolgál, noha a Prolog nem tartalmazza a teljes klasszikus logikát.

Lásd még: Kijelentés-kalkulus

DEKLARATÍV PROGRAMNYELVEK



Programozási technikák

Egy deklaratív programnyelvben a programozó egy sor „állítást", vagy tényt fogalmaz meg, amelyekből a rendszer logikai úton újabb tények fennállására következtet. Egy deklaratív program például az alábbi módon leírt tényeket tartalmazhatná:

A és B – C típusai
A kedveli B-t
A rendelkezik B, C és D tulajdonsággal
A igaz, ha B, C és D igaz

A deklaratív nyelvek egy nagyobb osztályát teszik ki a funkcionális vagy applikatív nyelvek: például az ML, Hope és Miranda.

Lásd még: Tudásbázis, Prolog

EGYÜTTFUTÁS

Általános számítógépes kifejezés

Azt jelenti, hogy az ember számítógépén egyidejűleg két vagy több dolog történik. Például a legtöbb mikrogépes rendszerben a printer képes együtt futni a géppel, tehát miközben kinyomtat egy oldalt, a számítógép már készíti elő a következőt. Ez tehát voltaképpen a párhuzamos feldolgozás egyik elnevezése, bár általában olyan rendszerekre tartják fenn, ahol viszonylag önálló részek teljesen különböző feladatokat látnak el (például printelés, számítás, illetve lemezről olvasás). Nagy gépeken futó időosztásos rendszerek képesek szimulálni sok kis gép együttfutását: noha egyetlen gépről van szó, minden felhasználó úgy érzékeli, hogy külön géppel dolgozik. Ez az illúzió csak akkor foszlik szét, ha egyszerre több százan dolgoznak, mert ekkor mindenkire a számítógép idejének csak egy parányi töredéke jut, és a felhasználók számának növekedésével lelassulnak a gép reakciói.

Lásd még: Párhuzamos feldolgozás, Folyamat

ELASZTIKUS LOGIKA

Szakértői rendszerek

Az elasztikus logika elve alapján döntjük el egy tudásbázisban, hogy egy adott esetben melyik szabályt kell alkalmazni. Szakértői rendszer alkalmazásokra találták ki.

Az elasztikus logika hasznossága abban áll, hogy segítségével a szakértői rendszer különböző forrásokból származó, részben esetleg ellentmondó szakismeretet is képes magába foglalni. Amikor a szakértői rendszertől tanácsot kérnek, az első percben esetleg egy A szakértő megkérdezésével nyert szabályrendszert, a második percben egy B szakember szakismereteit alkalmazza. A harmadik percben aztán visszatérhet A, a negyedikben pedig ismét B véleményéhez. A felhasználó tehát örülhet, hogy a szakértői rendszer nem egyoldalú, elfogult, és egy területen több szaktekintély véleményét képviseli.

ELEMI VÁZLAT

Látás

Egyes számítógépes látórendszerek képfeldolgozásában ez az első felismerési fázis. Az elemi vázlat olyan, mint egy vonalas karikatúra, amelyben a képet az adatokból kivont „élek" alkotta vonalakra redukáljuk. Az elemi vázlatnak az alábbi két fajtája ismeretes:

1. *Nyers első vázlat.* Ez azoknak az éleknek a gyűjteménye, amelyeket a látórendszer a kiinduló képben felfedez. Karikatúra-rajzra emlékeztet, csak a jelben levő „zaj" téves vonalakat is eredményez.
2. *Teljes első vázlat.* A nyers első vázlatból a téves vonalak eltávolításával keletkezik, csak a „valóságos képnek" megfelelő élek maradnak. A nehézséget itt az jelenti, hogy eldöntsük, mi tartozik a „valódi" képhez, és mi a zaj. A szokásos kritérium az, hogy egy zajos részlet általában elszigetelten, mindentől távol jelenik meg, míg a valóságos élek általában több más élhez csatlakoznak. Összetett képeknél a feladat még nehezebbé válik.

Az első vázlat a számítógépes látás „hagyományos" iskolájánál szerepel, amely David Marr munkásságából indult ki. Hiányzik viszont a látás connection-elvű megközelítéséből, ahol a világos-sötét mintákat vonatkoztatják a látótérben levő tárgyakra, és nem apró lokális jegyekből állítják össze a nagyobb tárgyakat.

Egyes látórendszerekben az input adatokat az élkereső fázis előtt előfeldolgozásnak vetik alá, amelynek során kiküszöbölik a vizuális „zaj" egy részét. Ez általában az izolált, sötét területen világos vagy világos környezetben sötét pontok eltávolítását jelenti.

ELEMZŐ

Természetes
nyelv
feldolgozása

A természetesnyelv-feldolgozásban az elemző (parser) olyan program, amely egy adott *grammatika* ismeretében teszteli a mondatokat, hogy megfelelnek-e a szabályainak. Azokból a mondatokból, amelyek „kiállják a próbát" (elfogadható mondatszerkezettel rendelkeznek) továbblépve *elemzési fát* készít (lásd alább).

A grammatika, legalábbis az elemző szempontjából bizonyos szimbólumok legális összekapcsolódási szabályait leíró szabályhalmaz. Például egy inzultusleíró grammatika, amely pontosan megadja, hogyan nézhet ki egy „inzultus", lehet az alábbi:

1. Az <inzultus> egy <javasolt cselekvés> a TE szócska és egy <durva megnevezés>.
2. A <javasolt cselekvés> FORDULJ FEL vagy TŰNJ EL.
3. A <durva megnevezés> egy <melléknév> és MOCSKOS DISZNÓ vagy DAGADT ÁLLAT.
4. A <melléknév> ROHADT vagy HÜLYE.

Ebből ilyen példamondatok jönnek ki:

Fordulj fel, te rohadt, mocskos disznó;
Tűnj el, te hülye, dagadt állat.

Ennek a grammatikának az elemzője végignézi a mondatot, hogy a megfelelő alkotóelemek a megfelelő sorrendben kapcsolódva szabályos inzultust alkotnak-e.

Egyes elemzők a legáltalánosabb struktúrából (a mondatból) kiindulva egyre kisebb összetevőket elemeznek. Ezek a *felülről lefelé működő (top-down) elemzők*. A fenti grammatikában egy elemző a következőképpen „gondolkodhatna":

„Kiadnak-e az adott szavak egy <inzultus>-t?",

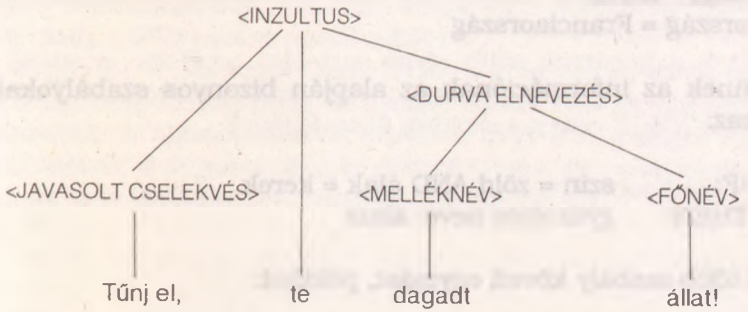
ami viszont így bontható le:

„Van-e köztük <javasolt cselekvés>"?
„Van-e köztük <durva megnevezés>"?

Ezért az elemző megnézi, hogy az egyes szavak szerepelnek-e a megadott melléknevek és durva megnevezések listáján.

Az *alulról-fölfelé működő (bottom-up)* elemző fordítva működik. „Hátrafelé" alkalmazza a szabályokat. Először megvizsgálja a szavakat, milyen mondatrészekhez tartoznak. Kettesével összetéve őket <javasolt cselekvés>-t és <durva megnevezés>-t talál. Mivel a TE szócska is köztük áll, érvényes inzultust alkotnak.

Az inzultus elemzése folyamán a program egy elemzési fát is készíthet. Ez az inzultus szintaktikai formáját ábrázolja. Például:



A mondat szintaktikai (a szavak sorrendje szerinti) elemzésének megvannak a maga korlátai. Sokkal nehezebb, de hatékonyabb, ha a jelentése alapján elemezzük a mondatot. Az 1970-es években egy egész sor elemzőt fejlesztettek ki, amelyek a mondatbeli szerepük alapján, olykor rendkívül rafinált módon elemezték a mondatokat. Például a fogalmi függőség elmélete (conceptual dependence theory) alapján, amely mindössze tucatnyi alapcselekvésből állítja össze az eseményeket és azok oksági láncolatait. Egy ilyen elemző az

Eleanor Thomasnak adta a játékmackót

mondat ábrázolásában azt is szerepeltetné, hogy ki kinek mit ad, ki kit szeret és így tovább.

ELŐRE LÁNCOLÁS

Szakértői
rendszerek

Az előre láncolás a szakértői rendszerek egyik következtetési módszere. A számítógép megvizsgálja a beadott adatokat:

szín = zöld
alak = kerek
ország = Franciaország

és ennek az információnak az alapján bizonyos szabályokat alkalmaz:

IF: szín = zöld AND alak = kerek
THEN: gyümölcs neve: alma

Ahol több szabály követi egymást, például:

A és B implikálja C-t
A és X implikálja D-t
A és Y implikálja F-et
C és D implikálja E-t
E és F implikálja Z-t

az előre láncolás az első alkalmazható szabály eredményét adja tovább. Így, ha például A, B, E és X igaz, akkor:

A és B implikálja C-t
A és X implikálja D-t
——————→ C és D implikálja E-t
——————→ E és F implikálja Z-t.

Ez természetesen rendkívül egyszerű példa az előre láncolás működésére. A valóságos szakértői rendszerekben akár több száz szabály is szerepelhet, amelyek az előre és hátra láncolás kombinációjával érnek el eredményt.

Lásd még: Hátra láncolás, Produkciós szabály

EMBERI TÉNYEZŐ KUTATÁSA

Általános számítógépes kifejezés

Sok olyan feladat, amelyben az MI a jövőben az emberek segítségére lesz, a számítógép és a felhasználó között az eddiginél sokkalta bensőségesebb kommunikációt követel meg. Milyen is lesz azonban egy ilyen interfész (találkozási felület)? Hogyan fog működni? Milyen fajta dialógusra lesz szükségük a számítógépet használó embereknek?

Ezek a kérdések több olyan kutatási területre vezetnek el, amelyeket az „emberi tényező", vagy „ember-számítógép együttműködés" (HCI) néven szoktunk összefoglalni.

Lásd még: Kooperatív rendszer

ERŐS TUDÁS/ GYENGE TUDÁS

Általános
MI-kifejezés

Az erős tudás speciális, magas szintű tudás az adott tartományról. A sakkban például erős tudás a nagymester tudása. Ez stratégiákat is magában foglal, képes klasszikus törekvések és helyzetek felismerésére. Nem tartoznak hozzá nagyszámú állásra érvényes „ökölszabályok", viszont adott lépéssorozatok következményeinek a felismerése annál inkább.

A gyenge tudás ezzel szemben az a fajta tudás, amilyennel a kezdő sakkozó rendelkezik. A gyenge tudás heurisztikákra épül: ökölszabályok alkalmazásával dönti el, hogy a lehetséges lépések közül melyik a legjobb. Míg a gyenge tudás gyorsan megszerezhető, az erős tudás elsajátításához hosszú idő kell.

ESET-GRAMMATIKA

Természetes
nyelv
feldolgozása

Az esetgrammatika olyan módszer, amellyel a mondatot meghatározó fogalmakat ábrázolhatjuk. Az „eset" szót itt nem a hagyományos értelemben használjuk, ahogy például a latin nyelvtanban szerepel, utalva a szó felszíni alakjára, hogy mondjuk alanyesetben, birtokos esetben stb. áll-e. A természetesnyelv-feldolgozás kontextusában az eset ennél még konkrétabban megjelöli, hogy az adott szó milyen szerepet játszik a mondatban. Ily módon a közönséges esetek között ilyesfajta fogalmak szerepelnek:

Cselekvő:	ami a cselekvést létrehozza;
Tárgy:	amire a cselekvés irányul;
Eszköz:	amit a cselekvő használ;
Forrás és cél:	ahonnan, vagy ahova a tárgy elmozdul.

Hogy a számítógép közelebb juthasson a mondat konkrét jelentéséhez, a mondat szavait az eset fogalmára alapozott alapuló keret jellegű struktúra nyílásaiba helyezhetjük..

Vegyük például az alábbi mondatot:

Eleanor Lego-kockákból házat épített.

Ennek az alábbi keret felelhet meg:

Ige:	épített;
Tárgy:	házat:
Cselekvő:	Eleanor;
Eszköz:	Lego-kockákból.

Lásd még: Grammatika, Természetes nyelv feldolgozása

ESPRIT

Finanszírozás

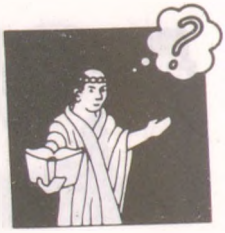
Az ESPRIT a Közös Piac által finanszírozott European Strategic Research Programme in Information Technology (Európai Stratégiai Kutatási Program az Információ Technikában) rövidítése. Kísérleti fázisa 1983-ban, fő programja 1984-ben, nyomkövető programja 1987-ben kezdődött. Az ESPRIT elindítása részben válasz a japán Ötödik Generációs Projekt Kezdeményezésre, amely az 1990-es évekre jobban felhasználható komputerek gyártását tűzte ki célul. Még általánosabban, az ESPRIT kísérlet az európai információtechnológiai erőfeszítések összehangolására.

Az ESPRIT-program többféle kutatást ölel fel. A program fejlett információfeldolgozási része egy sor mesterséges intelligenciakutatási projektet is magában foglal, mint például szakértői rendszerek készítése, tudásreprezentációs eszközök, egy természetes nyelvi kérdező, párhuzamos architektúrák és nyelvek, tudásalapú vezetési rendszerek és nem monoton következtetési technikák projektjei.

Az ESPRIT után 1987-ben az ESPRIT-II következett, amely tudásalapú tervezési technikák alkalmazását, a tudástervezés, fejlett rendszerarchitektúrák és a robottechnológia témáit is felölelte. Ezt egy újabb, ESPRIT-III elnevezésű program követte, amely 1994-ig tart, és 570 millió font költségvetéssel rendelkezik.

Lásd még: Ötödik generációs projektek

EXPLICIT ÉS IMPLICIT TUDÁS



Elmélet/
Filozófia

Explicit tudás az, amit kimondunk, míg az implicit vagy beleértett tudást nem fogalmazzuk meg, csak feltételezzük, hogy a felhasználó rendelkezik vele. Lehetséges, hogy az implicit tudás nem fogalmazható meg semmilyen formális módon. Tudásbázis megtervezésénél lényeges, hogy eldöntsük, a tudás mely része legyen explicit és mely része implicit, illetve mi az, aminek a kész rendszerben kell explicitté válnia. Ennek feltétele, hogy jó modellel rendelkezzünk a szükséges tudásról, illetve arról, hogy hogyan kell azzal dolgozni. Ezt nem könnyű elérni, mivel az explicit tudás eleme mögött nagy mennyiségű, a társadalom minden tagja által elsajátított implicit tudás rejlik.

Például valaki explicite azt mondja nekünk, hogy a tőkehal mélytengeri hal. A legtöbb ember még sohasem látott tőkehalat, ezért ez a tudás náluk egyszerű formális szabállyá válik. Ám példákból nagy általánosságban megtanultuk, hogy mi az a hal, amikor sok esetben nyálkás, hideg, különféle alakú és méretű élőlényekre mutatva azt mondták nekünk: "Ez egy hal." Soha nem tanultuk meg a hal formális definícióját. Noha taxonomikusan pontosan meg lehet adni a hal definícióját, a hétköznapi életben ezt nem használjuk. Maga a taxonomikus definíció is olyan tárgyak osztályozásán alapul, mint például „állat", amelyek maguk nem definiált, hanem implicit tudást képviselnek.

Egy tudásalapú rendszerben az explicit tudás vagy úgy jelenik meg mint az adatbázis egy adata, vagy mint egy program explicit eljárása; például a gravitációra vonatkozó tudásunkat a rakéta röppályájának kiszámítására szolgáló aritmetikai szabályokkal is megfogalmazhatjuk. Az implicit tudást formális értelemben nem is visszük be, ám a rendszer csak úgy működhet, ha mégis rendelkezik vele. Például implicit tudás húzódhat meg a program következtetési technikáiban, sokszor a következtetés alapfelte-

véseként. Például a rendszer feltételezheti, hogy minden tengerben élő állat hal, hacsak másként nem határozzuk meg. De megjelenhet a *zártvilág-feltevés* alakjában is.

Az implicit tudás leggyakrabban magában a felhasználóban rejlik. Nagy mennyiségű implicit tudásra van szükségük még látszólag egyszerű kijelentések megértéséhez is. Ezért van az, hogy sokkal könnyebb tudásbázist készíteni szakértők, mint laikusok számára, hiszen náluk jóval bővebb implicit tudásbázisra lehet támaszkodni.

A hétköznapi realitásról szóló tudás nagy része is implicit tudás. Amellett, hogy az implicit tudást konzisztens módon be kell építeni, a tudásbázis tervezőinek először is meg kell határozniuk, hogy az miben áll. A legagyafúrtabb program sem képes kihámozni azt a tudást, amit a felhasználó nem ad meg alkalmas definícióval vagy szabállyal, így tehát a programozó feladata az implicit tudás kódolása. Egy igazán intelligens szakértői rendszer képes lehetne arra, hogy maga is kivonjon implicit tudást, ám a jelenlegi szakértői rendszerek még nem tartanak itt.

Lásd még: Tudástervezés, Zártvilág-feltevés

ÉLKERESÉS

Látás

A számítógépes látásban ezzel az eljárásával egy képen a világosság-sötétség éles változásait keressük, amelyeket általában a képen látható tárgyak élei (szélei) okoznak. A valóságos világban azonban viszonylag ritkán állnak éles kontúrú fekete tárgyak fehér háttér előtt, ezért a képeket fel is kell dolgozni, hogy éleket lehessen kivonni belőlük. Ha például egy portrét nézünk, a kontrasztot addig fokozzuk, amíg Warhol-jellegű karikatúrát nem kapunk sok-sok kontrasztos éllel, gyakran csak a fekete és fehér marad meg belőle (*bináris kép*).

A módszer tehát az, hogy az egész képen keresztül megnézzük a fényességváltozásokat, és zérómetszéseket keresünk (azaz olyan helyeket, ahol a fényességváltozás megvátozása pozitívból negatívba megy át, vagyis ahol maga a fényesség változása a legnagyobb ütemű, tehát jó eséllyel találunk „élet").

Az retina neuronhálózata már az állatoknál is hasonló elven alapuló élkeresést végez az alakfelismerés korai fázisában.

Lásd még: Elemi vázlat

ÉPÍTŐKOCKA-VILÁG

Játékok
és játék-
tartományok

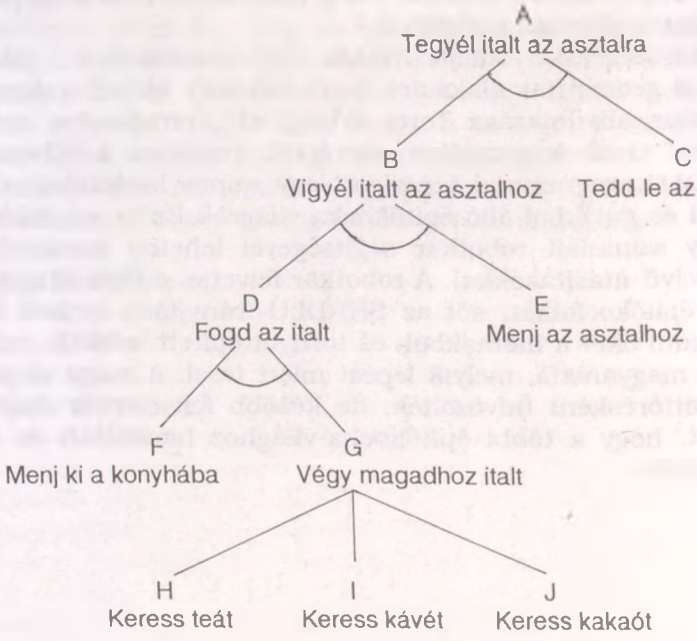
Az építőkocka-világ olyan egyszerűsített világ, amelyen MI-programokat és -módszereket szoktak tesztelni: maga is egyfajta „játék-tartomány". Eredetileg az építőkocka-világ egy lapos asztalon egymásra helyezett építőkockákból állt. Ily módon ezt a világot nagyon egyszerűen le lehet írni azzal, hogy melyik kocka melyiken helyezkedik el. Ez a logikai kijelentésekkel operáló algoritmusok egyszerű „próbapadja". Ebben egy olyan rendszer kiindulópontját látták, amely képes „gondolkodni" a világ összeállításáról, továbbfejlesztett változata pedig robotok és számítógépes látórendszerek alapjául szolgálhat.

A későbbiekben kifejlesztették egy általánosított, többféle szabályos geometriai alakzatot is tartalmazó térbeli változatát. Egyik híres alkalmazása Terry Winograd „Természetes nyelvek megértése" című dolgozatában szerepelt, amelyben a felhasználó az SHRDLU programmal társalgott egy színes kockákból, téglatestekből és gúlákból álló építőkocka-világról. Ezt a szimulált világot egy szimulált robotkar segítségével lehetett manipulálni, angol nyelvű utasításokkal. A robotkar felvette, odébb vitte és letette az építőkockákat, sőt az SHRDLU irányítása mellett ki is tudta szedni őket a halmokból, és tornyot épített belőlük, miközben el is magyarázta, melyik lépést miért teszi. A maga idejében jelentős áttörésként üdvözölték, de később felismerték alapvető hátrányát, hogy a többi építőkocka-világhoz hasonlóan ez sem elég általános.

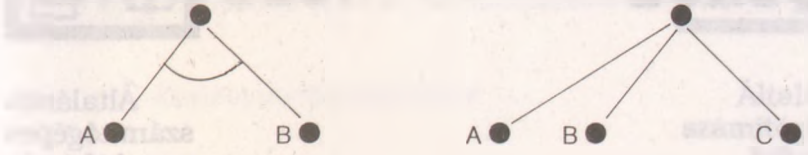
ÉS–VAGY FA

Keresés

Az ÉS-VAGY fa egy probléma megoldási fázisainak és közbenső állapotainak ábrázolására szolgál. Egy ilyen fa minden csúcsa, azaz elágazási pontja a teljes probléma valamely részproblémájának felel meg. Vegyük például azt a problémát, hogyan kell egy robotot utasítani, hogy a konyhában fogjon meg egy csésze forró italt, és vigye ki az ebédlőasztalra. A feladatot több részproblémára lehet bontani, a keresési fa valami ilyesmi leheti:



Ebben a reprezentációban az A csúcs ÉS-kiterjesztés: az A elvégzése előtt B-t ÉS C-t is el kell végezni. A G csúcs viszont VAGY-kiterjesztés: itt a H, I és J részproblémák közül egy is elég. Az ÉS-csúcsok és VAGY-csúcsok szokásos ábrázolása:



(a) ÉS-csúcs (b) VAGY-csúcs

Az ÉS-VAGY fákon keresthetünk szintben vagy mélységben indulva, és a megszokott módon heurisztikák alapján.

Egy játékban a nyerő lépés megkeresését is ábrázolhatjuk ÉS-VAGY fával. A fa csúcsai a tábla állásainak felelnek meg, így a gyökércsúcs a kezdőállásnak, a fa levelei pedig azoknak az állásoknak, amelyekben a játék véget ért. Az ÉS-csúcsok az ellenfél, a VAGY-csúcsok pedig a játékos lépéseit jelentik. Még részletesebb magyarázatot találunk a *Játékok fája* címszó alatt.

Lásd még: Alfa-béta nyírás, Keresés

FEJLESZTŐI ESZKÖZ

Általános számítógépes kifejezés

A fejlesztői eszköz olyan szoftver, amelynek az a feladata, hogy megkönnyítse MI-rendszerek kifejlesztését. A fejlesztői eszközöket fejlesztői környezetekbe összeállítva árulják.

A leggyakoribb fejlesztői eszközök az alábbiak:

1. *Programnyelvek.* A Prolog, Lisp, a C valamilyen változata, és így tovább. Szükséges hozzá egy compiler vagy interpreter típusú fordítóprogram, amely futtatható programot állít elő.
2. *Szerkesztői eszközök.* A programozó ezek segítségével megírhatja, átalakíthatja a programjait, részleteket törölhet belőlük. A színvonalasabb fejlesztői környezetek interaktív hibajavító eszközöket is tartalmaznak, amelyek lehetővé teszik a program listázását a képernyő egyik részén, és a fordítás vezérlését a másikon.
3. *Tudás-reprezentációs nyelv.* Ennek révén a programozó képes az adott területre vonatkozó tudást szabályok, vagy valamilyen más formalizmus segítségével bevinni a számítógépbe.

Lásd még: Gyors prototípus, Kísérleti rendszer

FEJLESZTŐI KÖRNYEZET

Általános számítógépes kifejezés

Egy MI fejlesztői környezet olyan fejlesztői eszközöket tartalmaz, amelyek segítségével a programozó MI alkalmazásokat tud készíteni. Vannak köztük hardver-specifikus eszközök: mások bizonyos típusú elterjedt gépeken, például PC-ken is futnak. Az alábbiak tipikus példái a fejlesztői környezeteknek:

1. *Prototípuskészítő-eszközök.* Segítségükkel kísérleteket végezhetünk valamilyen tudásreprezentációval vagy felhasználói felülettel, mielőtt végleg elköteleznénk magunkat egy költséges, professzionálisan megtervezett termék mellett. A Prolog ma is megbízható prototípuskészítő eszköz: egy egyszerű Prolog viszonylag olcsón kapható, szerkesztő és hibajavító eszközöket tartalmaz, és módszert ad a kód átírására C nyelvbe.
2. *Szakértői-rendszer shellek.* Válogatott programozási eszközöket – tudásreprezentációs nyelvet, szerkesztőt, következtetőgépet – szolgáltatnak szakértői rendszerek készítéséhez.
3. *Eszközkészletek.* Sokféle eszközt tartalmaznak nagyobb, jobban kidolgozott szakértői rendszerek kifejlesztéséhez. Az eszközkészletek rendszerint integrált programozási technikák széles skáláját tartalmazzák, többek közt logikai programozást, objektum-orientált programozást, produkciós szabályokat, és hagyományos programnyelveket is. A következtetőgép alkalmas lehet arra, hogy hálózaton keresztül más gépeken tárolt nagy adatbázisokból is felhasználjon információkat.

Lásd még: Gyors prototípus, Szakértői rendszer shell

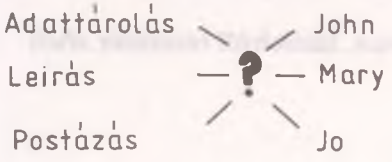
FELADATKIOSZTÁSI PROBLÉMA

Játékok
és játék-
tartományok

Számos tervező program tesztproblémája. Több ember különböző feladatokat eltérő gyorsasággal hajt végre. Milyen feladatkiosztással lehet a teljes munkát a leggyorsabban elvégezni? Itt kombinatorikai robbanás lép fel, mert N feladat szétosztása N emberre N faktoriális módon lehetséges. Egy életszerű, 50 feladatos problémánál a kiosztások száma 10^{64} nagyságrendű. Nem prakti-

	John	Mary	Jo
Adattárolás	1	1·5	2
Leírás	2·5	4	2
Postázás	1·5	3	4

Relatív hatékonyság



tikus mindet végigpróbálni, tehát ez jó modell a keresési stratégiákra. A probléma erősen összefüggő, mert egy feladat kiosztása egy emberre erősen korlátozza, amit a többiek vállalhatnak. Ezért a problémát nem könnyű részfeladatokra bontani. *Az utazó ügynök problémájához* hasonlóan itt is sikerrel próbálkoztak neuronhálózatokkal, amelyek jól alkalmazhatók erősen összefüggő problémákra.

Lásd még: Kombinatorikai robbanás

FELHALMOZOTT TUDÁS

Elmélet/
Filozófia

A felhalmozott tudás működő tudás, amelybe sokkal mélyebb ismeretek is beágyazódnak, és együtt bizonyos „ökölszabályokat" szolgáltatnak. Ezt a fajta tudást sajátítjuk el az általános iskolában. Például biológiaórán a tanár elmeséli, hogy a fehér egerek mindig fehér utódokat hoznak világra. Ez ugyan jelentős mértékben egyszerűsített kivonata az ő ismereteinek, ám az egértenyésztők többsége számára elég jó ökölszabály. Ezzel szemben a tanár számos biológiai és biokémiai ismeret birtokában azt is tudja, hogy miért fehérek a fehér egerek utódai. Ezt a fajta tudást jóval nehezebb megszerezni, mert sokkal nagyobb a mennyisége, viszont sok más szituációra is alkalmazható, tehát sokkal jobban fel lehet használni.

A tudásalapú rendszerekben sokszor a felhalmozott tudás a legmegfelelőbb, mert nagymértékben képes csökkenteni a rendszerben tárolt információ, illetve az abból levonandó következtetések mennyiségét.

Ehhez kapcsolódik a *felszínes* és *mély tudás* fogalma, nagyjából az „ökölszabály" illetve „alapvető okok megértése" értelmében. Nem világos, mennyire hasznos a mély tudás a mindennapi életben. Tudósokkal végzett vizsgálatok szerint ők sokszor olyan kérdésben is felszínes tudásra támaszkodnak, amelyben mély tudással rendelkeznek, mert az előbbi is megfelelően működik. Például a XVIII. században az egérfélék családjának tárgyalása a vérben levő csírákról, vagy a spermában megbúvó immanens egérről szólna, a DNS-ről említést sem tenne. Tehát mindhárom leírás megmagyarázza, miért vannak a fehér egereknek fehér utódaik. Ezért van némi vita afelől, hogy a mély tudás lényegesen alkalmasabb-e a világ leírására, mint a felszínes tudás.

Lásd még: Explicit és implicit tudás

FOLYAMAT

Általános
számítógépes
kifejezés

Számítógépes terminológiában a folyamat műveletek egyetlen logikai láncolata. Például folyamat lehet egy alkalom, amelyet egy terminálnál vagy mikroszámítógépnél tőltünk: bejelentkezünk, betöltünk egy file-t a memóriába, lefuttatunk egy programot, sorban egymás után. A különböző folyamatok csak jól definiált csatornákon keresztül érintkezhetnek egymással, általában meghatározott struktúrákban továbbítanak üzeneteket egymásnak.

Lásd még: Együttfutás

FORGATÓKÖNYV

Természetes
nyelv
feldolgozása

A forgatókönyv valamilyen forrásból, például egy újságcikkből nyert információk megszervezésének és ábrázolásának eszköze. A forgatókönyv alapjait a pszichológiában kell keresni. A pszichológusok kimutatták, hogy az ember az új szituációkat a korábbi tapasztalatok fényében értelmezi. Amikor például egy szupermarketbe megyünk, van elképzelésünk, hogy mit lehet ott találni: a bejáratnál bevásárlókocsikat, áruval megrakott polcokat, pénztárakat és így tovább. Hasonlóképpen bizonyos eseménysorokra is fel lehetünk készülve: hogy először megrakjuk a bevásárlókocsinkat, majd sorba állunk a pénztárnál. A forgatókönyvek ezt a gondolatot alkalmazzák az információ értelmezésére. A forgatókönyv tehát megszokott cselekvések sémája.

Képzeljünk el, hogy egy számítógép egy újságcikkből valamilyen tárgyra vonatkozó információt keres ki. A gép a természetesnyelv-feldolgozás módszereivel (lásd *Természetes nyelv feldolgozása* és *Elemző*) komponensekre bontja a mondatot, majd egy forgatókönyv kitöltésével valamifajta jelentést von ki az újságcikkből. Például egy élelmiszer-mérgezéssel foglalkozó cikkben valószínűleg talál információt arra nézve, hogy kit, hol és hogyan ért mérgezés. A forgatókönyvben tehát például az alábbi nyílásokat találhatjuk:

Kik kaptak mérgezést:	Mr. Black és 16 társa
Hol:	Hamm gyorsbüfé
Hogyan:	nem tudni
Milyen tüneteik vannak:	evés után négy órával heves gyomorgörcs
Teendő:	büféből kivezetni a kismalacot.

Lásd még: Keret

FUTTATÓ RENDSZER

Általános számítógépes kifejezés

Egy futtató rendszer azokból az alapvető szoftver- és hardverelemekből áll, amelyek egy adott program futtatásához szükségesek. (Szemben a fejlesztői környezetekkel, amelyek emellett szerkesztő és hibajavító eszközöket, compilert és egyéb, a program megírásához és lefuttatásához szükséges összetevőket tartalmaznak.)

Annak a felhasználónak, aki nem megváltoztatni, továbbfejleszteni, csak lefuttatni kívánja a megvásárolt szoftvert, a futtató rendszer az olcsóbb megoldás. A szoftver nem csupán önmagában lesz kevésbé költséges, de rendszerint olcsóbb gépen, például közönséges PC-n is futtatható. A teljes fejlesztői környezethez ezzel szemben technikai munkaállomásra lehet szükség.

GENETIKUS ALGORITMUS

Elmélet/
Filozófia

A genetikus algoritmusok területe egyelőre főként kutatási téma, kevés gyakorlati alkalmazással. Ezek egyfajta tanuló algoritmusok, az empirikus algoritmusok speciális változatai, amelyek próbálkozás útján tanulnak. Gyakran használják őket produkciós szabályokon alapuló rendszerekben. Minden szabályhoz valamilyen valószínűséget társítanak. A program minden ciklusban összegyűjti azokat a szabályokat, amelyek bal oldala az adott feltételekhez illeszkedik. Ezután közülük *egyet* alkalmaz. Hogy melyiket, az tisztán a véletlenen múlik, de az esélyeket úgy torzítja, hogy a magasabb valószínűségekkel társított szabályokra gyakrabban kerüljön sor. Ha egy szabály alkalmazásának az eredménye megfelel az ideális megoldásnak, akkor megnő a valószínűsége, hogy legközelebb azt alkalmazza. Így néhány ciklus után szinte kizárólag azok a szabályok kerülnek alkalmazásra, amelyek jó válaszokat adnak.

A genetikus algoritmus John Holland által adott eredeti megfogalmazása azt is lehetővé teszi, hogy a komputer új szabályokat generáljon. Ezek a régieknek pontatlan, „mutációkat" tartalmazó másolatai. Ha valamelyikük nagyon jól sikerül, akkor idővel magas valószínűségre tesz szert, és így átveszi az elsőbbséget a régi szabályok felett. A számítógép tehát még akkor is megtanulhat megoldani egy problémát, ha eredetileg nem állnak rendelkezésére erre alkalmas szabályok. Van egy sémája (körvonalazott terve) arról, hogy mik a megengedett szabályok. A legtöbb szabály régiek kombinálásával keletkezik (például úgy, hogy különböző szabályokból kivett darabokat illeszt össze); másoknak egyes részei véletlenszerűen, mintegy „mutációval" megváltoznak. Ennek révén a genetikus algoritmusok meg tudnak oldani a hegymászó módszerekkel megközelíthetetlen problémákat is, mert ezekkel az új kombinációkkal a probléma-tér új területeire

tudnak „ugrani", átrepülnek a táj szakadékai fölött, ahol egy földön járó hegymászó algoritmus nem tudna továbbjutni.

Érdekes ötlet, amelynek a kutatása jelenleg is folyik, hogy neuronhálózatokkal valósítsanak meg genetikus algoritmusokat (és fordítva, hogy genetikus algoritmusok segítségével optimalizáljanak neuronhálózatokat).

Lásd még: Neuronhálózat, Automatikus programozás, Tanulás

GÉPI FORDÍTÁS

Általános
MI-kifejezés

A számítógépes fordítás annyit jelent, hogy egy „forrásnyelvről" egy másik, úgynevezett „célnyelvre" való fordítás feladatát legalábbis részben számítógép segítségével végezzük el. Azonban minden ma kapható rendszernek szüksége van emberi közreműködésre is, amely a program outputját idiomatikus nyelvre alakítja.

A legkorábbi fordító rendszerek a következőképpen működtek. Először egyszerű grammatikai elemzésnek vetették alá a szöveget, majd kikeresték a szavakat és kifejezéseket a szótárból, a célnyelvi megfelelőikkel együtt. Ha többértelműség lépett fel, a szó környezete alapján választottak, de igen egyszerű módon.

A 60-as évek végére ezek az egyszerű módszerek elégtelennek bizonyultak, a kutatás tehát a fordítás közvetettebb módszerei felé fordult, ahol nagyobb hangsúlyt kapott a kontextus és a szemantika. Ezzel egyidejűleg dolgozta ki Chomsky a transzformációs grammatikák elméletét, amelyet azonnal fel is lehetett használni a mondatok egyfajta standard ábrázolásra való fordításában. A nyelv többértelműsége és bizonytalansága azonban továbbra is problémát jelentett, nem voltak jó eredmények.

Ma ismét megnőtt az érdeklődés a gépi fordítás iránt Európában, különösen a Közös Piac országaiban. Két projektet is indítottak, SYSTRAN illetve EUROTRA néven. Alább egy SYSTRAN fordításra adunk példát. Figyeljük meg: a számítógép alapvető félreértése (különböző nemzetiségű embereket feltételez, ahol pedig különböző nyelvekről van szó) arról tanúskodik, hogy a szöveg valódi jelentéséből semmit nem értett meg.

Francia szöveg: L'utilisation de FSSRS est basée sur un système de menus hierarchisés d'un usage asséz aisé. L'utilisateur a la possibilité de travailler en Français, Anglais ou Allemand. Ce manuel de

consultation et le manuel 'Contenu de la base' ne sont disponible qu'en Français et en Anglais; cependant si la demande existe ils pourraient être traduits en Allemand.

Gépi *fordítás:* Az FSSRS használata hierarchikus töredékek részleteinek rendszerén alapul igen könnyen használható. A felhasználó lehetősége franciaként, angolként vagy németként működni. Ez a kézikönyv és „Az alap tartalma" c. könyv csak franciaként és angolként áll rendelkezésre; míg, ha a kérés létezik, németre fordíthatók.

Az EUROTRA egy közbülső európai „ábrázolás" elképzelésén alapul, első lépésben minden forrásszöveget erre fordítanának, és ebből készülnének a célnyelvi fordítások. A célkitűzés merész, de egyelőre kevés előrehaladás történt.

Lásd még: ESPRIT, Természetes nyelv feldolgozása

GRAMMATIKA

Természetes
nyelv
feldolgozása

A „grammatika" a természetesnyelv-feldolgozásban szerepel, egy olyan szabályhalmazt jelöl, amely megadja, hogy a szavak hogyan kapcsolódnak össze mondatokká.

A grammatika egyik nagyon egyszerű típusa a *környezetfüggetlen grammatika.* Ez a Noam Chomsky nyelvész által definiált a négy grammatika-alaptípus egyike; egyben az egyik legnépszerűbb. Egyszerű természetes nyelvi alkalmazásoknál használják, mert rendkívül könnyen programozható. Ebben a részben a környezetfüggetlen nyelvtan alapján vezetjük be azt a terminológiát, amelyet általában számítógépes grammatikák leírására használnak.

Egy környezetfüggetlen grammatikát *újraírási (vagy produkciós) szabályokkal* lehet megadni. Egy újraírási szabály azt írja le a számítógép számára, hogy az alapvető mondatrészeket hogyan lehet összerakni egyszerűbb összetevőkből. Ha például azt akarjuk megmondani, hogy mit értünk „mondat" alatt, a mondat struktúráját mondjuk egy igei és egy névszói csoportból állítjuk össze:

< MONDAT > ⟶ < FŐNÉVI CSOPORT > < IGEI CSOPORT >

A névszói csoportot azután ismét újra lehet definiálni, például determináns és névszó vagy determináns, melléknév és névszó egymásutánjaként:

<FŐNÉVI CSOPORT > ⟶ < DET > <FŐNÉV >
<FŐNÉVI CSOPORT > ⟶
< DET > < MELLÉKNÉV >< FŐNÉV >

Végül a lexikon adja meg a számítógépnek, hogy melyek az elfogadható angol szavak, és milyen mondatrészhez tartoznak.

Alább egy teljes grammatikát adunk meg néhány igen egyszerű mondathoz. Ez a grammatika például előállítja az

Eleanor Thomasnak adta a szendvicset
Thomas Eleanornak rúgta a narancslét

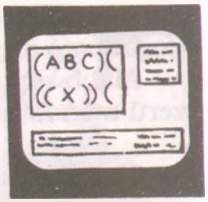
mondatokat, sőt,

A szendvics Thomasnak adta Eleanort

mondatot is, amely szintaktikailag ugyan helyes, de szemantikailag teljesen értelmetlen.

<MONDAT> ⟶ <FŐNÉVI CSOPORT> <IGEI CSOPORT>
<FŐNÉVI CSOPORT> ⟶ <DET> <FŐNÉV>
<FŐNÉVI CSOPORT> ⟶ <FŐNÉV>
<IGEI CSOPORT> ⟶
<IGE> <FŐNÉVI CSOPORT> <RAGOS CSOPORT>
<RAGOS CSOPORT> ⟶ <FŐNÉV> <RAG>
<IGE>⟶ adta
<IGE> ⟶ rúgta
<FŐNÉV> ⟶ Thomas
<FŐNÉV> ⟶ Eleanor
<FŐNÉV> ⟶ narancslé
<FŐNÉV> ⟶
<RAG> ⟶ -nak

GYORS PROTOTÍPUS



Programozási technikák

Egy szakértői rendszer megtervezése rendkívül bonyolult folyamat. Számos kérdésre kell választ keresni: mi mindent kell „tudnia" a rendszernek, hogyan nézzen ki a felhasználói felület, és így tovább. Ezek a problémák a *tudástervezés* körébe tartoznak.

A gyors prototípus olyan sebtében összeállított rendszer, amelyből a kliens benyomást nyerhet arról, hogy milyen lesz a végső rendszer. A rendszer ugyanakkor nem rendelkezik teljes tudásbázissal, és lehet, hogy nem használja ki optimálisan a rendelkezésre álló hardvert. Emiatt előfordulhat, hogy a prototípus működése lassú, memóriafelhasználása nem hatékony, és tudása a szóban forgó tartományról sekélyes. Egy finoman hangolt szakértői rendszer elkészítése olyan sokáig tart, hogy érdemes kétszeresen meggyőződni arról, hogy jó úton járunk, mielőtt túl messzire jutnánk.

Lásd még: Szakértői rendszer

HAMMING-HÁLÓ

Neuron-
hálózatok

Egyfajta neuronhálózat, amely a megengedett kapcsolatok természetében tér el a *Hopfield-hálótól*. Ez a hálózat három réteg neuront tartalmaz. A Hopfield-hálóval ellentétben, ahol bármelyik neuron összekapcsolható bármelyik másikkal, a Hamming-háló szigorúan „egyirányú", az input elemek csak a feldolgozó egységekhez kapcsolódnak, azok pedig vagy további feldolgozó egységekkel, vagy az output elemekkel vannak összekötve.

A Hamming-hálónak lényegesen nagyobb a memóriája, mint a Hopfield-hálónak, és könnyebben konvergál stabil viselkedési mintához. Emiatt az alakfelismerési feladatoknál, ahol a neuronhálózatok általában kiváló eredményt adnak, ez további előrelépést jelent.

Lásd még: Hopfield-háló, Neuronhálózatok

HÁTRA LÁNCOLÁS

Szakértői rendszerek

A hátra láncolás módszerét számos MI-rendszer alkalmazza a problémamegoldásban. A módszer hasonlít a „barkochba" elvére. A számítógép valamilyen feltevésből indul ki. Ezután kérdéseket tesz fel, hogy eldöntse, helyes-e ez a feltevés. Például a következőképpen:

Hipotézis:	Azt hiszem, bárányhimlőd van.
Tehát ezt kérdezem:	Vannak piros kiütéseid középen fehér hólyagokkal?
	Kisgyermek, netán csecsemő vagy?
	Voltak valaha hasonló tüneteid?

Az MI-ben a hipotézist sokszor „célnak", az igazolásához szükséges tényeket pedig „feltételeknek" hívják. A hátra láncolás célorientált módszer: a számítógép először a célt állítja fel, és aztán tények és szabályok segítségével próbálja eldönteni, hogy az adott cél elérhető-e.

A hátra láncolás ellentéte az előre láncolás. Ezt „adatvezérelt" módszernek mondjuk, mert a rendszer először a rendelkezésére álló adatokat vizsgálja meg, és csak azután, hogy milyen következtetéseket lehet levonni belőlük.

A hátra láncolással működő programok általában feltételes szabályok sorozatát tartalmazzák, mint például:

A ha B és C	emlős, ha rágcsáló vagy tehén
B ha D és E	rágcsáló, ha kicsi bundás állat és makog
C ha X és Y	tehén, ha növényevő és bőg
X ha R és S	növényevő, ha soha nem eszik húst és soha nem eszik halat.

A hátra láncolás algoritmusa lehetővé teszi, hogy módszeresen mindegyik szabályt sorra vegyük.

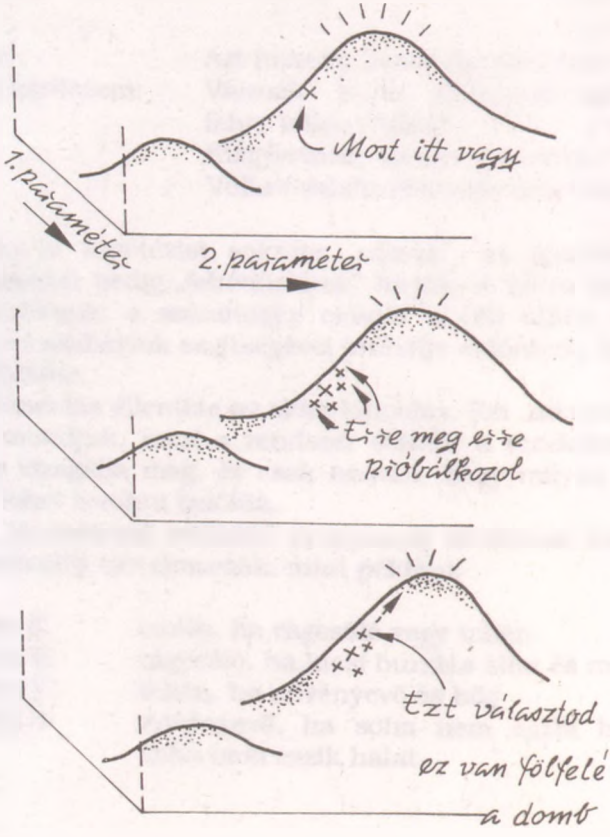
A felételes szabályokat általában egy előzménnyel (a feltétel) és következménnyel (mi történik, ha a feltételek teljesülnek) írjuk le. Ezzel e terminológiával a hátra láncolás azt jelenti, hogy sorba vesszük egy szabály előzményeit, és megpróbáljuk azonosítani őket más szabályok következményeivel. Az eljárás akkor ér véget, amikor a szabály előzményeit már nem lehet átdefiniálni.

Lásd még: Előre láncolás, Keresés, Szakértői rendszerek

HEGYMÁSZÁS

Általános
MI-kifejezés

Ezt a kifejezést problémamegoldó módszerekre használják. Legnagyobb népszerűségnek a 60-as években örvendett, de bizonyos alkalmazásokkal kapcsolatban még ma is sokszor említik.



A hegymászás algoritmusa úgy működik, hogy megnézi a probléma egyik aktuális megoldását, majd mindazokat a megoldásokat, amelyeket a probléma egy-egy paraméterének kis igazításával kaphat, és kiválasztja azt, amelyik legjobban illeszkedik az optimális megoldáshoz. A gyakorlatban ezt az illeszkedést valamilyen heurisztikával számoljuk ki, amely becslést ad arra, hogy ez a megoldás mennyiben hasonlít az ideális megoldásra. Sokszor könnyű megmondani, hogy egy konkrét kísérlet milyen távol esik a végső megoldástól, noha azt magát nagyon nehéz volna előállítani. Ily módon a program hegymászóként halad előre a „megoldás" gráfban.

A hegymászással csak *lokálisan* optimális megoldásra lehet jutni. Mindig a legközelebbi dombra kapaszkodunk fel. Ha a grafikonon nem ez az egyetlen csúcs, akkor lehet, hogy nem a legjobb megoldást kaptuk. Ezen genetikus algoritmusokkal és szimulált hűtéssel, vagyis statisztikai módszerekkel segíthetünk. A hegymászó algoritmusok csak akkor működnek, ha a „magasságot" megadó függvény folytonos, vagyis a felszínt nem tarkítják feneketlen szakadékok.

A hegy „magassága" persze számtalan dolgot jelenthet, például egy heurisztika értékét, amely a megoldástól való távolságot becsüli meg: más alkalmazásokban szokás ezt „számítási energiának", „célfüggvénynek" vagy „költségfüggvénynek" is nevezni.

HEURISZTIKA

Általános
MI-kifejezés

A heurisztika ökölszabályt, következtetésekben alkalmazott általános elvet jelent. Például klasszikus heurisztika: „az ember saját állítása szerint általában sokkal gyorsabban el tud végezni valamit, mint a valóságban". Vagy azoknak, akik sohasem tudnak ellenállni a „bütykölés" kisértésének: „először mindig a legnyilvánvalóbb hibára kell gyanakodni". A heurisztika erősen eltér az algoritmustól, amely lépésenként végrehajtandó, pontos eredményt adó módszer. Amíg egy algoritmustól elvárjuk, hogy tévedhetetlen legyen, a heurisztikák általában helyesek, ám kedvezőtlen esetben tévútra is vihetnek.

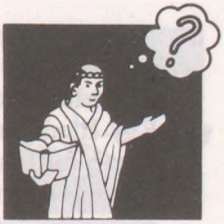
A heurisztikákat számos helyen alkalmazzák az MI-ben. Az MI-rendszerben sokszor előforduló IF ... THEN alakú szabályok (lásd *produkciós szabályok*) lényegében heurisztikák: csak egy speciális helyzetre alkalmazhatók, és inkább tanácsot adnak, mint határozott és tévedhetetlen utasítások sorozatát.

Heurisztikákat széles körűen alkalmaznak a keresés támogatására is. Nyilvánvaló például, hogy amikor a sakkban a következő lépést akarjuk megválasztani, nem volna praktikus vakon végignézni minden lehetséges lejátszást. Heurisztikákra van szükség, például: „ne tedd a királynőt olyan helyre, ahol az ellenfél le tudja venni", amelyek megadják a játék általános alapelveit. Lásd a *keresésről*, illetve *sakkról* szóló szócikkeket.

Végül gondoljunk a heurisztikák szerepére a tervezésben. Ha a matematikus egy algebrai kifejezést lát, rögtön tudja, hogyan lehetne egyszerűsíteni. A diák viszont „azt sem tudja, hol kezdjen hozzá". A heurisztikák, amelyek bizonyos felismerhető mintákra lehetséges válaszlépéseket sugallnak, segítségünkre lehetnek, hogy egy bonyolult problémában ki tudjunk igazodni.

Lásd még: Tudásbázis, Szakértői rendszer, Algoritmus, Keresés

HIEDELMEK



Elmélet/
Filozófia

Az MI jó része a tudással foglalkozik. Az emberi értelem, de az emberiség legjelentősebb vívmányai is nagy részben hiedelmekhez kötődnek. Az MI a tudás mellett a hiedelmek modellezésével is megpróbálkozik, mégpedig három okból. A hiedelmek kulcsszerepet játszanak a tudáselsajátítás bizonyos formáiban, különösen az *indukcióban*, ahol valamilyen kérdésről hiedelmeket, feltevéseket alkotunk, mielőtt teljes tudásra tehetnénk szert. Ehhez kapcsolódik, hogy az ember–számítógép kommunikációban, különösen pedig a számítógéppel segített tanulásban milyen fontos, hogy a gép megértse az ember hiedelmeit. Ez viszont az emberi hiedelmek megértésének általános törekvéséhez kötődik.

A hiedelemrendszereket (mind az olyan hiedelemgyűjteményeket, mint például a buddhizmus, mind a hiedelmek bizonyos aspektusait modellező számítógépes rendszereket) egy sor jellegzetesség különbözteti meg a tudásrendszerektől.

1. Két hiedelemrendszer ugyanazokból a tényekből kiindulva egészen eltérő következtetésekre juthat. (Hogy ez milyen fontos a megértés és a tudás szempontjából, lásd a *Paradigma* címszót).
2. Sokszor olyan elvont fogalmakkal dolgoznak, amelyeket különböző rendszerek nem tudnak egymással megosztani. Például a filozófusokat mélyen megosztja az a kérdés, hogy a „gépi intelligencia" kifejezés jelent-e egyáltalán valamit. Aki nem hiszi, hogy létezik ilyesmi, azzal értelmetlen volna arról vitatkozni, hogy a számítógép rendelkezik-e vele. Ezek az absztrakt fogalmak gyakran a „jó" és „rossz" valamilyen szinonimáját tartalmazzák. A hiedelemrendszerek ebben szembenállnak a tudásrendszerekkel, amelyek egyazon a tárgyról szólva ugyanazt kell, hogy mondják.

3. A hiedelemrendszerek sokszor nem létező világokról szólnak („Egy ideális világban...").
4. A hiedelmek nem kötődnek direkt módon a tényekhez.
5. Egy hiedelem igazolásához a hitelesség és az érzelmek éppen olyan fontosak, mint a logika.
6. A hiedelemnek több köze van a vágyakhoz (konáció), mint az ésszerűséghez (kogníció).

Az utolsó pont miatt a hiedelmek összeütközésbe kerülnek az MI szinte minden más fejleményével. A hiedelem nem arról szól, hogy mi igaz, hanem hogy mit *akarunk* igaznak tudni. Ezt demonstrálta a PARRY nevű, paranoiás betegeket szimuláló program. Bármiről tárgyaltak vele, mindenből kényszeresen ugyanarra a témára (a maffiára) lyukadt ki, más „érzékeny" kérdéseket pedig gondosan elkerült. Még „hazugságra" is képes volt, hogy fenn tudja tartani hiedelemstruktúráját. Mindennek a következményeit általában félreértették (és ebben nem segített a durva kvázi-természetes nyelvi kommunikációs felület sem), mert az MI-kutatók többsége úgy gondolta, hogy a PARRY (az MI rendszerek túlnyomó többségéhez hasonlóan) a kogníció és nem a konáció modellezésére szolgált.

Lásd még: Explicit és implicit tudás

HIPERKOCKA

Hardver

A hiperkocka-architektúra egy párhuzamos feldolgozást végző rendszer processzorainak lehetséges összekapcsolási módszere. Matematikai értelemben a hiperkocka egy négydimenziós kocka, itt azonban a fogalom azt jelenti, hogy a processzorokat úgy kapcsolják össze, mintha egy négy- vagy többdimenziós kocka csúcsaiban lennének elhelyezve. Ezzel az architektúrával bizonyos fokú kompromisszum valósul meg kétféle igény között: egyrészt, hogy minden processzor képes legyen kommunikálni mindegyik másikkal; másrészt, hogy ne legyen se túl sok útvonal a processzorok között (amely kivitelezhetetlenül sok összeköttetést kívánna), se túl kevés (nehogy a processzoroknak idejük nagy részét a többiek üzeneteinek továbbításával kelljen tölteniük).

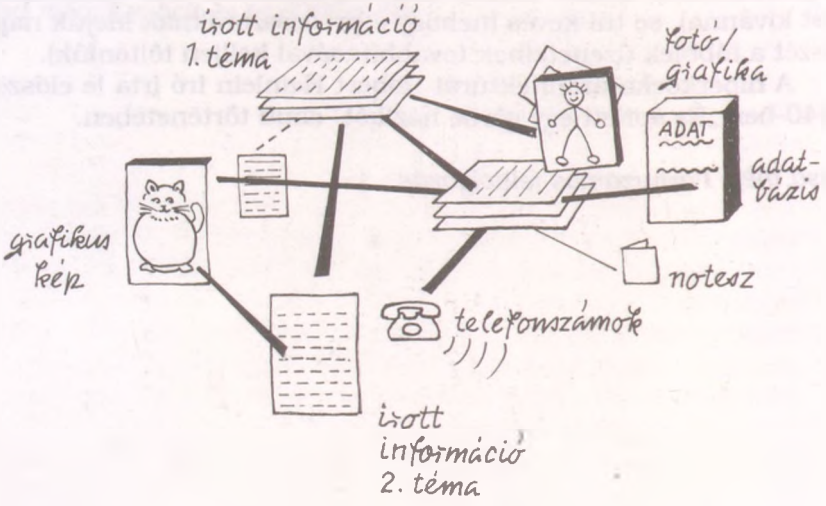
A hiperkocka-architektúrát Robert Heinlein író írta le először 1940-ben „És épített egy görbe házikót" című történetében.

Lásd még: Párhuzamos feldolgozás

HIPERMÉDIA

Általános
számítógépes
kifejezés

Hipermédiának az általános elnevezése annak az új stílusú információs rendszernek, amely nem szekvenciálisan képernyő- oldalanként jelzi ki az információt, hanem animált, interaktív módon. Egy ilyen rendszerben az információt tipikusan egy hálózat csomópontjaiban álló szöveg, grafika, alkalmazási programok stb. formájában képzelhetjük el. A hipermédia fő szempontja, hogy minél könnyebb legyen visszakeresni az információt: a hagyományos rendszerek a felhalmozott információra koncentrálnak.



1988-ban az Apple cég mutatta be az első hipermédia-csomagot HyperCard néven. Addig a hipermédia rendszerek nagyrészt oktatási intézményekre szorítkoztak. Az Apple megközelítése

a jó minőségű grafika készítésének lehetőségét, az objektumorientált programozási modellt és a könnyen használható programozási nyelveket hangsúlyozta. Ez utóbbi vonása lehetővé teszi, hogy az is tudjon hipermédia-alkalmazásokat írni, akinek nincs formális számítástudományi képzettsége.

Más hipermédia-rendszerekhez hasonlóan a HyperCard is sorba rakott kártyák alakjában mutatja be az információt, ahol maga a csomag is egy csúcsot képvisel a számítógépben tárolt információk hálózatában.

A kártyacsomag illusztrálására képzeljünk el egy számítógépesített noteszt. Bár a valóságban csak igen egyszerű adatbázis, a notesz teljesen élethűen jelenik meg a képernyőn: spirálos fedőlappal, a kezdőbetűknél kivágott regiszteres oldalakkal. Ha az egérrel rákattintunk egy kiválasztott betűre, a könyv a keresett lapon – vagy kártyánál – nyílik ki. Ha azután még egy kis telefonszimbólumra is rákattintunk, a gép fel is hívja a megadott telefonszámot.

Minden lap alján ikonok találhatók, amelyek, ha kiválasztottuk őket, adott oldalakhoz kapcsolhatnak bennünket. Ha rákattintunk a megfelelő ikonra, vissza- vagy előrelapozhatunk egy oldalt, de akár ki is ugorhatunk a noteszből egy másik alkalmazásba.

Hogy e jelenetek mögött mi is történik, azt egy kis program, a *forgatókönyv* határozza meg. Az alapgondolat az, hogy könnyen megírható forgatókönyvek révén (kis türelemmel) még a nem programozó is a hagyományos rendszereknél sokkal gazdagabb kapcsolatokat képes teremteni.

A hipermédia-technika még fejlődésének kezdeti stádiumában van. A jövő hipermédia-rendszerei valószínűleg szoros kapcsolatban lesznek a szakértői rendszerekkel: a mai hipermédia-rendszerekben még a felhasználóé a kezdeményezés, ő dönti el, mikor milyen információt szeretne látni. A jövőben itt beléphet egy szakértői rendszer, és kérdéseket tehet fel a felhasználónak, mielőtt további információkat jelenítene meg. Ilyesfajta „vegyes kezdeményezésű" rendszerek számos alkalmazásban szerepelhetnek. Például elképzelhetünk olyan képernyőn megjelenő katalógusokat, amelyek azon túl, hogy szövegek és képek segítségével leírják, mi minden kapható, ahol szükséges, ki is kérdezik a felhasználót, hogy kiderítsék, pontosan milyen termékre is van szüksége.

Lásd még: Emberi tényező kutatása

HOMLOKZAT

Általános számítógépes kifejezés

Egy rendszer homlokzatának nevezzük azt a részét, amely a felhasználó „felé néz", vagyis az inputot és outputot kezeli. Az MI-ben ez a program rendszerint egy adatbázis vagy szakértői rendszer, és *intelligens homlokzatnak* is szokás nevezni. Azt a homlokzatot, amely a felhasználótól akár angol nyelvű szövegeket is elfogad, és azokat alkalmas kódra tudja lefordítani, *természetes nyelvi homlokzatnak* nevezzük. Néha még a szakértői rendszer outputját is képes angolra fordítani. Az utóbbira példa a ROUNDSMAN orvosi tanácsadó rendszer, amely az orvosi szakirodalom adatbázisából vett tényeket angol nyelvű kijelentésekké szerkeszti össze. Akárcsak a ROUNDSMAN esetében, a homlokzat önmagában is nagy teljesítményű szakértői rendszer lehet.

A homlokzat sokszor a grafikus kijelzést is vezérli, és ennek segítségével a felhasználó bonyolult programokkal is együtt tud működni. Az Apple Macintosh mikroszámítógép által népszerűsített WIMP (ablakok, ikonok, egér és mutató) input-output rendszer ilyen típusú.

Ezzel kapcsolatos a *homlokzatprocesszor* is. Ez egy olyan hardver, amely több, rendszerint különféle terminálokkal és kommunikációs rendszerekkel dolgozó felhasználó, valamint egy központi számítógép között közvetít.

Lásd még: Természetes nyelvi felület

HOPFIELD-HÁLÓ

Neuron-
hálózatok

A Hopfield-háló az egyik legelső típusú neuronhálózat. A Hopfield-háló más neuronhálózatoktól az alábbiakban tér el:

1. A neuronok folytonos, analóg válaszokra, nem csupán minden-vagy-semmi típusú válaszokra képesek.
2. Minden neuronnak van belső kapacitása, azaz kis saját memóriája, amely megőrzi az utolsó inputot, valamint „szökő árama", amely idővel kitörli ezt a memóriát.
3. A háló neuronjai nem feltétlenül azonos időben sülnek ki (vagyis a hálózat aszinkron).

Ezen túlmenően a neuronok közötti kapcsolatok szimmetrikusak, tehát ha I neuron serkenti J neuron működését, akkor J neuron is serkenti I aktivitását. Ez ugyan nem lényeges feltétele a hálózat működésének, de Hopfield elemzésében az alapfeltételek közé számított.

Minden neuron összegzi a különböző inputjait, majd az adott pillanatnyi inputtól függő (de azzal meg nem egyező) outputot ad, majd a válasza lassan gyengül.

Kimutatták, hogy a Hopfield-hálók képesek olyan problémák megoldására, amelyek hagyományos komputeren NP-nehezek. Jó példa erre *Az utazó ügynök problémája*, amelyre a hálózat néhány tucat lépésben kvázi-optimális megoldást talál, míg hagyományos számítógéppel több millió lépés szükséges.

Lásd még: Neuronhálózatok, Connection-elv

HŰTÉS

Neuron-
hálózatok

A szimulált hűtés olyan problémamegoldási módszer, amelyet nagymértékben párhuzamosított komputereken lehet megvalósítani. Jelenleg kutatási stádiumban lévő eszköz, amely azonban ígéretes jövő elé néz.

A hűtés a „hegymászó" problémamegoldó módszerek egyik változata, és „lokálisan optimális" megoldásokkal közelíti meg a problémát. A hűtés metallurgiai folyamatát veszi modellként. A fémet kevéssel az olvadáspontja alá hevítik fel, majd lassan lehűtik: így minden atomnak elég ideje marad, hogy pontosan a helyére kerüljön. A számítógépes analógia, mint említettük, a hegymászó problémamegoldási módszerek változata, amely a hő számítógépes megfelelőjeként random „zajt" visz bele a számításba. Ezáltal megakadályozható, hogy a program megrekedjen egy lokális optimumnál: a zaj előbb-utóbb kimozdítja belőle. Ez a módszer azonban rendkívül időigényes, és így csak nagymértékben párhuzamosított számítógépen alkalmazható. A hűtés módszerét alkalmazták neuronhálózatok működésének optimalizálására.

Egy problémamegoldásra specializált számítógépet, amely a szimulált hűtés elvén működik, Ludwig Boltzman fizikusról *Boltzmann-gépnek* neveztek el.

Lásd még: Hegymászás, Neuronhálózatok

INDUKCIÓ

Logika

Az indukció az a fajta logikai következtetés, amelynek segítségével konkrét esetekből általános konklúziókat lehet levonni. Noha indukcióval nem juthatunk logikai értelemben szigorúan helytálló következtetésekre, az esetek nagy részében helyes eredményt kapunk vele, ezért a mindennapi életben is széles körűen használjuk. A kisgyermek, de sokszor a felnőtt is, indukció révén sajátítja el ismereteit, noha az tökéletesen félre is vezetheti a tanulót. A kisbaba, miután hosszú időn át a mama, a pelenka és szőrös állatkák veszik körül, megtanulja, hogy minden puha. Ebből aztán problémák származnak, amikor bútorokkal, kocsikkal stb találkozik. Tehát az indukcióval, bár nagyon hasznos eszköz, óvatosan kell bánnunk.

Az indukció segítségével történő tanulás három fázisban játszódik le:

1. *Indukció* – az első általánosítás, azaz a hipotézis megalkotása;
2. *Specializáció* – az a felfedezés, hogy az általánosítás túl széles volt, és csak a tárgyak egy rész-osztályára érvényes; vagyis a hipotézis szűkítése;
3. *Általánosítás* – az a felfedezés, hogy az általánosítás túl szűk volt, és a valóságban több tárgyra érvényes.

Egyes szakértői rendszerek olyan tanulórendszert tartalmaznak, amely indukció segítségével a tárolt konkrét adatokból általános szabályokat alkot. Ezt többek között *diszjunktív* és *konjunktív szerkesztéssel* lehet elérni: VAGY műveletek alkalmazásával általánosabb kijelentést kapunk (Mari haja vörös VAGY

zöld), ÉS műveletekkel pedig speciálisabb kijelentést (Mari haja vörös ÉS a cipője zöld). Ezt logikai nyelvekkel, például Prologgal elég egyszerűen lehet programozni; azonban, akárcsak saját indukcióinkkal, néha hibás eredményhez jutunk.

Az indukció egy további jelentése, hogy egy vezetőben a váltakozó mágneses tér áramot kelt. Ennek azonban nem sok szerepe van a mesterségesintelligencia-kutatásban!

Lásd még: Dedukció, Tanulás

INTERPRETER

Általános
számítógépes
kifejezés

Ez a program más, magas szintű nyelven (például Basicben vagy Lispben) írt számítógépes programokat interpretál, fordít le a gép által végrehajtható programra. Az interpreter alapvető feladata tehát az, hogy segítségével a számítógép megértse a programnyelveket (például a Basicet). Egy időben a programnak csak kis részével – általában egyetlen sorával – foglalkozik, és ezeket egymás után fordítja le magának.

Az interpreternek nagy előnyei és nagy hátrányai vannak a compilerrel szemben. Az előnye abban áll, hogy a számítógép a fordítás közben le is futtatja a programot. Ha tehát a 17-es sorban hiba van, a gép megáll, és lehetővé teszi, hogy kijavítsuk a hibát, még mielőtt a 18. sorhoz érne.

Hátránya, hogy menet közben minden sort dekódolnia kell, még akkor is, ha már 100-szor dekódolta. A dekódolás része magának a futtatásnak, így az ilyen program futtatása sokkalta lassabb.

Akárcsak a természetes nyelvi feldolgozó rendszerek, az interpreter is a magas szintű nyelvet fordítja le alacsonyabb szintű megfelelőjére. Az interpreterhez azonban szükséges a teljes nyelv működőképes leírása, szemben a természetesnyelv-feldolgozó rendszerekkel, amelyeknek a természetes nyelv kétértelműségeivel is meg kell tudnia birkózni.

Lásd még: Compiler

INTUÍCIÓ

Elmélet/
Filozófia

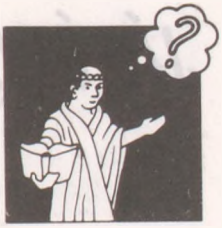
Az MI-vel szembeni egyik súlyos kritika, hogy miután kizárólag logikai és matematikai módszerekkel dolgozik, nem képes számot adni az intelligencia legvalóságosabb jeléről – az intuícióról. Roger Penrose a The Emperor's New Mind (A Császár új esze) című könyvében úgy érvel, hogy az intelligens gép egész fogalma hibás, mivel a mai módszerek egyike sem foglalkozik azzal a szereppel, amit az intuitív ugrás játszhat az intelligenciában.

Ebben az összefüggésben az intuíció két külön dolgot jelent. A hétköznapi életben az intuíció alig jelent többet, mint egy elvárást. Intuitív érzésünk van arról, hogy egy kocsinak hogyan kellene működnie, hogy milyen lesz az idő, vagy hogy valaki mit fog mondani a következő pillanatban, ám a pszichológiai tanulmányok megmutatták, hogy ez majdnem teljes egészében annak tulajdonítható, hogy már láttunk autót, időjárást, tapasztaltuk barátunk efféle viselkedését. Az intuíciónak ellentmondó esemény tehát egyszerűen váratlan eseményt jelent. Ebben az értelemben a számítógépnek is lehetne intuíciója, ha elég nagy memóriája volna a korábbi tapasztalatok tárolására.

Egy kreatívabb értelemben az intuíció azt jelenti, hogy nem teljes vagy ellentmondó információból egységes magyarázatra tudunk átugrani. Nem világos, hogy ez a fajta intuíció mely részben köszönhető szintén a széles körű tapasztalatoknak, amelyeket produkciós szabályok alakjában lehet megfogalmazni, és mely részben a nem logikai gondolkodásnak.

Lásd még: Szimuláció vagy emuláció

ISMERETELMÉLET



Elmélet/
Filozófia

Az ismeretelmélet a tudás tudománya. Ezt a némiképp elvont diszciplínát az a kívánság hozta kapcsolatba az MI-vel, hogy pontosan tudjuk definiálni a tudást, hogy tudásalapú rendszereket készíthessünk. Az ismeretelmélet rákérdez, mit jelent például, hogy egy rendszer tud valamit. Az a szövegszerkesztő, amellyel ez a könyv készült, sok MI-vel kapcsolatos szót tartalmaz, de vajon *tud*-e valamit erről a tárgykörről? Az ismeretelméletet az is érdekli, hogyan lehet ismereteket szerezni, és hogyan módosíthatja őket a tapasztalat. A hiedelmek szintén fontos részét képezik az ismeretelméletnek, hiszen a legtöbb emberi tudás tények, illetve tényekre vonatkozó hiedelmek vegyüléke.

Végül az ismeretelmélet egy olyan általános kifejezés, amely lefedi a különböző tudástípusok, például az implicit, illetve explicit, sekély vagy mély tudás vizsgálatát.

Lásd még: Tudástervezés

JÁTÉKOK FÁJA

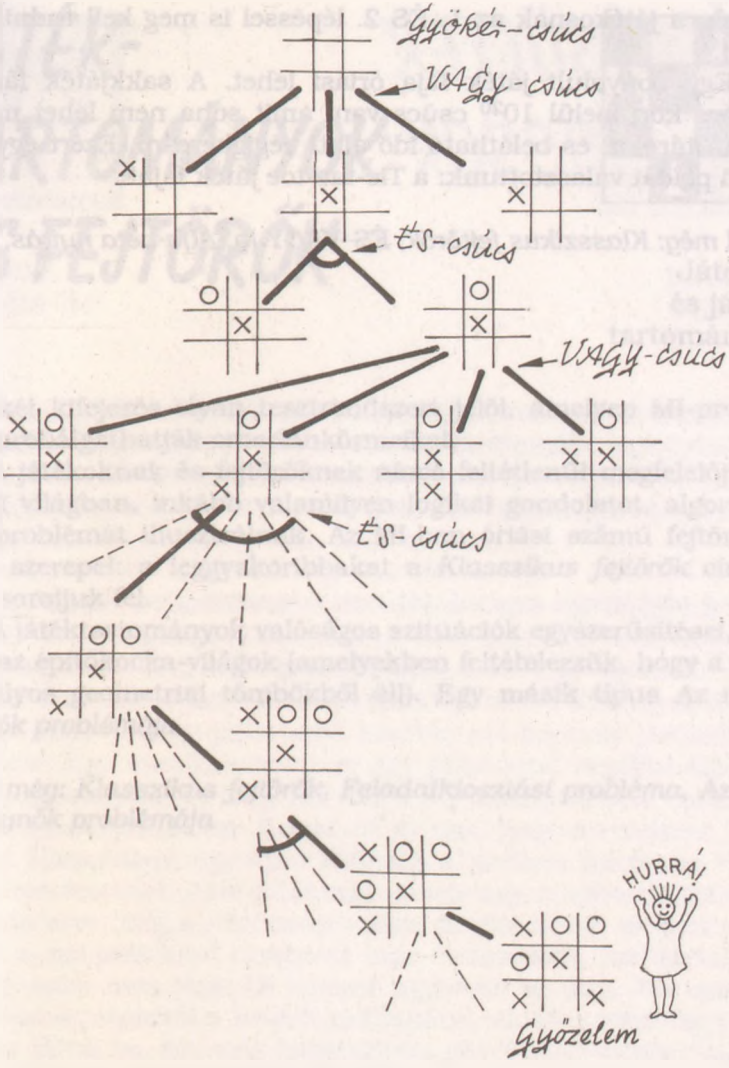
Játékok
és játék-
tartományok

Egy játék fája valamennyi lehetséges lejátszást reprezentálja. Ebben az esetben speciális játékokról van szó, amelyeket két, felváltva lépő ellenfél játszik. A játék folyamán végig mindkét játékos tudja, hogy mik a lehetséges lépések, és egy-egy lépésnek mi a várható hatása. Ellentétben a Kígyók és létrákkal vagy a Ludóval, amelyekben kockadobás formájában a szerencse is szerepet játszik, a játék itt kizárólag a játékosok ügyességén múlik. A játékban az sem lehetséges, hogy a játékosok takarják a kezüket, mint például a kártyában.

A játék fájának van egy gyökércsúcsa, amely a játék kiinduló állásának felel meg (lásd a túloldali ábrán). A gyökércsúcs után következő csúcsok azok az állások, amelyek a kezdő állásból egy lépésben elérhetők. Az ágak végén levő csúcsokat *terminális csúcsoknak* nevezzük. Ezek a játék végső fázisait jelentik, ahol valamelyik játékos nyert, vagy döntetlen alakult ki.

A játékosok által tehető lépéseket VAGY-, illetve ÉS-csúcsokként tüntethetjük fel a keresési fában (lásd ÉS-VAGY fák). Ez a következőképpen magyarázható: amikor egy játékosra kerül sor, választhat, hogy milyen lépést tesz. Ha legalább az egyik lépés olyan állásba viszi, ahol biztosan győzni tud, akkor lehetséges a győzelme. A játékos egyik VAGY másik lépéssel győzni tud: a csúcs tehát VAGY-kiterjesztés.

Amikor az ellenfél lép, neki nincs választása. Ő csak akkor tud garantáltan nyerni, ha az ellenfél minden lehetséges lépése olyan állásba vezet, ahol számára biztos a siker. A játékos lépésének az ellenfél minden válasza mellett működnie kell. Eszerint a játékban az ellenfél lépéseinek megfelelő csúcsok ÉS-kiterjesz-


Gyökér-csúcs
VAGY-csúcs
ÉS-csúcs
VAGY-csúcs
ÉS-csúcs
HURRÁ!
Győzelem

tésűek: a játékosnak az 1. ÉS 2. lépéssel is meg kell tudnia birkózni.

Egy bonyolult játék fája óriási lehet. A sakkjáték fájában például körülbelül 10^{20} csúcs van, amit soha nem lehet memóriában tárolni, és belátható idő alatt végigkeresni. Ezért egy egyszerű példát választottunk: a Tic-tac-toe játék fáját.

Lásd még: Klasszikus fejtörők, ÉS–VAGY fa, Alfa-béta nyírás, Sakk

JÁTÉK-TARTOMÁNYOK ÉS FEJTÖRŐK

Játékok
és játék-
tartományok

Mindkét kifejezés olyan tesztrendszert jelöl, amelyen MI-programok próbálgathatják oroszlánkörmeiket.

A játékoknak és fejtörőknek nincs feltétlenül megfelelőjük a valódi világban, inkább valamilyen logikai gondolatot, algoritmikus problémát illusztrálnak. Az MI-ben óriási számú fejtörő és játék szerepel: a leggyakoribbakat a *Klasszikus fejtörők* címszó alatt soroljuk fel.

A játéktartományok valóságos szituációk egyszerűsítései, például az építőkocka-világok (amelyekben feltételezzük, hogy a világ szabályos geometriai tömbökből áll). Egy másik típus *Az utazó ügynök problémája.*

Lásd még: Klasszikus fejtörők, Feladatkiosztási probléma, Az utazó ügynök problémája

JÓLFÉSÜLTEK ÉS KÓCOSOK

Általános
MI-kifejezés

A jólfésültek és a kócosok a gondolkodás két szélsőségét képviselik a mesterségesintelligencia-kutatásban. A jólfésültek úgy gondolják, hogy a gondolkodásnak van logikai és matematikai elmélete: hogy az intelligenciát valamilyen formális elmélettel le lehet írni.

A kócosok ezzel szemben a gondolkodás erejében hisznek: hogy a problémák megoldását nem a matematikai elmélet, hanem a tudás biztosítja.

A kócosokat tehát a problémamegoldás gyakorlati módszerei érdeklik, és kevésbé a háttérben húzódó elmélet.

KÁOSZ

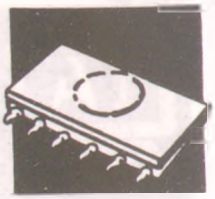
Elmélet/
Filozófia

A káosz egy divatos matematikai kifejezés, amely az MI-ben is megjelent. Segítségével megjósolhatatlan rendszereket lehet leírni. A káoszelmélet alkalmazható a neuronhálózatok és a genetikus algoritmusok területén, és általában segítséget nyújthat az intelligencia alapjelenségeinek a leírásában.

Egy *determinisztikus rendszerben*, például egy lengő ingánál, a rendszer pillanatnyi állapota pontosan meghatározza, hogy milyen állapotban lesz a következő pillanatban. Egy egyszerű rendszernél, ha tudjuk, milyen állapotban van most, ki tudjuk számítani az állapotát egy pillanattal később, sőt bármely jövőbeli pillanatban. Egy *kaotikus rendszer* egy pillanattal későbbi állapota még megjósolható, de hogy hol lesz a távolabbi jövőben, azt szinte lehetetlen megmondani. Ennek az az oka, hogy a rendszer pillanatnyi állapotában egy apró változás a jövőben hatalmas változásokhoz vezethet. Az a játékinga, amely egy mágnes fölött leng, jó példa erre. Míg a közönséges inga megjósolható módon viselkedik, a mágnes által rángatott inga összevissza, szabálytalanul mozog, soha nem járja be kétszer ugyanazt az utat. Ha *egészen pontosan* ugyanarról a helyről indítanánk, akkor persze ugyanazt az utat járná be. Ám elég hajszálnyira eltérő kezdőpontot választani, hogy egészen másképp mozogjon.

Mind a determinált, mind a kaotikus rendszerek szemben állnak a *véletlen rendszerekkel*, amelyeknek a jövőbeli állapotait nem kizárólag a pillanatnyi állapot határozza meg.

KÉPKOCKA-TÁROLÓ



Hardver

Ez a gép látó rendszerének része, amellyel a képdetektor (rendszerint egy módosított TV-kamera) outputját fogadja, és numerikus formában tárolja a számítógép számára. Szükség esetén több képkockát is tud tárolni, hogy kellő időben átadja a komputernek.

Lásd még: Látás

KERESÉS

Keresés

Általában keresésnek nevezzük az olyan módszert, amellyel egy probléma megoldásához el lehet eljutni, amikor a lehetséges utak száma magas.

Tegyük fel például, hogy egy nagy kulcscsomóról akarjuk kikeresni azt, amelyik nyitja az ajtónkat. A legtöbb ember összevissza próbálkozna. Aztán meg feldühödne, mert nem emlékszik, hogy melyik kulcsot vizsgálta már. Nem sokan próbálnák végig módszeresen egyik kulcsot a másik után, félretéve a rossz kulcsokat, amíg elő nem kerül az igazi.

Az MI-ben a keresés folyamata ilyen módszeres gondolkodást igényel. A programozónak alaposan végig kell gondolnia, hogyan kell ábrázolni a probléma kiinduló helyzetét, a keresett megoldást, valamint a közbülső fázisokat. Azt is meg kell határoznia a számítógép számára, hogy melyek a megengedett lépések a problémamegoldás egyik átmeneti állapotából a másikba.

A problémamegoldás fázisainak egyik legkorábbi ábrázolási módszere az „állapotgráf" fogalmára épült. Ebben a problémamegoldás közbülső fázisait „állapotok" jelölték. A hálószerű struktúrában elhelyezkedő állapotok közötti átmeneteket „operátorok" segítségével lehetett megtenni. A megoldást egy út képviseli a kezdeti állapot és a végállapot között. Az állapotgráfra találunk példát a *Játékok fája* szócikkben. Az állapotgráf pontjainak (vagy „csúcsainak") a halmazát szokás „keresési térnek" nevezni.

A keresési fa megmutatja, hogy az állapotgráfban mely utakat jártunk már be. A keresés az állapotgráf kezdő csúcsából indul, és az egyes ágak bejárásával egyre növekszik. Ha egy adott állapotra minden lehetséges operátort alkalmazunk, az adott csúcs mindegyik rákövetkezőjét megkapjuk.

Sok olyan probléma van, ahol a keresési tér kezelhetetlenül nagy: minden csúcsnak óriási számú rákövetkezője lehet. Ne

feledjük, hogy a számítógép egy időben a keresési fának mindig csak egy kis részletét látja. Nem képes arra, hogy intelligens módon az egészet áttekintse, és a megoldást azonnal megtalálja, amire az ember képes lehet. A számítógépnek a csúcsokat szisztematikus módon egyenként kell bejárnia, és valamilyen módszerrel felmérnie, hogy az éppen sorra vett csúcs vajon közelebb visz-e a megoldáshoz. Emellett arra is ügyelnie kell, hogy egyszerre ne legyen túl sok információ a memóriában.

A szisztematikus keresés klasszikus módszerei a mélységben, illetve a szintben induló keresés. A *mélységben induló keresésnél* a számítógép kiválaszt egy ágat, és azt teljes egészében végigjárja, mielőtt visszalépne, és egy másik ággal próbálkozna. A *szintben induló keresésnél* ezzel szemben a keresési fa szintjeit nézzük végig sorban egymás után.

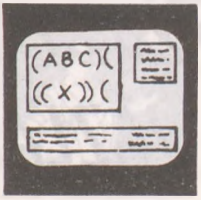
Noha a számítógép sokszor nagyon gyorsan be tudja járni a keresési teret, a kiterjedt és irányítatlan keresés gyakran nem hatékony módszer. Sok esetben segíthetnek a heurisztikák (ökölszabályok, amelyek a csúcsokhoz értékeket rendelnek, jelezve, hogy melyik mennyire „hasznos" a keresés szempontjából). Például, ha a londoni metrótérképen a Finchley Road állomást akarjuk megkeresni, eldönthetjük, hogy a pillanatnyi helyünktől északra eső állomásokat jó célba venni. Heurisztikánk ekkor így hangzik: „a kiinduló helyünktől északra eső állomások a jók". Természetesen megeshet, hogy ez tévedés, de mindenesetre a heurisztika segített leszűkíteni a választási lehetőségeket.

Vannak olyan algoritmusok, amelyek garantálják, hogy a számítógép által kijelölt útvonal a kezdőcsúcs és a végső csúcs között egyben minimális költségű útvonal is legyen. Egy jól ismert példa erre az A^T algoritmus (algorithm for trees: fa-algoritmus), amely minden csúcshoz a t értéket rendeli, ha t a gyökércsúcstól az aktuális csúcsig vezető utak összes költsége.

A keresés során a számítógép mindig a kis t értékű csúcsokat részesíti előnyben, és azoknak a rákövetkezői között keres tovább. A kedvezőtlen értékű csúcsokat eleve kihagyja a további keresésből. Be lehet bizonyítani, hogy az A^T algoritmus mindig megtalálja a minimális költségű utat.

A keresés iránti érdeklődés a 70-es években volt a tetőpontján; ma nagyobb hangsúlyt kap a tudásreprezentáció problémája.

KERET



Programozási technikák

A „keret" fogalmát Marvin Minsky vezette be az 1970-es évek közepén. A „keret" egyike az információ megszervezésére és reprezentálására szolgáló számos módszernek, amely lehetővé teszi szisztematikus visszakeresését és felhasználását is. Egy keretet miniatűr adatbázisként foghatunk fel, amelyben nyílások vannak, bennük adatokkal. Íme egy példa:

A keret neve:	tészta
Tágabb osztály:	élelmiszer
Tészta_típus:	
Íze:	(csokoládé, citrom felfújt, sodó)
Alapértelmezés:	csokoládé
Szabályok:	If barna töltelék then csokoládé
	If sárga töltelék then citromfelfújt
	If fehér töltelék then sodó
Méret:	
Értékek:	(kicsi, közepes, nagy)
Alapértelmezés:	közepes

A keret jellemzően hierarchikus szerkezetű, egyes nyílások neve szülőről leszármazottra öröklődik. Ezek a „globális" nyílások. A „lokális" nyílások csak az adott keretre jellemzőek. Példánkban minden élelmiszernyílás örökli a „méret" globális nyílást, míg a „tészta_típus" nyílás csakis a tésztakeretre jellemző: más keretben nem szerepel.

A keret a benne levő adatok használatára vonatkozó szabályokat is tartalmazhat. A rendszer a szabályok alkalmazása után az eredményeket más névvel ellátott kereteinek is át tudja adni.

Lásd még: Forgatókönyv, Tudásbázis

2½ DIMENZIÓS VÁZLAT

Látás

A látásfeldolgozó rendszer utolsó előtti lépése egy 2½D vázlat, amely tartalmazza a képből kivonható összes háromdimenziós információt, de semmi egyebet. Egy kocka 2½ D vázlata például a szemben levő, felső és egyik oldalsó lapot tartalmazná, de a hátsó lapot már nem, mert azt csak egy olyan adatbázis alapján lehet kikövetkeztetni a képből, amely a kocka teljes formáját tartalmazza. A teljes képet az összes lapjával együtt 3D képnek szokás nevezni.

Hasonló értelemben szerepel a 2½D vázlat a CAD (computer-aided design: számítógéppel segített tervezés) területén. Itt a 2½D vázlat egy teljes háromdimenziós tárgy megjelenítése a lapos VDU képernyőn különböző technikák (például „takart vonalak") segítségével, de a 3D-s tárgy a számítógépben nincs teljes mértékben ábrázolva. Ezzel szemben a 3D-s vázlat a gépben is teljesen ábrázolja a háromdimenziós tárgyat. Ehhez több információra van szükség, viszont élethűbb képeket is lehet vele előállítani.

Lásd még: Mélység

KIBERNETIKA

Elmélet/
Filozófia

A kibernetikát Norbert Wiener dolgozta ki a 40-es években mint a gépek és a biológiai rendszerek vezérlésének és kommunikációjának egységes elméletét. Így egyaránt képes leírni, ahogy az állatok a végtagjaikat mozgatják, illetve ahogy a gépek végrehajtanak bizonyos funkciókat.

A társadalom kibernetikájának tanulmányozása ma már a szociológia területére tartozik: a biológiai rendszereké beleolvadt az ökológiába, fiziológiába és kognitív tudományba (melyek közül csak az utóbbinak van közvetlen kapcsolata az MI-vel). Ezért a kibernetika szűkebb, modern értelmében már csak gépek, elsősorban a robotok vezérlésére és kommunikációjára vonatkozik. Elsősorban a visszacsatolás mechanizmusait tanulmá-nyozza, amelyek révén a gép nyomon követi saját tevékenységét. A visszacsatolás a vezérlés kulcsfogalma. A visszacsatolás mechanizmusában egy-egy művelet eredménye vezérli ugyanazt a műveletet. Például egy autó sebességmérőjét össze lehet kapcsolni a gázpedáljával. A negatív visszacsatolás azt jelenti, hogy az output növekedése az input csökkenését eredményezi – ha például az autó felgyorsul, a rendszer csökkenti a gázt – ez egy klasszikus „stabilizáló" mechanizmus. Pozitív visszacsatolásnál az output növelése az input növekedéséhez vezet.

KIJELENTÉS

Logika

A kijelentés olyan egyszerű ténymegállapítás, amit a program igaznak tekint. Például:

Szeretem a halat.
Szeretem a tejet.
A macskák szeretik a tejet és a halat.

A logikai programozásban a program néhány egyszerű típusú kijelentésből áll, amelyek objektumok és ideák kapcsolatát írják le. A program „futtatásával" új tényeket vezetünk le, amelyek szintén kijelentésekben jelennek meg. Így a fenti tényeket felhasználó program a következő kijelentésre juthat:

Én macska vagyok.

Lásd még: Logikai programozás, Deklaratív programnyelvek

KIJELENTÉS-KALKULUS

Logika

Azt a számítógépes logikát, amely a komputerrel végzett logikai következtetéseket is irányítja, hagyományosan „szimbolikus" vagy „matematikai logikának" nevezik. Ennek egyik része a kijelentés-kalkulus és az ennél bonyolultabb *predikátum-kalkulus*. A predikátum-kalkulus a logikai programnyelvek alapja, amelyeket sokszor használnak MI-alkalmazásokban. A kijelentés-kalkulust, mint olyat, nem használják, az csak a predikátum-kalkulus előfoka.

A kijelentés-kalkulus az atomi kijelentés, a „kijelentésnek" nevezett alapvető állítás gondolata körül forog, amelyhez az IGAZ (TRUE) vagy HAMIS (FALSE) kijelentést rendelhetjük. Például

s kijelentés	2 meg 2 az 4	IGAZ
p kijelentés	Harry haja vörös	HAMIS
q kijelentés	Meg vagyok fázva	IGAZ.

A kijelentések önmagukban nem túl hasznosak. A velük kifejezhető tények köre azonban jelentősen megnő a kijelentés-műveletek használatával:

NEM (NOT);
ÉS (AND);
VAGY (OR);
KÖVETKEZIK (IMPLIES).

A kijelentéseket segítségükkel számtalan módon kombinálhatjuk, és a kijelentés-logikai mondatokat építhetünk belőlük;

(p ÉS (p-ből KÖVETKEZIK q))-ból következik q.

A kijelentés-mondatokban kisbetűk jelzik az egyes kijelentéseket, a zárójeleket pedig úgy használjuk, mint általában a matematikában. A kijelentések felfogása tényleg nagyon matematikus módon történik, tekintet nélkül egyes elemeinek valóságos jelentésére.

A kijelentés-műveletek viselkedését az *igazságtáblák* írják le. Ebből ránézésre kiderül az egyes műveletek hatása:

Igazságtábla a VAGY-hoz

p	q	(p VAGY q)
I	H	I
I	I	I
H	I	I
H	H	H

Ebből láthatjuk, hogy például ha p és q kijelentés igaz, akkor a (p VAGY q) kijelentés is igaz.

Beszéltünk tehát a kijelentés-kalkulusban megengedett állításokról, de arról nem, hogyan használjuk őket.

A kijelentés-kalkulus a predikátum-kalkulustól eltérően nem engedi meg, hogy a szabályok átalakítsák magukat a kijelentéseket. Kijelentés-műveletekkel mégis lehet egyszerű következtetéseket levonni. Vegyük például a következőképpen leírható következtetési módszert, a *modus ponenst* (lásd *Dedukció*):

(p ÉS (p-ből KÖVETKEZIK q))-ból KÖVETKEZIK q

vagyis: ha p igazságából következik q igazsága, és p-ről tudjuk, hogy igaz, akkor q is igaz. Ezt a módszert így is ábrázolhatjuk:

p		premissza
p-ből KÖVETKEZIK	q	premissza
q		konklúzió

Egy példa a *modus ponensre*:

Hideg van
Ha hideg van, kabátot kell vennem

Kabátot kell vennem

Ez ugyan triviálisnak hangzik, de a lényeg az, hogy matematikailag helytálló és logikailag megalapozott. A kijelentés-kalkulus nagyszámú következtetési sémát alkalmaz, amelyeket az igazságtáblák alapján lehet igazolni.

KÍNAI SZOBA

Elmélet/
Filozófia

A Kínai szoba az MI, speciálisan a természetesnyelv-feldolgozás metaforája. John Searle találmánya azt hivatott megmutatni, hogy az MI nem valóságos intelligencia, csak annak szimulációja.

Képzeljük el, hogy valaki egy szobában áll egy nagy könyvvel. Semmit nem tud kínaiul, de a könyv nagyszámú szabályt ír le kínai karakterekre vonatkozólag. Például előírhatja, hogy ha két speciális karakterkombináció egymást követi, akkor ezt írja le, és dobja ki az ajtón levő levélnyíláson.

Mi, akik a kísérletet végezzük (és jól értünk kínaiul), kintről előbb egy kínaiul írt történetet dobunk be a szobába, majd több, szintén kínaiul írt kérdést. Az ember végignézi a történet és a kérdések karaktereit, kikeresi majd leírja őket a könyvből, és kidobja. Így kérdéseinkre kínai válaszokat kapunk. Azonban, érvel Searle, a férfi csupán végrehajt egy utasítást, hogy adott körülmények közt valamit kell tennie. A jelek nem hordoznak számára jelentést. Semmit nem ért meg, csak szimulálja a megértést.

Világos az analógia a szakértői rendszerekkel és a természetesnyelv-feldolgozással. A szabály-alapú rendszerek, a nagy szabálykönyvre hasonlítanak, amit a komputer feldolgoz, noha semmit nem „ért" belőle, csupán szimulálja az intelligenciát.

Természetesen ugyanezt a gondolatmenetet emberekre is alkalmazhatjuk: ha látom, hogy értelmesen válaszoltál a kérdésemre, még nincs okom feltételezni, hogy meg is értetted.

Erre az érvre válaszol részletesen a *rendszer elve*, amit Douglas Hofstadter és Daniel Dennett fejtett ki *The Mind's I* (*Az ész énje*) c. könyvében. Az intelligencia nem a szoba egyes elemeiben rejlik, ahogy az embernél sem az egyes idegsejtekben, hanem a rendszer mint egész konfigurációjában.

Lásd még: Szimuláció vagy emuláció

KÍSÉRLETI RENDSZER

Általános számítógépes kifejezés

Kísérleti rendszereket a laboratóriumban állítanak össze, hogy egy elvet vagy egy probléma megoldási módszerét demonstrálják.

A fejlesztés következő lépése a *prototípus* (amely mindazt tudja, amit a végső kiadásnak ideális feltételek mellett tudnia kell), valamint a próbaváltozat (a béta-tesztelésre szánt változat, amellyel a valóságos működést lehet kipróbálni). A gyakorlatban persze elmosódik a határ ezek között a fázisok között.

Lásd még: Gyors prototípus

KITERJESZTETT ÁTMENETHÁLÓ

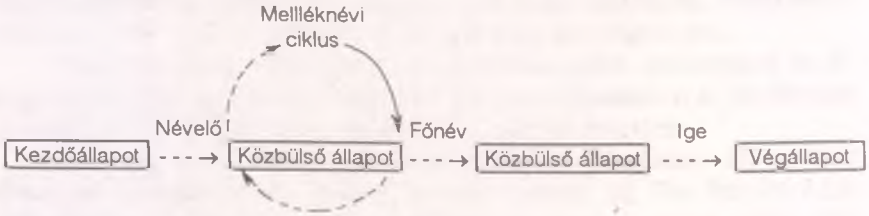
Természetes nyelv feldolgozása

A legismertebb természetesnyelv input eszközök jó része a kiterjesztett átmenetháló (augmented transition network, ATN) technikáját használja a mondatok feldolgozására.

Amikor az „az" névelőt halljuk, a következő szóra főnevet vagy melléknevet várunk. Ha a második szó melléknév, utána még egy melléknév, vagy főnév következhet. Ily módon anticipáljuk a mondat struktúráját, és töltjük be sorban a szavakat.

Az ATN ezen az ötleten alapul. A mondat struktúráját a számítógép főnevekből, melléknevekből, igékből, stb. álló háló formájában ábrázolja, amelyben utakat is kijelöl a hálózat által elfogadott szósorrendeknek megfelelően.

Az alábbi hálóban az elemző (a mondatot elemző program) a „start" állásból indul ki. Ezután a „névelő" címkéjű élen halad tovább (ami „a", „az" vagy „egy" lehet), és felismeri az első szót. A középső kockában egy vagy több melléknevet fogad el. Hogy eljusson az utolsó kockába, majd a végállapotba, egy főnévre, majd egy igére van szükség.



Ez a hálózat többek közt az alábbi mondatokat fogadná el:

A gyapjas fehér birka béget.
A nagy fehér szőrös cica eszik.

A valóságos természetes nyelvi alkalmazásokban ennél persze sokkal bonyolultabb hálók szerepelnek. Különböző hálók össze is kapcsolódhatnak, és az egyik valamelyik mondatrész (névszói csoport, igei csoport stb.) elemzése után a mondatot át is adhatja a másiknak. Ahol nem azonnal egyértelmű, funkcióregiszterekben tartjuk számon a „mondatrész" elemzés lehetséges eredményeit. Ez teszi lehetővé, hogy megtaláljuk például az alanyt olyankor is, amikor a mondat bonyolult struktúrával rendelkezik. Tulajdonság-regiszterekben tartjuk számon az egyes szavak grammatikai jegyeit, például, hogy a főnév egyes vagy többes számban áll-e. Az ATN nem pusztán egyik vagy másik típusba sorolja a szavakat: ellenőrzi például a főnevek és igék számbeli egyezését, és a végződések alapján a jelentéshez is közelebb visz.

Az ATN legnagyobb hátránya a rugalmatlansága. Az ATN elemző csak akkor fogad el egy mondatot, ha a végállapotba jut. Ha egy úton nem jut el a végállapotba, visszalép, és új útvonallal próbálkozik. Az utak száma a háló nagyságával exponenciálisan nő, egy bonyolult mondatnál akár 600 különböző utat is végig kell próbálni a sikeres elemzéshez. Természetesen az is előfordulhat, hogy egy mondat „nem fér bele" az ATN nyelvtanába, és egyáltalán nem elemezhető.

Lásd még: Elemző, Természetes nyelv feldolgozása, Grammatika

KLASSZIKUS FEJTÖRŐK

Játékok
és játék-
tartományok

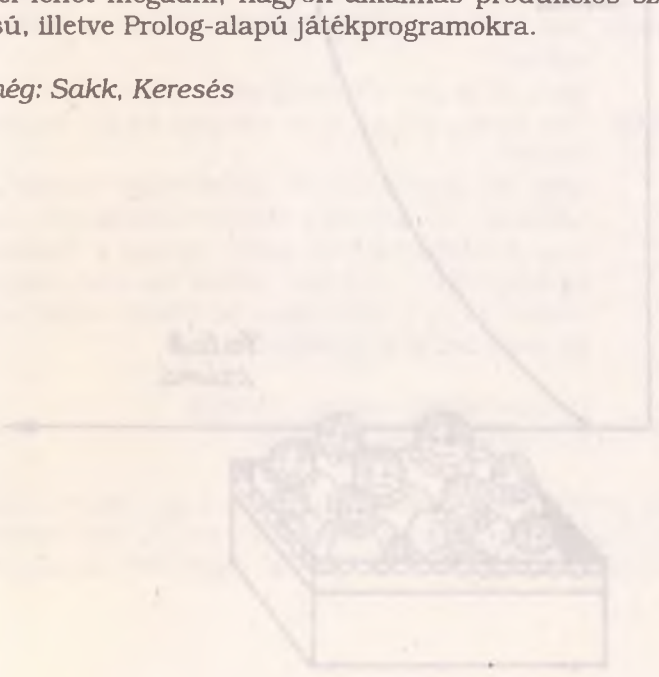
Van néhány fejtörő, amely az MI egyes területeinek klasszikus próbájává vált. A játéktartományokkal ellentétben azonban ezek nem a valóságos világot kívánják ábrázolni.

1. *Kannibálok és misszionáriusok.* Két kannibál és két misszionárius át kell keljen a folyón kétszemélyes csónakban. Csak misszionárius evezhet, és ha valamelyik parton több a kannibál, megeszik a misszionáriusokat. Milyen sorrendben keljenek át? Ezt az egyszerű problémát egy állapotgráf szintben vagy mélységben induló végigkeresésével lehet megoldani.
2. *Sakk (vagy dáma).* Mindkettőt 8 × 8-as táblán játssza két játékos, egynél többféle figurával.
3. *Nyolc kocka.* Egy 3 × 3-as táblán nyolc 1 × 1-es kocka van, 1-től 8-ig vagy A-tól H-ig jelölve, egy mező pedig üres. Kezdetben a kockák össze vannak keverve. Egy kockát mindig az üres helyre kell tolni. Az a cél, hogy minél kevesebb lépésben emelkedő vagy csökkenő sorba tudjuk rendezni őket. Hasonló játék a Rubik-kocka, amelynek oldalai kisebb részkockákra vannak osztva, és mindegyik más színű. A kis kockák csoportjainak elforgatásával el kell érni, hogy a nagy kocka egy-egy oldala azonos színű legyen.
4. *Go.* Kétszemélyes taktikai játék. A játékosok felváltva helyezik el saját színű köveiket egy kockákra osztott táblán. Ha az egyik játékos körbezár egy területet, akkor onnan az ellenfél minden követ leveheti. Az nyer, akinek a játék végén több köve van a táblán. A Go szabályai nagyon egyszerűek, de miután minden állásban nagyon magas a lehetséges lépések száma, a játék állapottere elképesztően nagy.
5. *Nim.* Két játékos játssza több (általában legalább három) halom pénzérmével (kővel, vagy másfajta számlálóval). Felváltva

vesznek el köveket, de egy játékos csak egy halomból vehet. Az veszít, akinek az utolsó követ kell elvennie. Ez a játék nagyon alkalmas teszt nyerő stratégia keresésére, amellyel biztosan legyőzhetjük az ellenfelet. Miután ez egy egyszerű játék, könnyű megkonstruálni a teljes keresési (= állapot = játék) fát.

6. *Tic-tac-toe.* Ketten játsszák egy 3 × 3-as táblán. Felváltva lépnek, az egyik kört, a másik keresztet rajzol egy-egy mezőbe. Az nyer, akinek sikerül három jelét egy vonalba rajzolnia. Ez is olyan játék, amelyen tesztelhetjük a keresési fák teljes kimerítésére vagy „nyírással" való végignézésére, illetve nyerő stratégiák keresésére szolgáló eljárásokat.
7. *Hanoi torony.* A táblán három rudacska áll, és a bal oldalira egy sor középen kilyukasztott korong van felfűzve, nagyság szerint csökkenő sorrendben. Állítólag a buddhista meditáció segédeszközéül szolgáló eredeti változat 64 korongot tartalmazott. A játék célja, hogy a korongokat átrakjuk a jobb oldali rudacskára, egyszerre mindig csak egyet mozgatva, és mindig ügyelve arra, hogy nagyobb korong ne kerülhessen kisebb fölé. Akárcsak *Az utazó ügynök problémájánál*, a probléma állapotfája hihetetlen mértékben megnő, ha növeljük a korongok számát. Miután a problémát két egyszerű szabály segítségével lehet megadni, nagyon alkalmas produkciós szabály típusú, illetve Prolog-alapú játékprogramokra.

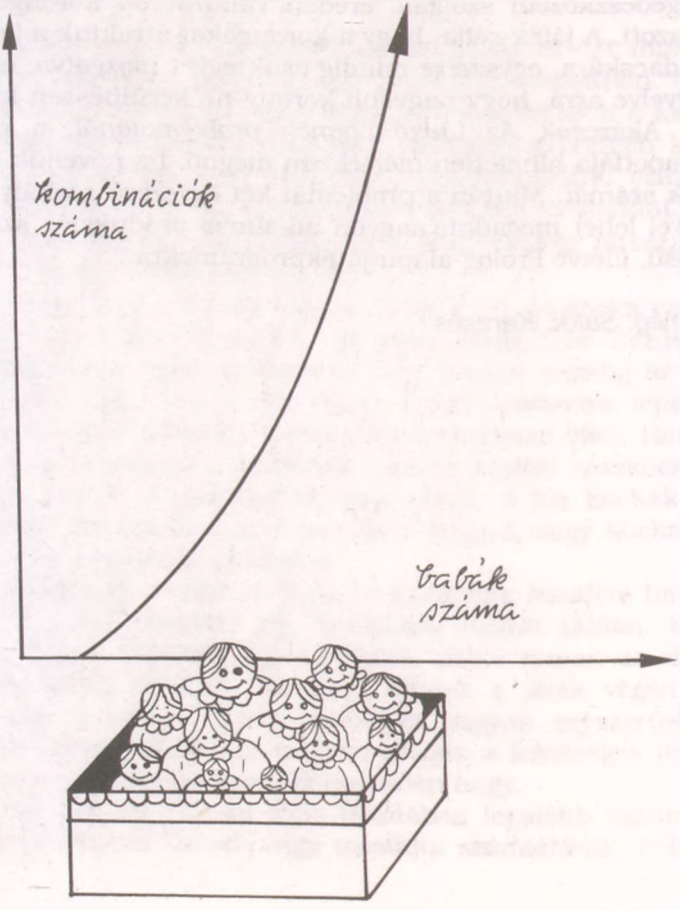
Lásd még: Sakk, Keresés



KOMBINATORIKAI ROBBANÁS

Keresés

Azt jelenti, hogy a problémák méretének növekedésével robbanásszerűen nő azok bonyolultsága. Általánosan használt kifejezés annak leírására, hogy dolgok kombinációinak a száma milyen



óriási mértékben nőhet, ha maguknak a dolgoknak a számát valamivel növeljük. Például képzeljük el, hogy Maruszja-babát kell összeállítanunk egy dobozból véletlenszerűen kiválasztott öt elemből. Minél több a baba a dobozban, annál többféle kombináció van, és annál tovább tart találni az öt egymásba illő elemet. Ebben a feladatban is a kombinatorikai robbanás a probléma: a lehetséges részhalmazok száma meredeken felszökik, ha újabb babákat teszünk a dobozba.

Az utazó ügynök problémája is olyan feladat, amelynek a bonyolultsága robbanásszerűen nő a feladat méretével.

Kombinatorikai robbanás szerint növekvő problémákat nem lehet úgy megoldani, hogy minden lehetséges megoldást végignézünk, ezért más megoldási módokat (heurisztikákat, szabályokat, nyírást) kell alkalmaznunk, hogy csökkenthessük a lehetőségek számát.

Az MI-vel foglalkozó nagy-britanniai *Lighthill jelentés* fogalmazza meg, hogy a mai, egyszerű problémákkal jól működő programok valószínűleg a kombinatorikai robbanásba ütköznének, ha a valóságos problémák komplexitásával kellene megbirkózniuk, vagyis kevéssé alkalmasak az intelligencia modellezésére.

Lásd még: Algoritmus, NP

KOOPERATÍV RENDSZER

Szakértői rendszerek

A „kooperatív rendszer" kifejezés főképpen a szakértői rendszerek területén használatos. Nagy általánosságban egy olyan ideális jövőbeli rendszerre vonatkozik, amely a felhasználó és a gép közötti kapcsolatot gondos tervezéssel a felhasználó igényeihez igazítja. Egy ilyen rendszer interaktív módon szorosan együtt tudna működni a felhasználóval.

Hogy jobban el tudjuk képzelni ezt a látomást, gondoljuk el, hogyan működhetne egy számítógépesített *Yellow Pages* (Szakmai telefonkönyv). A felhasználó és a rendszer között ilyesfajta párbeszéd folyhatna:

Felhasználó: Van 24 órán belül elérhető vízvezetékszerelő a környéken?

Gép: Igen, 30 is van. Felsoroljam?

Felhasználó: Van olyan, aki a ért az eldugult lefolyó megjavításához is?

Gép: Igen, Mr Jones (telefon: 56087) úgy hirdet: „lefolyótisztítás". Õ azonban tizenöt kilométerre lakik innen. A lefolyótisztítás külön címszó a Szakmai telefonkönyvben. „Lefolyó, vízcső és csatornatisztítás" címszó alatt a környéken az alábbi cégek kínálnak 24 órán belüli szolgáltatást:

Drága Cső Kft. telefon: 333409
Kidugítók, telefon: 378409

A kooperatív rendszernek tehát kell, hogy legyen valamilyen modellje, hogy előre el tudja „képzelni", mi az, ami fontos a felhasználónak, és milyenek az igényei. Egyszerű következtetések

levonására is képesnek kell lennie, olyan fogalmakat, kifejezéseket kell tudjon használni, amelyeket a felhasználó is megért.

Egy ilyen kooperatív rendszer megtervezéséhez alaposan meg kell értenünk, hogy az ember hogyan kommunikál egy szakértővel, és mit lehet várni egy ilyen konzultációtól. A kognitív pszichológusok, akiket az emberek gondolkodása és következtetési technikája érdekel, jelentős szerepet játszanak a kooperatív rendszerek kutatásában.

Lásd még: Emberi tényező kutatása

KÖVETKEZTETŐGÉP

Szakértői
rendszerek

A „következtetőgép" fogalma a szakértői rendszerekhez kapcsolódik. Ez az a része a programnak, amely interpretálja a szakértői rendszer szabályait. Amikor a szakértői rendszernek felteszünk egy kérdést, a következtetőgép dönti el, hogy mely tényeket és szabályokat kell használni a válaszhoz. A két leggyakoribb módszer, amit a következtetőgépek a szabályok feldolgozásában alkalmaznak az *előre* és *hátra láncolás*. Mindkettőre külön szócikkben hozunk példát.

Lásd még: Szakértői rendszerek

LÁTÁS

Látás

A látás az a módszer, amelynek révén a gépek képesek (és a jövőben még inkább képesek lesznek) valamilyen távoli érzékelővel, többnyire fénydetektorral nyert adatokat felhasználni. Ezt sokszor két részre osztják:

1. *gépi látás*, azaz a feladat mechanikus része, és
2. *számítógépes látás*, amely az adatfeldolgozást jelenti.

(noha gyakran gépi látásnak nevezik az egész folyamatot). Az MI szinte kizárólag a második területtel foglalkozik.

A hagyományos MI látórendszerek lényegében David Marr munkásságából nőttek ki, aki egy többlépéses módszert vázolt fel a kép elemzésére, egyre növekvő léptékben. Megoldandó problémák itt többek között az alábbiak:

1. *Jegyazonosítás.* Jegynek nevezzük a képnek bármely megkülönböztető részletét, például egy lapját vagy egy élét. A jegyeket *valódi jegyekre* (dőlés, megvilágítás-erősség, irány), illetve *képjegyekre* (síkok, vonalak) szokták osztani. A képfeldolgozás fázisai, a szegmentáció és az élkeresés a képben megtalálható jegyek egész repertoárját állítják össze, amelyek azután együttesen lehetővé teszik a tárgyak azonosítását.
2. *Tárgyazonosítás.* Tulajdonképpen minek felel meg az árnyék? Ehhez kapcsolódik a
3. *Tárgy irányítása.* Merrefelé áll? Ez jelentős probléma a robottechnikában.
4. *Mélység.* Milyen távol van?
5. *Rejtett tárgyak.* Ha nem látom a gnúnak mind a négy lábát, vajon ez azt jelenti-e, hogy háromlábú gnúra bukkantam?

6. *Mozgás.* Hogyan tudom eldönteni, hogy ez a gnú itt azonos-e azzal, amelyet egy pillanattal korábban ott láttam?

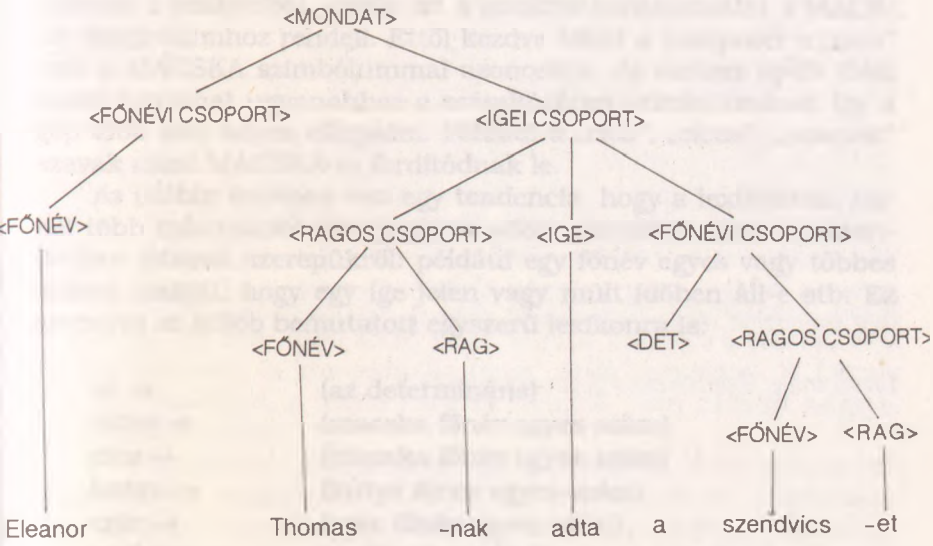
A látással kapcsolatos többi kérdés általában két gondolat köré csoportosítható. A hagyományos látórendszerek fekete-fehér *bináris képből* indulnak ki, ebből *élkereséssel* képrészletekből felépülő éleket vonunk ki, amelyekből azután megszerkesztjük az *elemi vázlatot*. Ezáltal a képet területekre tagoljuk, amelyeket viszont képszegmentumokra szegmentálunk. A *mélységinformációból*, ami az egyes képrészletek távolságát adja meg a kamerától, a felszín *textúra*-elemeivel együtt összeállíthatjuk a *2½D vázlatot*, amely már félig-meddig a teljes jelenet háromdimenziós ábrázolását adja.

Másfajta megközelítést képviselnek a *modellalapú látás*, illetve a *neuronhálózatos* módszerek.

LEVEZETÉSI FA

Természetes
nyelv
feldolgozása

A nyelvtanokat sokszor „levezetési fa" diagramokban ábrázolják. Az alább látható levezetési fa az „Eleanor Thomasnak adta a szendvicset" mondat szerkezetét mutatja be. Figyeljük meg az ilyen ábrázolásoknál használatos terminológiát.



1. *Nem terminális szimbólumok* mindazok, amelyek nem a fa ágainak végén állnak. Az újraírási szabályokban szögletes zárójelben szerepelnek, és további szimbólumokkal lehet helyettesíteni őket.

2. *Terminális szimbólumok* azok, amelyeket már nem lehet tovább átdefiniálni: ezek a fa leveleinél helyezkednek el, mint például „Eleanor" és „adta".

Lásd még: Grammatika, Elemző

LEXIKON

Természetes
nyelv
feldolgozása

A lexikon egy számítógépes szótár. Amikor beírunk egy mondatot az emberi nyelv megértésével fáradozó számítógépbe, első fázisban a lexikon lép működésbe, és a számítógép által kezelhető alakra fordítja le a begépelt szöveget.

Tegyük fel például, hogy a „cica" szót írtuk be. A számítógép kikeresi a lexikonból, amely ezt a konkrét betűsorozatot a MACSKA szimbólumhoz rendeli. Ettől kezdve tehát a komputer a „cica" szót a MACSKA szimbólummal azonosítja. Az emberi nyelv több szava tartozhat ugyanahhoz a számítógépes szimbólumhoz, így a gép több szót képes elfogadni. Például a „cica", „cicus", „macsek" szavak mind MACSKÁ-ra fordítódnak le.

Az utóbbi években van egy tendencia, hogy a lexikonban minél több információt tároljunk az adott szavakról, illetve a mondatban játszott szerepükről: például egy főnév egyes vagy többes számú alakját, hogy egy ige jelen vagy múlt időben áll-e stb. Ez érvényes az alább bemutatott egyszerű lexikonra is:

az →	(az determináns)
cicus →	(macska főnév egyes szám)
cica →	(macska főnév egyes szám)
kutya →	(kutya főnév egyes szám)
egér →	(egér főnév egyes szám)
gyékény →	(padlótakaró főnév egyes szám)
ült →	(ül ige múlt)
-on	(-on rag)

Lásd még: Elemző, Természetes nyelv feldolgozása

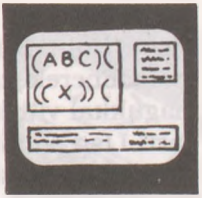
LIGHTHILL-JELENTÉS

Finanszírozás

1972-ben Sir James Lighthill elnökletével összeült az Egyesült Királyság Tudományos-Műszaki Kutatások Tanácsa, hogy értékelje a mesterséges intelligencia néven ismert kutatást. Lighthill konklúzióit, melyek szerint az általános célú problémamegoldó programok kifejlesztésére fordított munka értéke kétségbe vonható, és speciális technikák jól definiált kutatásai mögé kell rangsorolni őket, a számítógépes közösségen kívül a mesterségesintelligencia-kutatás általános elutasításaként értelmezték. Lighthill úgy gondolta, hogy a „játéktartományokkal" folytatott munka aligha lesz alkalmazható a valós problémákra, mert az utóbbiak komplexitása mellett a kombinatorikai robbanás használhatatlanná tenné az elért eredményeket. Különösen keményen kritizált néhány túlzó állítást a robottechnika területén. A jelentés következtében az MI Nagy-Britanniában teljesen háttérbe szorult egészen addig, amíg a japán Ötödik Generációs Projektek kezdeményezése meg nem növelte politikai jelentőségét.

Lásd még: Kombinatorikai robbanás

LISP



Programozási technikák

A Lisp olyan szimbólumfeldolgozó nyelv, amely az objektumok, számok és más szimbólumok közötti kapcsolatokat a listastruktúra segítségével ábrázolja. Maga a Lisp szó is a List Processor (Listafeldolgozó) rövidítése.

Az az ötlet, hogy szimbólumokat listák segítségével lehet feldolgozni, már 1956-ban felmerült, a Logic Theorist (Logika-tudós) programnál, amelyet egy IPL (Információfeldolgozó nyelv) nevű alacsony szintű, nehezen használható nyelven írtak. A Lisp célja egy IPL-re alapozott, könnyebben használható, magasabb szintű nyelv volt.

A Lisp program összetéveszthetetlen jellegzetessége a számtalan zárójel, amely e listákon belüli listák elhatárolására szolgál (lásd *Lista-feldolgozás*). A Lisp az adatokat eljárások és függvények segítségével dolgozza fel: vagyis függvényalapú programnyelv.

Korai megjelenése óta a Lisp folyamatos fejlesztésen ment keresztül. Ma már több tucat Lisp-változat kapható a piacon. Sok dialektusa nem is kompatibilis (nem „értik" egymást). A Lispek két nagy osztálya, a Keleti Parti és Nyugati Parti csoport nagyjából inkompatibilis.

A Lisp Keleti Parti dialektusai a MacLisp nevű változatból fejlődtek ki, amelyet az MIT MI-laboratóriumában készítettek, (ahol a Mac valószínűleg a Machine-Aided Cognition – Számítógéppel támogatott megismerés – rövidítése, semmi köze a Macintosh számítógéphez). A Keleti Parti csoportban mára széles körűen elfogadják a Common Lisp (Közönséges Lisp) változatot mint szabványt. A Common Lisp változatai PC-n is futnak. A Keleti Parti Lisp-változatokhoz tartozik a Symbolics Lisp, Franz Lisp, C-Lisp és PSL.

A Nyugati Parti Lisp valójában Boston környékéről származik, a Bolt, Baranek és Newman cégtől. Amikor a számítógépes szakemberek egy csoportja a kaliforniai Palo Altóba költözött, magukkal vitték Lisp-fejlesztésüket is. Ma a legismertebb Nyugati Parti Lisp az InterLisp, amely mára nagy teljesítményű, rugalmas fejlesztői eszközzé vált.

A Lisp általában interpretált nyelv. Ez azt jelenti, hogy végrehajtás közben egy Lisp-*interpreter* soronként fordítja le az utasításokat gépi kódra. Ezzel szemben a C, Pascal vagy Fortran nyelvek fordítása compilerrel történik: az egész programot előbb le kell fordítani, hogy futtatható formába kerüljön.

Lásd még: Interpreter, Compiler, Lisp chip, Listafeldolgozás

LISP CHIP (LISP-GÉP)

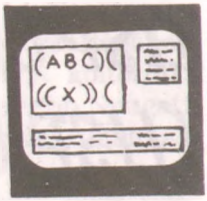
Hardver

A Lisp chipek és Lisp-gépek a Lisp-nyelv futtatására szolgáló hardverek. A legtöbb számítógép processzorát a Basic, Fortran és hasonló jellegű nyelvek futtatásához szükséges műveletekre tervezték, így egyetlen gépi kódú utasításuk van olyan bonyolult feladatokra, mint például JUMP-IF-NUMBER-ZERO (Ugorj-ha-szám-nulla) vagy SEARCH-FOR-NON-ZERO-NUMBER (Keress-nem-nulla-számot) stb. Tehát a Fortran programokat nagyon hatékony gépi kódra lehet lefordítani az ilyen gépeken, a Lispet azonban nem, mert annak egészen más a struktúrája. A Lisp-gépnek más az utasításkészlete, például egyetlen gépi utasítás van a CRD és CAR két alapvető Lisp-eljárásra. A Lisp-gépek architektúrája speciális Lisp-mikrochipekre épül. A Lisp-gépekben sokszor automatikusan történik a „hulladékgyűjtés" (a feleslegessé vált adatrészletek, listák törlése a memóriából), illetve más, Lisp-programozók számára hasznos tevékenység.

A Lisp-gépek egyik jellegzetessége a memóriájuk szervezése, amely megkönnyíti listák készítését. A Lisp-gépek memóriája lineáris tömbök helyett párokból áll, amelyeknek az első eleme tartalmazza az adatot, a második pedig a lista következő elemére mutat.

Lásd még: Lisp

LISTA-FELDOLGOZÁS



Programozási technikák

A lista olyan adatstruktúra, amelyet sok MI-nyelv, különösen a Lisp szívesen használ. A lista függetlenül elérhető, hierarchikusan strukturált komponensekre bontja az ábrázolni kívánt adatokat.

A lista zárójelek közé zárt szimbólumokból áll. Ezeket azután eljárások dolgozzák fel. Például a

PRINT FIRST(sárga piros zöld kék)

eljárás arra szolgálhat, hogy kiírja a (sárga piros zöld kék) lista első elemét.

A nem programozókat talán meglepi, hogy milyen sokoldalú lehet ez a listákon és eljárásokon alapuló egyszerű ötlet. Egy szakértői rendszer szabályait éppúgy lehet listákkal ábrázolni, mint a nyelvtani szabályokat (a *természetesnyelv-feldolgozásban*). A listák szerepének illusztrálására tekintsük az MI egyik kedvenc példáját, a *misszionáriusok és kannibálok fejtörőjét*, és nézzük meg, hogyan lehet a listákkal ábrázolni a megoldás különböző fázisait. A misszionáriusok és kannibálok problémáját egyébként részletesen kifejtjük a *klasszikus fejtörők* címszó alatt.

Bizonyos MI-technikákkal kezelhető problémákat egy sor közbülső *állapottal* lehet jellemezni (lásd *keresés*). Ezeket az állapotokat többek között listákkal lehet ábrázolni. A misszionáriusok és kannibálok problémájában például egy listában tüntethetjük fel a folyó két partján álló embereket. Például

(MMMK CSÓNAK)	három misszionárius, egy kannibál és a csónak
(KK)	két kannibál
( )	üres lista: senki nincs a parton

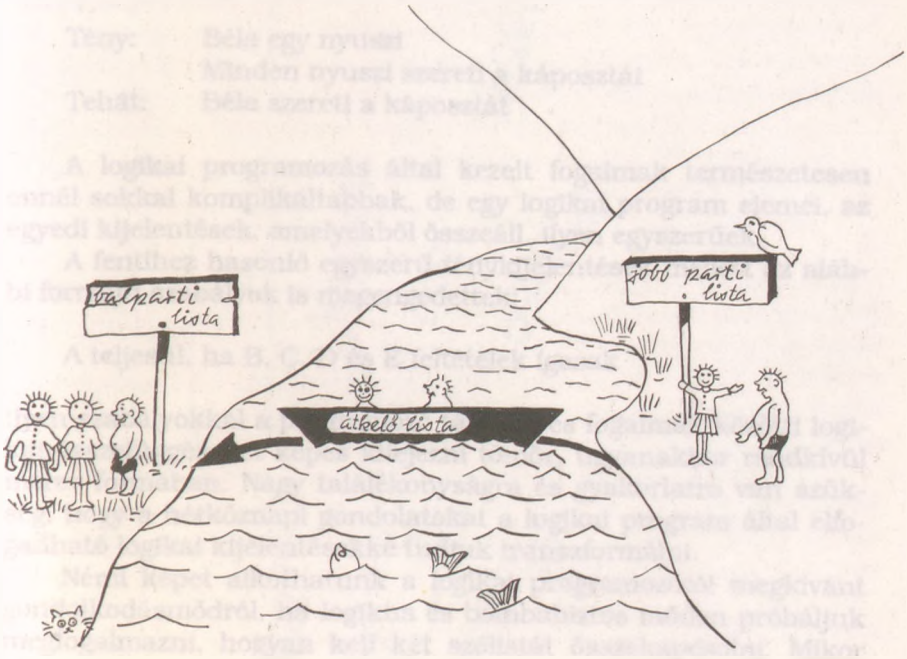
A listákat *eljárásokkal* dolgozzuk fel. Az eljárás tehát a keresési fa egyik állapotából átvisz egy másikba. A listából lehet törölni elemeket, újakat felvenni, felcserélni stb.

A misszionáriusok és kannibálok problémájában az emberek átvitelét egyik partról a másikra olyan eljárás szimulálhatja, amely egy listából (például (MM CSÓNAK)) kivesz néhány elemet, és áthelyezi őket a másik parthoz tartozó listára.

Ebben a fejtörőben az a probléma, hogy egy-egy parton bizonyos misszionárius-kannibál kombinációk halálos kimenetelűek lehetnek, tehát el kell kerülni őket: ha túl sok a kannibál, elfogyaszthatják a misszionáriusokat. Hogyan lehet leírni az ilyen kombinációkat? Az alábbi programrészlet azt definiálja, hogy milyen esetben kerül sor misszionáriusevésre:

```
(DEFUN MISSZI_EVÉS( )
  (OR (M_EVÉS BALPART) (M_EVÉS JOBBPART)))

(DEFUN M_EVÉS(PART)
  (AND( > ('K SZÁMA PART) ('M SZÁMA PART))
       ( > ('M SZÁMA PART) 0)))
```



A fenti kód szerint a misszionáriusok akkor vannak veszélyben, ha a bal vagy jobb parton túl kevesen vannak. „Túl kevés" ebben a szituációban azt jelenti, hogy a kannibálok száma (K) nagyobb, mint a misszionáriusoké (M).

A probléma teljes megoldását végző program sok oldalt töltene be, ezért itt nem közöljük. Reméljük azonban, hogy a megadott példák legalábbis érzékeltetik, hogyan lehet listák segítségével ábrázolni a probléma fázisait.

Lásd még: Klasszikus fejtörők, Lisp

LOGIKAI PROGRAMOZÁS

Logika

A logikai programozás a tudás reprezentálására és a matematikai logika törvényei szerint való felhasználására szolgáló módszer.

A logikai programozás az automatizált következtetés kutatásából nőtt ki. A következtetés az az eljárás, amely egyes detektíveket híressé tett: ismert tényekből és megfigyelésekből okos konklúziókat kell levonni. A kutatókat kezdetben a *rezolúciós következtetés* módszere foglalkoztatta, amelynek révén a *predikátum-kalkulusban* következtetéseket lehet levonni.

Nagyon egyszerű példa a predikátum-kalkulusból:

Tény:	Béla egy nyuszi
	Minden nyuszi szereti a káposztát
Tehát:	Béla szereti a káposztát

A logikai programozás által kezelt fogalmak természetesen ennél sokkal komplikáltabbak, de egy logikai program elemei, az egyedi kijelentések, amelyekből összeáll, ilyen egyszerűek.

A fentihez hasonló egyszerű ténykijelentések mellett az alábbi formájú szabályok is megengedettek:

A teljesül, ha B, C, D és E feltételek igazak

Ilyen szabályokkal a programozó tárgyak és fogalmak közötti logikai összefüggéseket képes kifejezni tömör, ugyanakkor rendkívül merev formában. Nagy találékonyságra és gyakorlatra van szükség, hogy a hétköznapi gondolatokat a logikai program által elfogadható logikai kijelentésekké tudjuk transzformálni.

Némi képet alkothatunk a logikai programozótól megkívánt gondolkodásmódról, ha logikus és bombabiztos módon próbáljuk megfogalmazni, hogyan kell két szólistát összekapcsolni. Mikor

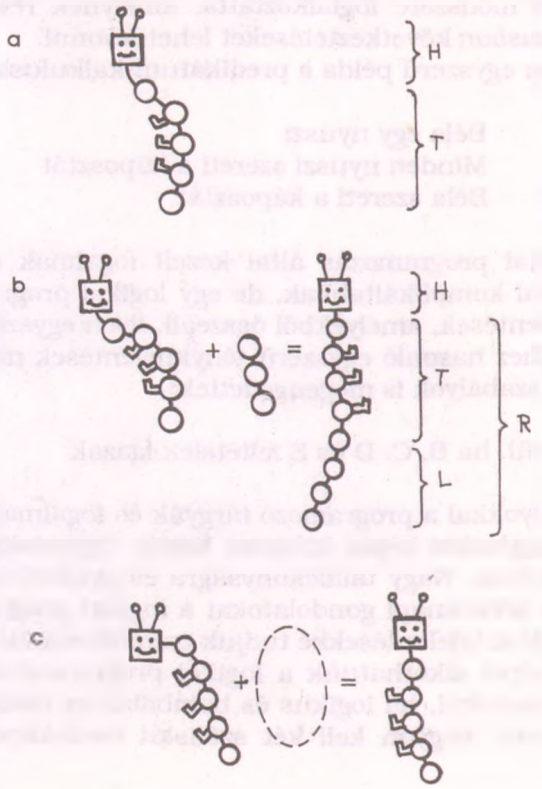
igaz teljes általánosságban, hogy ha az XYZ-vel kezdődő listát összekapcsoljuk az ABC listával, akkor XYZABC-t kapunk?

Az egyik lehetőség, hogy először a különálló listákra állapítjuk meg az elemek sorrendjét, majd akkor, amikor már összekapcsoltuk őket. Ezután a különálló és az összekapcsolt listák logikai viszonyait egy szabályhalmazzal tehetjük explicitté.

Úgy képzelhetjük, hogy minden lista fejrészből (F) és farokrészből (K) áll. A listát a Prolog gyorsírásával így írhatjuk: [F|K] egy F fejből és K farokból álló listát jelent.

Tegyük fel, hogy egy [F|K] listához akarunk kapcsolni egy második B listát. Így olyan listát kapunk, amelynek hosszabb a farokrésze, a fejrésze azonban ennek is F. Az új, hosszabb farokrészt J-vel szokták jelölni. Prologban mindezt így írhatjuk:

append ([F|K], B, [F|J])	ha [F|K]-hoz fűzzük B-t, [F|J]-t kapunk
append (K, B, J)	ha K-hoz fűzzük B-t, J-t kapunk



Másrészt ha van egy olyan listánk, amelynek egyetlen eleme sincs (az üres lista jelölése [ ]), és ezt hozzátapasztjuk B listához, akkor B nem változik.

append ([ ], L, L) ha [ ]-et fűzzük B-hez, B-t kapunk.

Ezeket az összefüggéseket ábrázolja a túloldali diagram. Ennek alapján a teljes *append* predikátum a következőképpen írható:

append ([ ], B, B)
append ([F | K], B, [F | J]) ha append (F, B, J)

Ez a két sor egy logikai programot ad az append predikátumra. Pontosan és kimerítően definiálják, hogy logikai értelemben mit jelent két lista összefűzése.

Észrevehetjük, hogy az append predikátum rekurzív. Más szóval az „append" definíciója magát a definiálandó append-relációt is tartalmazza. A Prolog jellegzetes vonása, hogy széles körűen alkalmazza a rekurziót.

Lásd még: Prolog

MAGYARÁZÓ RENDSZER

Szakértői rendszerek

A magyarázó rendszer a szakértői rendszer szolgáltatása, amelylyel elmagyarázza a következtetési módszerét. A mai magyarázó rendszerek általában *nyomkövető* információr támaszkodnak, a konzultáció során időrendben feljegyzik a szakértői rendszer működésbe lépő szabályait. A magyarázó eszköz tehát a következtető rendszer működésének szoros megfigyelésén alapul.

Sajnálatos módon megeshet, hogy a nyomkövető információ teljesen érthetetlen, és nem emlékeztet arra, amit a felhasználó remélt. Végül is, ha egy vegyész szakértővel konzultálnánk, és meg akarnánk ismerni a gondolatmenetét, amellyel bizonyos következtetésekre jutott, olyasfajta modellt várnánk, amelyet jól használható metaforák és hivatkozások alapján értenénk meg. Az ilyesfajta információ:

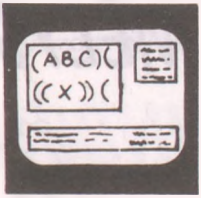
a 31. szabály szerint T = 34.6
és SU konc 56 = IGAZ
tehát bárium prec kalcium (0.7) & meghatározatlan

aligha felelne meg a várakozásainknak. Valójában a legkomolyabb szakértői rendszerek ennél már egy kicsivel jobban működnek, és magyarázataikat az angolhoz valamivel közelebb álló nyelven fejtik ki. Az viszont, hogy a rendszert még alaposabban kifaggathassuk, és feltehessük azokat a kérdéseket, amelyeket egy szakembernek tennénk fel – ez egyelőre a jövő zenéje.

Hogy még jobb magyarázóeszközöket tudjanak készíteni, a kutatók egyre alaposabban vizsgálják a szakemberrel konzultáló laikus dialógusait. A következő egy-két évtizedben még kooperatívabb rendszerek kifejlesztése várható.

Lásd még: Kooperatív rendszer

MEGCÁFOLT TAGADÁS



Programozási technikák

A logikai programozási nyelvek (mint például a Prolog) csak a logika egy részhalmazán alapulnak (a predikátum-kalkulus szarv-klauzula részhalmazán), és nincsenek bennük olyan eljárások, amelyek megállapítanák konkrét dolgok igazságát vagy hamisságát. Ilyen helyzetben a megcáfolt tagadás a második legjobb megoldás.

A megcáfolt tagadás mindenről azt feltételezi, hogy hamis, egészen addig, amíg be nem bizonyítódik, hogy igaz.

Vegyük az alábbi állítást:

X szereti az epret, ha X nem Bristolban él.

Fellapozod a noteszodat. Nem derül ki belőle, hogy John hol él. Ha tehát nincs bizonyítva, hogy John Bristolban él, az azt jelenti, hogy

John nem Bristolban él,

ebből pedig az következik, hogy szereti az epret. A megcáfolt tagadás során a „nem F" feltétel kiértékelése a „fennáll-e F" feltétel kiértékelésévé alakul át. Amíg F igazságát nem tudjuk bizonyítani, „nem F"-et tekintjük igaznak.

A megcáfolt tagadás azon a feltevésen alapul, hogy a tartományt tökéletesen le tudjuk írni a logikai axiómáival. Ez pedig a *zártvilág-feltevés* lényege.

Lásd még: Logikai programozás

MÉLYSÉG

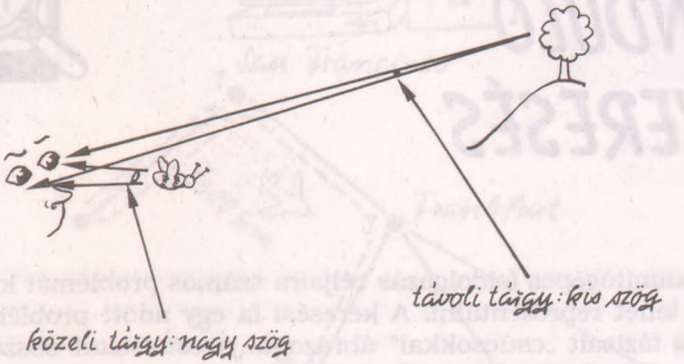
Látás

A látásban és a robottechnikában a mélység azt méri, hogy valamely dolog milyen távol helyezkedik el a kamerától vagy robottól. A gép inputjából a mélységet több módszerrel lehet megállapítani:

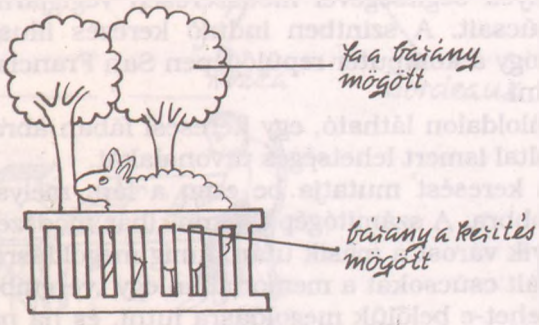
1. *Térlátás* (a. ábra). Ezt a módszert alkalmazza a szemünk (és egyes keresők is) rövid távolságok esetén. Ugyanarról a jelenetről két képet vesznek fel enyhén eltérő helyzetű kamerákkal, és a két kép különbségéből számítják ki a rajtuk látható tárgyak mélységét.
2. *Takarás* (b. ábra). Ha látjuk, hogy Charlie félig takarja Fredet, akkor tudjuk, hogy Fred Charlie mögött van. Ennek persze feltétele, hogy ismerjük Fred alakját, hogy akkor is fel tudjuk ismerni, ha egyes részletei nem láthatók. Azt is tudnunk kell, milyennek látszik Fred különböző irányokból.
3. *Méretismeretek.* Ez a módszer azon alapul, hogy a távolabbi tárgyak kisebbnek tűnnek. Ha ismerjük valaminek az abszolút nagyságát, akkor a képen látható méretéből ki tudjuk számítani, milyen távol van tőlünk. Azonban ezt a módszert is csak kisszámú jól ismert alakra lehet alkalmazni.
4. *Radar.* A radar és egyéb sugárzás (például az ultrahang) visszaverődésén alapuló rendszerek pontos távolságértékeket szolgáltatnak (persze megfelelő elektronikus kiértékelés után). Ez a módszer teljesen megbízható, és így széles körűen alkalmazzák a robottechnikában.

Számos ilyen rendszer terjedelmes számítógépes tudásbázist igényel a tárgyak alakjáról, hogy képes legyen kikövetkeztetni, a látott képek minek felelnek meg a valóságban. Ezért a valóságos jeleneteknél sokkal könnyebben megvalósíthatók egyszerű kör-

nyezetekben (például *építőkockavilágokban*, vagy azok fizikai megvalósításaiban, mindössze néhány jól meghatározott alakú tárggyal), ahol alacsony a látható elemek száma.



a



b

A mélységinformáció alapján a számítógépes látórendszerben keletkező kétdimenziós „lapos" képből egy *2½ dimenziós vázlat* készül, amely feltünteti a tárgyak távolságát a kamerától.

Lásd még: 2½ dimenziós vázlat

MÉLYSÉGBEN INDULÓ KERESÉS

Keresés

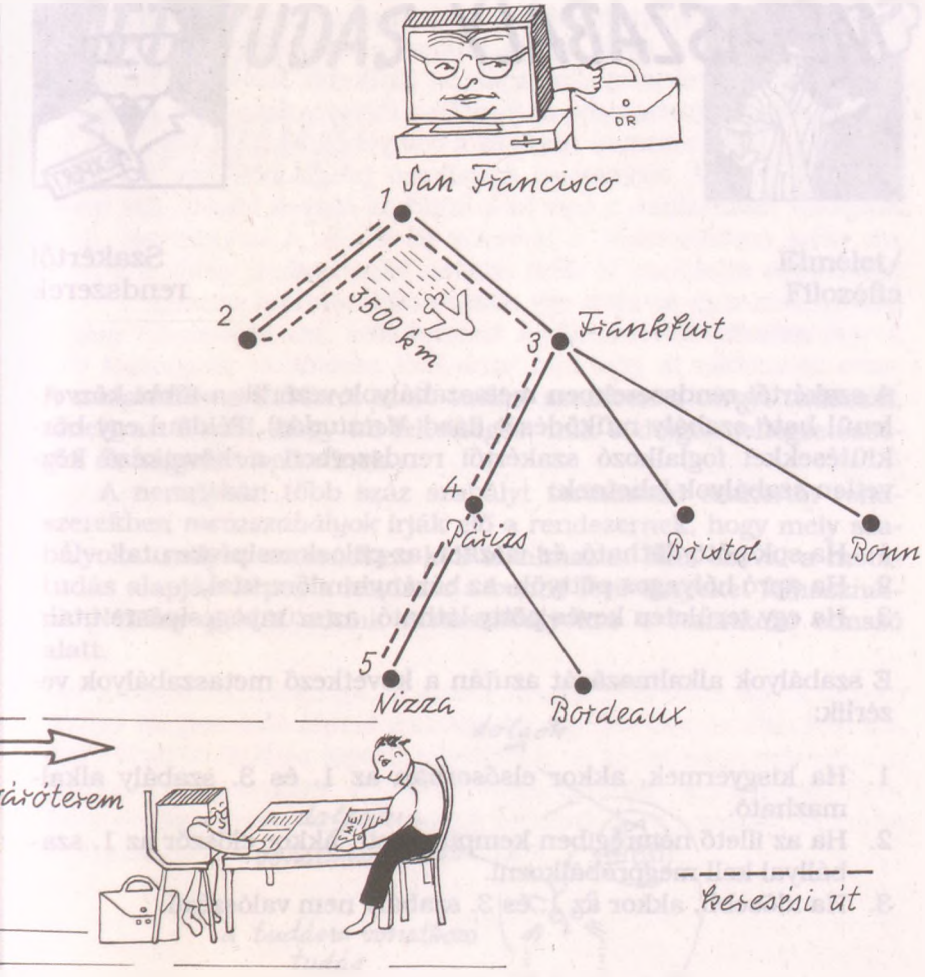
A számítógépes feldolgozás céljaira számos problémát keresési fával lehet reprezentálni. A keresési fa egy adott probléma megoldási fázisait „csúcsokkal" ábrázolja. A csúcsokat összekötő élek „operátorokat", vagyis olyan eljárásokat és tevékenységeket jelképeznek, amelyek alkalmazásával a megoldás egyik fázisából átjutunk egy másikba.

A mélységben induló, illetve *szintben induló keresés* két olyan technika, amelyek segítségével módszeresen végigjárhatjuk egy keresési fa csúcsait. A szintben induló keresés illusztrálására képzeljük el, hogy a komputer repülőgépen San Franciscóból Nizzába akar eljutni.

Amint a túloldalon látható, egy keresési fában ábrázolhatjuk a számítógép által ismert lehetséges útvonalakat.

Az ábra a keresést mutatja be ezen a fán, mélységben indulva, balról jobbra. A számítógép „vakon" (bár módszeresen) végigvizsgálja egyik várost a másik után, amíg megoldásra nem jut. Az éppen vizsgált csúcsokat a memóriában egy „verembe" helyezi, megvizsgálja, lehet-e belőlük megoldásra jutni, és ha nem adnak használható eredményt, kiveszi a veremből.

A mélységben induló keresés előnye a szintben induló kereséssel szemben többek közt az, hogy egyszerre csak korlátozott számú csúcsot kell a memóriában tartani.


DR
San Francisco
1
2
3500 km
Frankfurt
3
4
Párizs
Bristol
Bonn
5
Nizza
Bordeaux
váróterem
Keresési út

METASZABÁLY

Szakértői
rendszerek

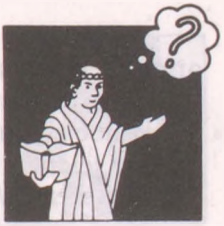
A szakértői rendszerekben metaszabályok vezérlik a többi közvetlenül ható szabály működését (lásd *Metatudás*). Például egy bőrkiütésekkel foglalkozó szakértői rendszerben a következő közvetlen szabályok lehetnek:

1. Ha sok pötty látható, és viszket, az poloskacsípésre utal.
2. Ha apró hólyagos pöttyök, az bárányhimlőre utal.
3. Ha egy területen kevés pötty látható, az szúnyogcsípésre utal.

E szabályok alkalmazását azután a következő metaszabályok vezérlik:

1. Ha kisgyermek, akkor elsősorban az 1. és 3. szabály alkalmazható.
2. Ha az illető nemrégiben kempingezett, akkor először az 1. szabállyal kell megpróbálkozni.
3. Ha idősebb, akkor az 1. és 3. szabály nem valószínű.

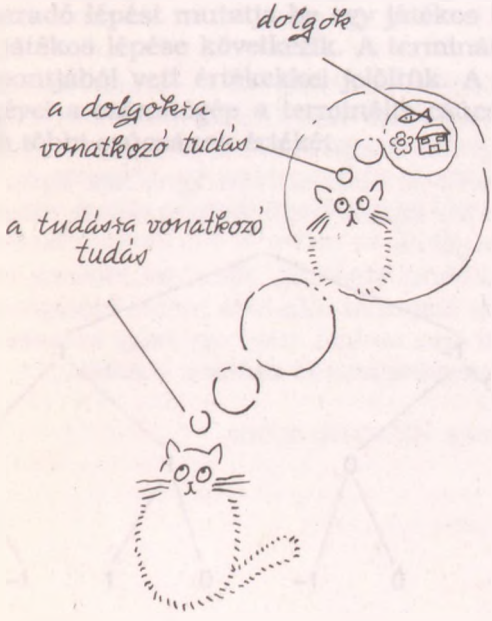
METATUDÁS



Elmélet/
Filozófia

A metatudás a tudásról szóló tudás, szemben a tárgyi tudással, amely arról szól, hogy mi lehetséges: mik a dolgok jellegzetességei, és hogyan viselkednek.

A nemritkán több száz szabályt tartalmazó szakértői rendszerekben *metaszabályok* írják elő a rendszernek, hogy mely szabályokat milyen sorrendben kell alkalmazni. Más szóval a metatudás alapján képes a rendszer a benne levő tényeket felhasználni. Néhány példát adunk metaszabályokra a vonatkozó címszó alatt.



Metanyelv. A nyelv kódok halmaza, amelyek a világot vagy annak egy részét írják le. Ily módon a metanyelv olyan kódok halmaza, amelyek a nyelvet írják le. Szűkebb, kevésbé filozofikus értelemben a metanyelv a fordítási folyamatban egy közbülső nyelvet jelent a kiinduló nyelv és a célnyelv között.

Metaterv. A tervezéssel foglalkozó programok egy közbülső állapotot használnak a terv és a kiinduló tények között. Ezt nevezik néha metatervnek: a tervezés tervének. A metatervek, de a metatudás általában is több szinten jelentkezhet, például stratégiai szinten vagy az alatt egy részletesebb taktikai szinten.

A számítástudományban használatos „metakifejezés" még a meta-assembler (a gépi kód leírásából assembler programot előállító program) és a meta-interpreter.

Lásd még: Tárgyi tudás.

MINIMAX TECHNIKA

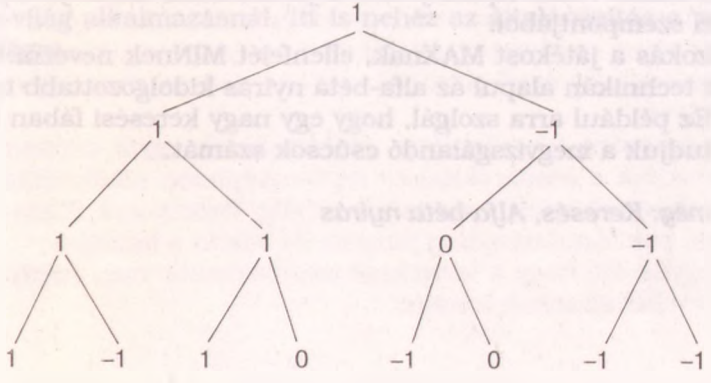
Játékok
és játék-
tartományok

A minimax technika olyan eljárás, amely egy játék fájának csúcsaihoz rendel értékeket, ezáltal növeli a keresés hatékonyságát.

Tegyük fel, hogy kitaláltunk egy játékot, amelyben az alábbi pontszámokat lehet szerezni:

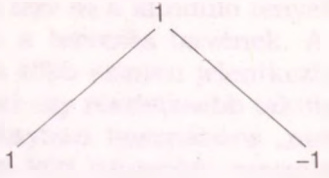
+1	győzelem esetén
−1	vereség esetén
0	döntetlen esetén

Ezt a helyzetet ábrázolja az alábbi diagram. A játék fája a három utolsó megmaradó lépést mutatja be egy játékos és ellenfele között. Most a játékos lépése következik. A terminális csúcsokat a játékos nézőpontjából vett értékekkel jelöltük. A minimax technika segítségével a számítógép a terminális csúcsokból ki tudja számítani a fa többi csúcsának értékét.

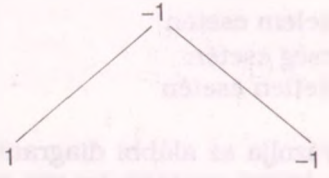


Ehhez az alábbi szabályokat alkalmazzuk.

1. Ha a játékos következik lépésre (VAGY-típusú csúcs): lásd *ÉS-VAGY fa*), akkor az érték a csúcs utódainak maximális értéke lesz.



2. Ha az ellenfél következik (ÉS-csúcs), az érték az utódok értékének minimuma.



Ha a csúcsokhoz már sikerült értékeket rendelni, ezek alapján a játékos meg tudja találni a nyeréshez vezető utat. Ehhez mindig a legnagyobb értékű csúcsba kel lépnie. Közben persze az ellenfél a legkisebb értékű csúcsba igyekszik lépni. Az alacsony értékű csúcs rossz lépést jelent a játékosnak, tehát jó lépés az ellenfél szempontjából.

Szokás a játékost MAXnak, ellenfelét MINnek nevezni. A minimax technikán alapul az alfa-béta nyírás kidolgozottabb technikája. Ez például arra szolgál, hogy egy nagy keresési fában korlátozni tudjuk a megvizsgálandó csúcsok számát.

Lásd még: Keresés, Alfa-béta nyírás

MODELL ALAPÚ LÁTÁS

Látás

A látásnak olyan megközelítése, amelynél a számítógép a kamera által látható lehetséges tárgyakból indul ki, majd ellenőrzi, hogy az ezeknek megfelelő jegyekből mi látható az elemzendő képen. Ezzel szemben az „alulról fölfelé" módszernél először azt keressük meg, milyen jegyek találhatók a képen, aztán, hogy milyen valóságos felületeknek felelhetnek meg, és végül, hogy milyen tárgyaknak lehetnek ilyen felületei. Az utóbbi módszerrel elvben bármit meg lehet találni egy képen. Ez azonban nagyon nehéz feladat. Az első módszer csak azt találja meg, amit keres, de azt sokkal gyorsabban.

A klasszikus modell alapú (vagy modellvezérelt) látó rendszert 1965-ben írták le. Egy egyszerű élkereső rendszer megrajzolja a látvány vonalas rajzát, (egy *építőkocka-világról*), majd jelzéseket keres, hogy mit is néz valójában. Például három Y alakban találkozó él jó jelzés, hogy egy kocka vagy téglatest legközelebbi sarkát látjuk. A program ekkor visszatér a képhez újabb jelzésekért, amelyek egy kocka jelenlétére utalnak. Mint minden építőkocka-világ alkalmazásnál, itt is nehéz az általánosítás a valóságos világra.

MONOTON KÖVETKEZTETÉS

Elmélet/
Filozófia

A logikai programozásban tényeket adunk a számítógépnek, rábízzuk, hogy logikailag feldolgozza őket, és levonjon minden adódó következtetést: bizonyos értelemben a logikai programozás a „gépben rejtőzködő logikusra" támaszkodik. Ha ez a „logikus" hibátlan logikai következtetést alkalmazna, a rendszer monoton lenne. A logikai programozásban azonban a következtetés eltér a klasszikus értelemben vett tiszta logikától, bizonyos tekintetben „tisztátlan" lesz. A logikai programozási rendszerek következtetési módszere tehát nem monoton.

A monoton és nem monoton következtetés közötti különbségek a nem logikus számára alig észrevehetők. A monoton következtetésben a rendszerhez hozzávett új tények soha nem kerülhetnek összeütközésbe, ellentmondásba a korábban meglevőkkel. Mindannak, ami addig logikailag bizonyítható volt, az új tények ismeretében is bizonyíthatónak kell maradnia.

A nem monoton következtetésben megengedettek az adatbázisban levő tények közötti ütközések. Ahol nagyszámú tényt használunk, több értelme van a nem monoton érvelésnek. Ezért a számítógépes rendszerek általában nem monoton következtetést alkalmaznak, és ezzel elárulják a tiszta matematikai logika bizonyos elveit.

A monoton következtetés problémáinak egyik népszerű demonstrációja a madarakkal és repülésképtelen madarakkal operál. A repülésképtelen madarat mindig Tweety-nek (Csip-csirip) nevezik, sok tanulmányban pingvin (és Nikita a barátja).

Tegyük fel, hogy a következő adatbázisunk van, amelyből logikai érveket akarunk levonni:

X tud repülni, ha X madár;
X akkor és csak akkor repülésképtelen, ha nem tud repülni;
Tweety madár;
Tweety repülésképtelen.

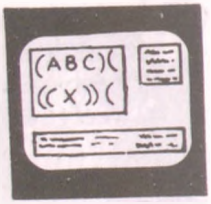
Ekkor a tiszta logika szerint az adatbázis ellentmondásos lesz. Ennek az az oka, hogy levezethető, miszerint Tweety, lévén madár, tud repülni;

Tweety tud repülni.

Nem monoton következtetés mellett azonban Tweety repülésképtelen volta nem jelent problémát.

Lásd még: Zártvilág-feltevés, Logikai programozás

N-ES



Programozási technikák

Számítógépes alkalmazásokban az *n*-es egyetlen rekordot jelent, amelyben *n* elem megadott sorrendben követi egymást. Például egy térképről leolvasott koordináta-pár is felfogható *n*-esként – speciálisan 2-esként, hiszen két elemről van szó – (párok esetében az *n*-est „rendezett párnak" szokták nevezni). Általában az *n*-es *n* elem rendezett listáját jelenti.

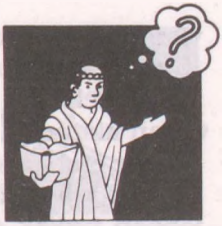
n-esek sokszor szerepelnek programok paramétereként. Ezzel a módszerrel több elemet egyetlen argumentumként lehet átadni.

Párhuzamos számítógépekhez vannak olyan programozási rendszerek, amelyeknek a programjait és adatait nem elemek lineáris listájaként adják meg (mint a hagyományos programokat), hanem listák minden sorrend nélküli együtteseként. A listák maguk *n*-esek, a teljes programot pedig úgy lehet felfogni, mint egy *n*-esekből álló „tengert", amelyből a számítógép, valahányszor szüksége van rá, programrészleteket és adatokat halászik magának.

Ha az *n*-esek gyanúsan emlékeztetnek a listákra, nem tehetünk róla. Magunk is így látjuk.

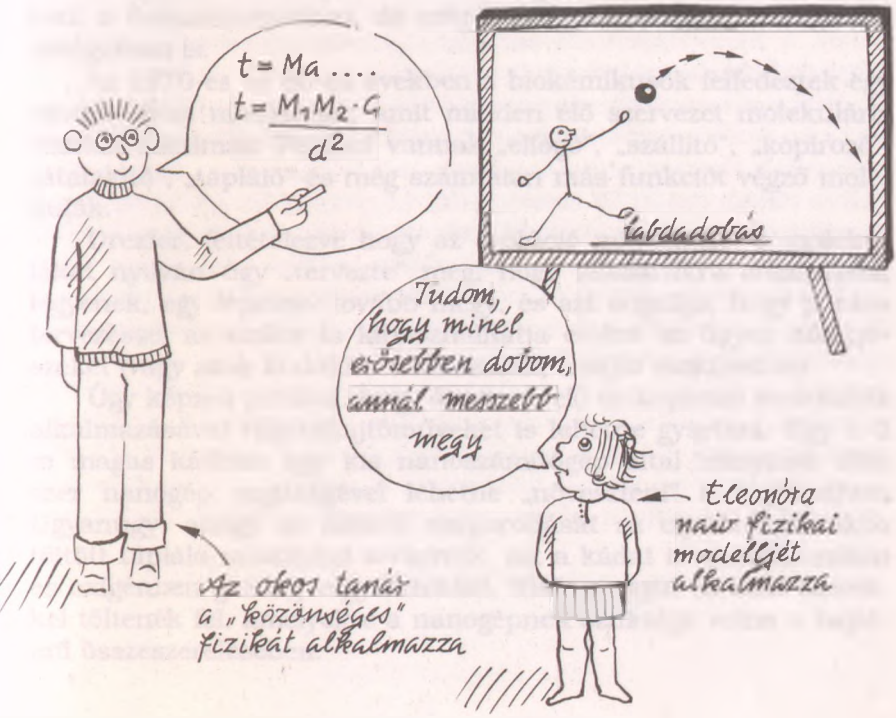
Lásd még: Listafeldolgozás

NAIV FIZIKA



Elmélet/
Filozófia

Olyasfajta fizikát jelent, amelyben nincs sok szám. Például a naiv fizika egyik szabálya az lehet, hogy ha valamit eldobunk, az leesik a földre, és minél erősebben dobjuk, annál messzebbre repül. Ezek nem kvantitatív szabályok, és ha különböző kísérletek közt összehasonlítást akarunk tenni, akkor olyasmit mondunk, hogy „nagyobb" vagy „távolabb", nem pedig azt, hogy „1.72-szerese". A

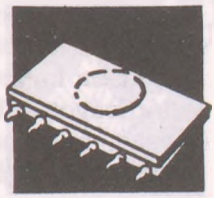


viszonylag egyszerű számrendszerek, például „0, 1, 2, sok" vagy „5-nél kevesebb", vagy „5-nél több", szintén a naiv fizika szellemének felelnek meg. Az „igazi fizika" ezzel szemben mindent pontosan kiszámol.

Az emberek sokszor a naiv fizika alapján vonnak le következtetéseket a valóságos világról. A tudásalapú rendszerek sokszor jobban tudják alkalmazni a naiv fizikát, mintsem hogy számszerűleg pontosan leírják a világot. Ez a *felhalmozott tudás* egyik válfaja, amely könnyebben használható, de kivételes esetekben, ahol az eredeti számítások és modellek feltevései esetleg már nem érvényesek, könnyen csődöt mondhat.

Lásd még: Természetes nyelv feldolgozása

NANO-TECHNOLÓGIA



Hardver

A hagyományos gépekkel ellentétben, amelyek fogaskerekekből, dugattyúkból, hengerekből és hasonló alkatrészekből épülnek fel, a nanogépek molekuláris komponensekből állnak. K. Eric Drexler Engines of Creation, the Coming Era of Nanotechnology (A teremtés motorjai, a nanotechnológia eljövendő korszaka) c. könyvében, szemét a jövőre függesztve arról ábrándozik, hogy a milliméter milliomodrészével (egy nanométer = 10^{-9} méter) mérhető méretű gépezetek milyen hatalmas fejlődést hozhatnak az MI-ben, a formatervezésben, de még a gyógyászatban és az űrtechnológiában is.

Az 1970-es és 80-as években a biokémikusok felfedeztek egy csomó olyan motívumot, amit minden élő szervezet molekuláris szinten alkalmaz. Például vannak „elfogó", „szállító", „kopírozó", „átalakító", „tápláló" és még számtalan más funkciót végző molekulák.

Drexler, feltételezve hogy az evolúció mindezeket a molekulákat nyilván úgy „tervezte" meg, hogy feladatukra alkalmasak legyenek, egy lépéssel tovább megy, és azt sugallja, hogy pontos tervezéssel az ember is kihasználhatja ezeket az ügyes alkatrészeket (vagy azok átalakított változatait) a saját eszközeiben.

Úgy képzeli például, hogy összeszerelő és kopírozó molekulák alkalmazásával rakétahajtóműveket is lehetne gyártani. Egy 1–2 m magas kádban egy kis nanoszámítógép által irányított több ezer nanogép segítségével lehetne „növeszteni" a hajtóművet. Ugyanúgy, ahogy az élesztő szaporodását az erjesztőkamrákba töltött tápláló moslékkal serkentik, ezt a kádat is alumíniumban és oxigénben gazdag vegyületekkel, fűtőanyaggal és más szerekkel töltenék fel, amelyekre a nanogépnek szüksége volna a hajtómű összeszerelésében.

Az összeszerelő molekulák pontos rend szerint helyezkednének el a kádban, és kivonnák az alumínium-oxid és egyfajta egymásba kapcsolódó szén bizonyos formáját, amely a szükséges szilárdságot és hőállóságot biztosítaná a hajtóműben. Drexler úgy képzeli, hogy az összeszerelőknek ostorai is lennének, amelyek folytonos mozgásban tartanák körülöttük a tápláló anyagokat, a tengeri szivacs sejtjeihez hasonlóan.

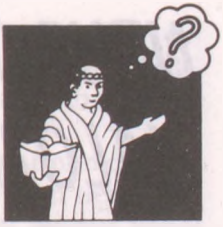
Az elkészült hajtóművet aztán lemosnák és kiemelnék a kádból. Drexler szerint „varratmentes" lenne, mint egy igazgyöngy.

Drexler emellett még a nanotechnológia számos alkalmazását el tudja képzelni. Kis nanogépekkel meg lehetne például oldani az emberi szövetek javítását. Nagyszámú nanogépet vinnének be az ember szervezetébe, amelyek lokalizálnák és eltávolítanák az elpusztult anyagot.

Az MI vonatkozásában Drexler rámutat, hogy az MI programozásához egy új tudományra van szükség: a jelenlegi technológia nem tesz lehetővé igazi MI-kutatást. Szerinte a nanotechnológusok vírus nagyságú molekulákkal tanulmányozhatnák az agy működését sejtről sejtre. Úgy gondolja, így nyerhetnénk valódi ismereteket az agy működéséről, és ez nagyban elősegítené az MI fejlődését: miután megismertük a neuronok működését, a mérnökök a nanoelektronika és nanogépek alapján hasonló eszközöket tudnának szerkeszteni.

Noha a nanotechnológia egyelőre csak spekulatív tudomány, egyre nagyobb népszerűségnek örvend a gondolat, hogy a szerves molekulák életszerű tulajdonságait kell felhasználni. Nemrégiben Julius Rebek Jr. (MIT) bejelentette, hogy sikerült egy olyan ozintriacid észter vegyületet előállítani, amely az élő anyaghoz hasonlóan az önreprodukció képességével rendelkezik. A vegyület megfog két egyszerű molekulát, olyan helyzetbe hozza őket, ahol automatikusan össze tudnak kapcsolódni, majd kibocsátja az új molekulát, önmaga másolatát.

NEUMANN-FÉLE GÉP



Elmélet/
Filozófia

Neumann-féle gép alatt voltaképpen a hagyományos számítógépet értjük, amely jellegzetesen

1. az alábbi alapvető összetevőkből áll:
   (a) vezérlő egység;
   (b) aritmetikai-logikai egység;
   (c) memória;
   (d) input és output egységek;
2. a programokat és az adatokat ugyanabban a memóriában tárolja (természetesen nem ugyanazon a helyen);
3. a vezérlő és aritmetikai-logikai egység határozza meg, hogy a számítógép a memóriából meghatározott sorrendben beolvasott utasításokból mit hajt végre legközelebb.

A program tehát a memóriában tárolt utasítássorozat, a „következő" utasítást a programszámláló jelöli ki. Mindez egyben a soros számítógép leírása, noha a gyakorlatban sok párhuzamos komputer is, különösen a tömb-vektor számítógépek több egymással kommunikáló Neumann-féle gépnek tekinthető.

Tehát majdnem minden igazi számítógép Neumann-típusú. A kivételek közé tartoznak az adatfolyamat-számítógépek (ahol a következő utasítást nem a programszámláló jelöli ki, hanem az utasítás számára rendelkezésre álló adatok), illetve a neuronhálózatok (ahol a vezérlő és aritmetikai-logikai egységek, illetve a memória funkcióját is egyetlen fajta egység, a „neuron" látja el).

Lásd még: Soros működés, Párhuzamos feldolgozás

NEURON-HÁLÓZATOK

Neuron-
hálózatok

A neuronhálózatok olyan elméleti és kutatási célokra tervezett számítógépek, amelyekben egyszerű processzoregységeket párhuzamos feldolgozást végző tömbökbe kapcsolnak össze. Az elemek az idegsejtek (neuronok) modelljére készülnek, ezért őket is neuronoknak szokás nevezni. A neuronhálózatok három típus valamelyikéhez tartoznak:

1. *biológiai modellek:* azt modellezik, ami az egyes idegsejtekben vagy azok kis csoportjaiban megy végbe;
2. *kognitív modellek:* a kognitív tudomány modelljei az emberi gondolkodásról;
3. *connection-elvű modellek:* számítástudományi modellrendszerek, amelyek az idegeken és azok viselkedésén alapulnak.

Az MI elsősorban a harmadik típussal foglalkozik.

Az egyrétegű hálózatok a legegyszerűbbek, ám alkalmazási lehetőségeik is korlátozottak. A kétrétegű hálózatokban a neuronokat „axonok" kötik össze: az egyik réteg az inputot fogadja, a másik az outputot állítja elő. Ebből a típusból a perceptron modell a legismertebb. A perceptronok képesek az alakfelismerésre, és ezzel a teljesítménnyel kevés hagyományos számítógép dicsekedhet. Viszont nem képes olyan egyszerű problémák megoldására, mint a „kizáró VAGY" (exclusive OR) kiszámítása. Ehhez ugyanis már egy harmadik rétegre van szükség az input és output között, amit „rejtett rétegnek" szokás nevezni, a benne levő neuronokat pedig „interneuronoknak".

A legtöbb neuronhálózattal kapcsolatos vizsgálatot szimulációkon végzik, párhuzamos gépeken, elsősorban transzputereken vagy közönséges (soros működésű) gépeken. Az utóbbi nagyon lassú, amikor tehát valaki azt állítja, hogy egy neuronháló-

zatos modell nagyon gyorsan megold valamilyen problémát, ezalatt azt érti, hogy alacsony az egész hálózat által megtett ciklusok vagy lépések száma. A szimulációban ugyanakkor ez sok időt vehet igénybe.

Egy neuronhálózat processzorainak – a neuronoknak – rendszerint több input vonaluk van (amelyeket az igazi neuronok analógiájára dendriteknek szoktak nevezni) és egy output vonal (amit hasonló meggondolásból axonnak hívnak), bár egy axon több dendritre is csatlakozhat. Az axon és dendrit közötti kapcsolat lehet serkentő (azaz a fogadó neuron aktivitását növelő) vagy gátló. A fogadó neuron a beérkező inputok összegétől függően diszkrét, azaz néhány fokozatú skálán leírható választ ad.

A vizsgált connection-elvű hálózatok általában az alábbi háromféle neuront használják:

1. *McCulloch–Pitts neuron.* A neuron több inputtal és egy outputtal rendelkező logikai ÉS-, illetve VAGY-kapuként való felfogását jelenti.
2. *Adaptív neuron.* Ez bonyolultabb elem, amely az inputok összegétől függő lineáris választ ad. Rendszerint alakfelismerő áramkörök alapjául szolgál.
3. *Hopfield-háló neuron.* Ilyen elemek szerepelnek a Hopfield-hálóban. Szigma-jellegű választ ad, tehát az input növekedésével az output értéke csak egy adott értékkel nő, majd stabillá válik, további inputnövekedés már nem érinti.

A neuronhálózatok két okból is érdekesek az MI számára. Először is nagymértékben párhuzamosított komputerek, és így képesek felgyorsítani a számításigényes feladatok megoldását; másfelől alternatívát kínálnak az MI hagyományos algoritmusvezérelt, soros feldolgozású számítási módszereihez képest. Nagy lehetőségek rejlenek benne egyes, hagyományos módon gyakorlatilag kiszámíthatatlan problémák, például alakfelismerési feladatok megoldására. Speciálisan a neuronhálózatok saját struktúrájuktól, és nem egy kívülről beadott programtól függően képesek „megtanulni" egyes problémák megoldását.

A neuronhálózatok eddig két problématípus megoldásában érték el a legnagyobb sikereket: a nagymértékben összefüggő problémáknál és az osztályozási problémáknál. Az összefüggő problémák azok, amelyeknél az egyik részlet megoldása szorosan összefügg más részletek megoldásával. Erre példa *Az utazó ügynök* és a *feladatkiosztás* problémája.

Az osztályozási problémáknál egyedi tárgyakról kell eldönteni, hogy több osztály közül melyikbe tartoznak. A neuronhálózatot az összes osztályból vett mintákkal „betanítják", majd adnak neki egy mintasorozatot, amelyet egy osztályba kell besorolni. Neuronhálózatok segítségével osztályoztak emberi arcokat (a WIZARD gép ezt valós időben képes elvégezni), speciális elektronikával azonosították az „ismerős" arcokat), népszámlálási adatokat és futószalagon érkező alkatrészeket.

Lásd még: Connection-elv, Feladatkiosztási probléma, Az utazó ügynök problémája

NP

Általános
számítógépes
kifejezés

NP a „nem determinisztikus polinomiális" kifejezés rövidítése, és problémák (még pontosabban megoldások) bizonyos típusára utal. Túlságosan nagy NP-problémák esetén a megoldás rendkívül nehézzé válik.

A „polinomiális" kifejezés a probléma megoldásához szükséges időre vonatkozik. Tegyük fel például, hogy egy listában a legnagyobb számot kívánjuk megkeresni. Az ehhez szükséges idő a lista hosszával arányos. Ha úgy akarjuk csökkenő sorrendbe rendezni a lista elemeit, hogy a legnagyobb elemeket sorban egymás után kivesszük, ez a lista hosszának négyzetével arányos időt igényel. Ezek P-problémák: a megoldást egy polinommal arányos időben meg lehet találni.

Melyek a nem P-problémák? Nagyon súlyos a helyzet az exponenciális problémáknál, ahol a megoldáshoz szükséges idő a probléma méretének exponenciális függvényével arányos. Egy exponenciális függvény mindig gyorsabban nő, mint egy polinom, tehát elég nagy problémáknál a megoldás több időt vesz igénybe. Az exponenciális problémák egyik csoportjába a kombinatorikai robbanás által „sújtott" keresési problémák tartoznak.

A kettő között „félúton" vannak az NP-problémák. Ezek nem oldhatók meg polinomiális időben, viszont ha van egy megoldás, annak helyessége már polinomiális időben ellenőrizhető. Jó példa egy lista elemeinek növekvő sorrendbe rendezése. Ehhez a lista hosszának exponenciális függvényével arányos idő kell. Hogy ellenőrizzük, jó sorrendben van-e már, ahhoz viszont elég minden elemet egyszer megnézni, és meggyőződni, nagyobb-e az előzőnél.

Hogy egy probléma P vagy NP, az a megoldáshoz választott algoritmustól is függ. Így egyes NP-problémák valójában P-problémák is lehetnek, ha a megfelelő algoritmust választjuk hozzá.

Azt a problémát, amely algoritmustól függetlenül mindig NP, *NP-teljes* problémának nevezzük. *Az utazó ügynök problémája* például NP-teljes. Az ehhez kapcsolódó másik kifejezés, az *NP-nehéz*, a probléma méretének növekedésével a megoldás idejének gyors növekedésére vonatkozik: *Az utazó ügynök problémája* egyúttal NP-nehéz is.

Az NP-problémákat nem mindig nehéz megoldani. Sokszor csak rendkívül nagy méretű problémáknál tart sokáig a megoldás: kis változatnál gyors megoldás is létezhet. Ezenkívül lehet gyors közelítő vagy heurisztikus megoldás: az MI szerepe gyakran az ilyen közelítő megoldások megtalálásában van.

Lásd még: Algoritmus, Kombinatorikai robbanás

OBJEKTUM-ORIENTÁLT PROGRAMOZÁS

Programozási technikák

Az objektumorientált programozás először a Xeroxban kidolgozott SmallTalk nyelvvel jelent meg, az utóbbi években pedig egyre növekvő népszerűségre tett szert mint az információ strukturálásának és ábrázolásának módszere. MI-gondolatokra és technikákra épülő alkalmazások tömegében használják.

A nem programozó számára az objektumorientált programozás mindig egy kicsit titokzatos marad: megpróbáljuk tehát röviden elmagyarázni a lényegét.

A programozó előtt álló egyik nagy feladat: hogyan kell jól összeszervezni az adatokat és a velük dolgozó eljárásokat. A régebbi típusú programnyelvek többsége (például a Fortran) a „spagettikód" elnevezésű gyengeségben szenvedett. Az ilyen nyelveken írt programokban annyi „go to" utasítás volt, ami különböző helyekre ugratta a programot, hogy a kód egészét szinte lehetetlen volt megérteni, és így módosítani.

Ennek orvoslására korlátozott elágazási struktúrákat vezettek be, például

If ... then ... else ...
Do ... until ...
While ... do the following ...

hogy könnyebben lehessen követni, mi történik. A Pascal és Algol 68 példa azokra a „strukturált programozást" elősegítő nyelvekre, amelyek a 70-es években voltak népszerűek.

Az objektumorientált programozás újfajta szemléletet hordoz. Ennek központi fogalma az „objektum", egy olyan kódrészlet, amely mindazt tartalmazza, amit egy programban szereplő tevékenységgel vagy elemmel kapcsolatban fel akarunk használni. Készíthetünk például egy „dolgozó" objektumot a személyzeti

osztály számára írt programban. Ennek az objektumnak azután attribútumai lennének, például a lakcím vagy a beosztás. Ezután már könnyű az általános objektum új „egyedeit" létrehozni (például az egyes konkrét dolgozókat), és ezek az utódok öröklik a szülői attribútumokat.

Olyan új dolgozócsoportokat is létrehozhatunk, mint például a számítógépes osztály dogozói.

Egy objektum azután nem pusztán adatokból áll, hanem „metódusok" is szerepelnek benne. A metódus eleve azzal a képességgel ruházza fel az objektumot, hogy „üzenetekre" tudjon reagálni – akár a felhasználótól, akár más objektumoktól érkezik. Például egy üzenettel megváltoztathatjuk egy dolgozó beosztását vagy lakcímét. Egy másik üzenet arra szolgálhat egy bizonyos objektumnál, hogy egyes attribútumok értékét jelezze ki a felhasználónak.

Hogy elképzelést alkothassunk az objektumorientált megközelítésről, íme egy elképzelt program kis (egyszerűsített) részlete.

Definiáld a dolgozó objektumot mint a személy objektum egy fajtáját

Minden dolgozó objektumnak van

neve, alapértelmezés szerint Joe Bloggs;
beosztása, alapértelmezés szerint Mérnök;
fizetése, alapértelmezés szerint £10,000;
dolgozói besorolása, alapértelmezés szerint 0.

Ha én dolgozó objektum vagyok, és egy üzenetben azt kérdezik tőlem, hány nap szabadságom van, megnézem, hogy a dolgozói besorolásom nagyobb-e, mint 5.
Ha igen, válaszom „20 nap", ellenkező esetben „18 nap".

Figyeljük meg a következőket:

1. A „dolgozó objektum"-nak vannak attribútumai.
2. A „dolgozó objektum" attribútumokat örökölhet az általánosabb „személy objektumtól".
3. A „dolgozó objektum" képes válaszolni az attribútumaira vonatkozó kérdésekre.

Az objektumorientált programozásban kétféle üzenetet szoktak használni. Ha a program *szinkronüzenetet* ad ki, akkor türelmesen be is várja a választ. Ez megfelel a hagyományos programnyelvek eljáráshívásának, a program egy eljárást elindítva csak akkor folytatja a saját működését, ha az befejezte a dolgát. Az *aszinkron üzenet* olyan, mint amikor postára adunk egy levelet. Elküldjük az üzenetet a címzetthez, és azt várjuk, hogy egy idő múlva válasz érkezik. Ezt a fajta üzenetet használják az osztott üzemmódú rendszerek (hálózatok), ahol az üzenetek azonnali feldolgozása nem volna praktikus.

A gyakorlatban többféle objektumorientált programozási nyelv létezik, köztük az Objective C, C++ és a SmallTalk.

OPERÁTOR

Általános számítógépes kifejezés

Az „operátor" szó az MI számos területén felbukkan, többnyire valamilyen matematikai értelemben. Az operátor meghatározza, hogy valamilyen adatot hogyan kell feldolgozni, azaz hogyan kell valamilyen műveletet végrehajtani rajta. Például a „+" operátor azt jelenti, hogy „adjunk össze két számot". Az „in" vagy „∈" („eleme") operátor azt teszteli, hogy egy elem szerepel-e egy listában vagy halmazban. Például

„kutya *in* [kutya macska egér]" kifejezés logikai értéke: *true*.

A téroperátor (látórendszerekben) a digitalizált képekre alkalmazható az élkeresés céljára. Az MI-n kívül a matematika és a programozás számos területén szerepelnek operátorok.

OPTIKAI ÁRAMLÁS

Látás

Az optikai áramlás a kép mozgását ábrázoló módszer. A számítógépes látórendszerek általában feltételezik, hogy a kép változatlan marad a feldolgozás idejéig. A képen szereplő objektumokról azonban, akárcsak azok mozgásáról, információt lehet szerezni abból is, ahogy a kép változik. A mozgást folyamatosnak tekintjük, és a kép minden pontjáról feltételezzük, hogy az átáramlik egy másik pontba. Ezt gyakran egy oszlopsorozat segítségével képzeljük el, ahol az oszlop iránya a környezetében levő pontok elmozdulási irányát jelzi, hosszúsága pedig az elmozdulás sebességét.

OSZTOTT FELDOLGOZÁS

Hardver

Az osztott feldolgozás olyan számítás, amelyet egyidejűleg több számítógép végez, tehát a párhuzamos feldolgozás egyik formája. A lényeges különbség párhuzamos és osztott feldolgozás között az, hogy a számítás az utóbbinál több olyan processzor között oszlik meg, amelyek teljesen különböző dolgokat végeznek. Ezek ily módon egészen bonyolult részei az egésznek, amelyek önmagukban is képesek programokat futtatni. A legegyszerűbb, osztott feldolgozásra alkalmas processzor talán a Connection Machine processzora. A legösszetettebb egy nagyszámítógép is lehet. Nemrég olyan kísérletet végeztek az USA-ban, amelyben egy százjegyű szám prímtényezőkre bontását 400 önálló gép végezte.

Az osztott processzorok a hálózatok különleges fajtáját jelentik. Ezzel szemben nem minden hálózat áll osztott processzorokból. A fő különbség, hogy az osztott feldolgozás áttetsző a felhasználó szempontjából, tehát az egyik processzor felhasználójának nem kell azzal törődnie, hogyan kapcsolódik a többiekhez.

Ugyanúgy, ahogy az osztott feldolgozás egy speciális hálózatot jelent, a párhuzamos feldolgozás is tekinthető egyfajta speciális osztott feldolgozásnak, ha maguk a processzorok nem azonosak, ám – legalábbis a SIMD (egyetlen utasítás, különböző adatok) gépeknél – ugyanazokat az utasításokat hajtják végre.

A neuronhálózatok ismét más értelemben tekinthetők osztott processzoroknak. Nem lehet eldönteni, hogy egy neuronhálózatban *melyik* neuron melyik számítást végzi, maga az algoritmus nem tesz különbséget az egyes részek között. A teljes hálózat végzi az egész számítást, egyik neuronra sem tudunk rámutatni, hogy valamelyik műveletért felelős lenne. Így a program mellett magát a számítást is megosztjuk a nagyszámú processzor között

Lásd még: Neuronhálózatok, Párhuzamos feldolgozás

ÖTÖDIK GENERÁCIÓS PROJEKTEK

Finanszírozás

Szervezetek és tevékenységek sokasága nőtt ki az 1981-ben elindított japán Ötödik Generációs Projekt kezdeményezésből, amely mesterséges intelligencia felhasználásával egy új számítógép-generáció létrehozására irányult. Az „ötödik generáció" a komputerek fejlődési fokozataira utal, amelyek közül az első generáció elektroncsöves, a második tranzisztoros és ferritmemóriás, a harmadik integrált áramkörös és a negyedik a 32- és 64-bites VLSI (nagyfokú integráltságú) áramkörös komputereket jelentett.

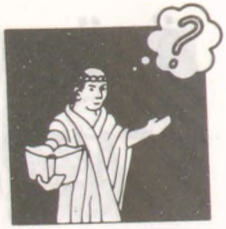
Az ötödik generációs számítógépet eddig még nem építették meg. Az elképzelés egy természetesen kezelhető, a köznapi nyelvet jó értő, az emberhez hasonló fokú következtetésre képes komputerről szól, amely a bizonytalansággal és a hiányos információval is képes lenne megbirkózni.

A japán program tíz évre terjed, és Japánban szokásos módon nagyvállalatok egy csoportjának kell végrehajtania a Kereskedelmi és Ipari Minisztérium ellenőrzésével, a kormány viszonylag csekély pénzügyi támogatása mellett. A projekt 1988-as áttekintése arról számolt be, hogy a fejlődés lassabb a vártnál; egyes célokat a tervezettnél sokkal lassabban lehet elérni, másokat valószínűleg soha. Többek között továbbra is távoliak a kilátások az Ötödik Generációs következtető gépek működését lehetővé tévő tudásbázis felépítésére.

1991-ben a japánok az Ötödik Generáción is túllépő program megvitatásának javaslatával álltak elő. Ez a program neuronhálózatokra koncentrálna, amelyek magas fokon párhuzamosított számításokat tesznek lehetővé. Informálisan természetes módon ezt szokták Hatodik Generációnak nevezni.

Lásd még: ESPRIT, ALVEY

PARADIGMA



Elmélet/
Filozófia

A paradigma egy séma a világ értelmezésére, tágabb értelemben pedig a világszemlélet és magyarázat módszereinek gyűjteménye. Tudományos körökben Thomas Kuhn tette népszerűvé a kifejezést, aki arra keresett magyarázatot, hogyan lehetséges, hogy a tudósok különböző generációi ugyanazt a világot drasztikusan különböző elméletekkel magyarázták. (Kuhn szerint éppenséggel nem is ugyanabban a világban éltek, akkor viszont nehezen érthető, hogyan járhattak például ugyanabba az étterembe.) A „paradigma" szónak van egy „tág" és egy „szűk" értelme is.

Jelentheti elméletek és kísérleti bizonyítékok teljes, egymást kölcsönösen alátámasztó rendszerét, amelybe új tapasztalatokat is bele lehet illeszteni (egyes számú paradigma). Például a számítógép-tudomány nagyrészt az úgynevezett „információ-elméleti paradigma" keretei közt működik, amely szerint valamilyen tároló-rendszerben (például számítógépben vagy könyvben) az információ jelek sorozatát jelenti, amely egyszerre hordozhat leíró információt és utasításokat, és amelyet más szimbolikus utasításokkal feldolgozhatunk, például számunkra kényelmesebb szimbólumokra fordíthatunk le. Ez talán mind magától értetődőnek tűnik, de csak azért, mert ezt a paradigmát már a múlt század közepe óta ismerjük. Isaac Newton szemében még különösnek tetszett volna, és a connection-elv hívei ma is gyanakvással szemlélik a szimbólumok és a velük végzett műveletek túlzott hangsúlyozása miatt.

A második jelentés (kettes számú paradigma) az elsőhöz kapcsolódik. Az első típusú paradigmára korszakalkotó kísérletek jellemzőek, önmagukban is megannyi „paradigma", amelyek az első típusú paradigma lényegi gondolatait forradalmian új megvilágításba helyezik. A számítógép-tudományban a második típusú paradigma két példája a Charles Babbage által a XIX. század

legvégén tervezett és sokak által a mai elektronikus változat előfutárának tekintett mechanikus számológép, a Babbage-gép, illetve a Turing-gép. Ezek olyan „kísérletek", amelyek azt bizonyítják, hogy minden számítás egy géppel végrehajtható diszkrét utasításokra vezethető vissza, és hogy minden digitális számítógép egyenértékű. A kettes számú paradigma tehát speciális kísérlet, az egyes számú paradigma pedig gondolatok, kísérleti eredmények és elméletek egész gyűjteménye, amelyek értelmezik a kettes számú paradigma kísérleteit, de ugyanakkor alátámasztást is nyernek általuk.

A kettes számú paradigmának tulajdonképpen nem is kell *működnie*. A Babbage-gép például soha nem működött ténylegesen: meg sem építették. Csak példaként szolgált egy probléma kezeléséhez.

A „szindrómához", „struktúrához" és sok egyéb szakkifejezéshez hasonlóan a „paradigma" kifejezést is réges-rég elcsépelték. Ne dőljünk be azoknak, akik unos-untalan alkalmazzák az alábbi értelemben:

1. *Példa*	Csak néhány alapvető példa tartozik a kettes típusú paradigmához
2. *Elmélet*	Elméletek és alappéldák egész gyűjteménye *esetleg* paradigmát alkothat. Önmagában egy elmélet azonban még nem paradigma: egy fecske nem csinál nyarat.
3. *Gondolat vagy hipotézis*	Lásd előbb.
4. *Szemléletmód*	A paradigma sokkal több, mint *egyszerűen* nézőpont. Jobb szinonima az „előítélet".
5. *Saját legújabb munkám*	A szó leggyakoribb használata. Halljuk eleget.

PÁRHUZAMOS FELDOLGOZÁS

Hardver

A párhuzamos feldolgozás olyan programok futtatását jelenti, amelyek egy időben több műveletet hajtanak végre. A hagyományos (*soros*, vagy szekvenciális) gépek csak egymás után tudják végrehajtani az utasításokat.

A párhuzamos feldolgozásnak négyfajta stílusa ismeretes:

1. vektorfeldolgozás;
2. tömbfeldolgozás;
3. osztott feldolgozás;
4. neuronhálózatos feldolgozás.

A *vektorfeldolgozásban*, például a Cray számítógépeken több processzor dolgoz fel egyszerre egy-egy adat-elemet. A műveletet (például egy mátrix két elemének összeszorzását) egyszerűbb lépések sorozatára bontják, és minden processzor egy lépéssel foglalkozik, ugyanannak a számításnak egy különböző részletével. Tehát a processzorok párhuzamosan működnek, különböző műveleteket hajtanak végre ugyanazokon az adatokon. A vektorprocesszorokat gyakran használják hagyományos processzorok kiegészítőjeként.

A tömbfeldolgozásnál a számítógépben egy csomó processzor van, amely különböző adatokon ugyanazt a műveletet hajtja végre. Tegyük fel például, hogy 64 különböző számhoz szeretnénk 17-et hozzáadni. A tömbprocesszor minden processzorát arra utasítaná, hogy az első számhoz, amivel találkoznak, adjanak 17-et, majd mindegyiknek valóban adnak is egy számot. Ezt nevezik SIMD (single instruction, multiple data = egyetlen művelet, több adat) módszernek. Az ICL DAP (Distributed Array Processor = Osztott Tömb-Processzor) nevű gépe például egy négyprocesszoros tömbfeldolgozó gép volt.

A vektor- és tömbfeldolgozás igen közel áll egymáshoz, így hasonló feladatok megoldására is használják őket.

Osztott feldolgozásról olyankor beszélünk, amikor a különböző processzorok különböző dolgokat csinálnak, de mégis összhangban működnek együtt egy nagyobb probléma megoldásában. Ezt MIMD (multiple instruction, multiple data = több utasítás, több adat) programozásnak szokták nevezni. A transzputerek és a *Connection Machine* egyaránt MIMD-gépek.

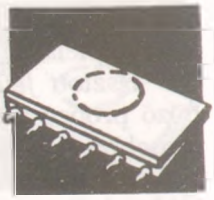
Szokás még *közös memóriájú processzorokról* beszélni, amelyeknél több processzor osztozik ugyanazon a memórián (ilyenek például a vektorprocesszorok); valamint *osztott memóriáról*, ahol mindegyik processzornak megvan a maga memóriarésze, és a többi processzorokkal csak korlátozott kommunikációs csatornákon keresztül kommunikál (ide tartoznak az osztott processzorok).

Az eddigi párhuzamos processzoroknak van egy irányító főprocesszora, amely programozza és adatokkal látja el a többit. Az utolsó osztott processzor-típusnál, a *neuronhálózatoknál* viszont nem ez a helyzet.Ez is egyfajta osztott feldolgozás, de radikálisan különbözik a hagyományos feldolgozástól, amennyiben nincs központi vezérlő, sőt gyakran még program sem.

A párhuzamos feldolgozás két okból is fontos az MI szempontjából. Először is, bizonyos MI problémák megoldása óriási számítási időt igényel. Ha ezt sikerül „belegyömöszölni" egy párhuzamos feldolgozású megoldásba, akkor nagyon sok futási időt meg lehet spórolni. Másrészt vannak olyan problémák, amelyeket jobban meg lehet oldani párhuzamos módszerekkel. Például a látás egy olyan tevékenység, ahol az egész képet egy időben kell feldolgozni, és az egyik részlet értelmezése a másik értelmezésétől függ. Például egy csomó „szőrös" vonal értelmezése attól függ, hogy egy kutyához vagy egy bothoz csatlakoznak-e, a kutya felismerése viszont attól is függ, hogy látjuk-e a farkát. Az ilyesmit hatékonyan lehet kezelni párhuzamos feldolgozással. Vannak formálisabb problémák is, mint például *Az utazó ügynök problémája*, amelyekkel szintén ez a helyzet: azt mondjuk, ezek nagymértékben *összefüggő* problémák.

Lásd még: Connection-elv, Osztott feldolgozás, Neuronhálózatok

PÁRHUZAMOS KÖVETKEZTETŐ-GÉP



Hardver

A párhuzamos következtetőgép a japán ötödik generációs komputer fejlesztés alatt álló központi komponense. Ennek egy párhuzamos gépnek kell lennie, 1000 processzorral, amelyek csoportokba vannak osztva, ahol egy-egy csoport közös memóriát használ, egymással pedig egy külön csoport által szervezett hálózaton keresztül kommunikálnak. Ezáltal a gép szükségképpen gyorsabban működik, mert a fő processzorok nem töltenek időt számukra lényegtelen információk továbbításával.

A processzorokat úgy tervezték, hogy a logikai és szimbólumkezelő műveletek legyenek könnyen programozhatók, ne pedig a numerikus számítások. A PIM (personal inference machine = párhuzamos következtetőgép) a bizonytalan adatok ábrázolását is lehetővé tenné.

A PIM előfutára a PSI (personal sequential inference machine = személyi soros következtetőgép) a PIM PC-n működő változata, amelynek a személyi számítógép szintjén következtető és logikai képessége van. A PSI nem párhuzamos számítógép, de a processzora hasonlít arra, amelyik a PIM alapjául szolgál.

A PIM-re és PSI-re új nyelvek és operációs rendszerek is fejlesztés alatt állnak.

A PIM és más, potenciálisan MI-re specializálódott gépek sebességét a hagyományos számítógéptől eltérően nem az egy másodperc alatt végrehajtható aritmetikai műveletek, hanem a logikai következtetések számával mérik, egysége a LIPS (logical inference per second = logikai következtetés másodpercenként) és KLIPS (= kiloLIPS).

PERCEPTRON

Neuron-
hálózatok

A perceptron egyszerű kétrétegű neuronhálózat, amely a retina működését modellezi. A perceptront az 50-es évek végén és 60-as évek elején tanulmányozták. Meg lehetett tanítani minták felismerésére, ami a hagyományos komputereknek nagyon nehéz feladat.

Minsky és Papert azonban kimutatta, hogy vannak olyan problémaosztályok, amelyeket perceptronnal elvileg sem lehet megoldani. Ezt a következtetést minden neuronhálózatra általánosították, és ebből azt a konklúziót vonták le, hogy a neuronhálózatok nem játszhatnak fontos szerepet a számítógép-tudományban. A neuronhálózatos modellek iránt újabban ismét megnövekedett érdeklődés annak a széles körű felismerésnek köszönhető, hogy ez az általánosítás nem helytálló, ugyanis a „rejtett réteggel" rendelkező neuronhálózatokra nem érvényes.

Lásd még: Neuronhálózatok

PRAGMATIKA

Természetes
nyelv
feldolgozása

A természetesnyelv-feldolgozásban a pragmatika azt tanulmányozza, hogy az emberek hogyan fejezik ki közlendőiket. Ha valakinek azt mondjuk: „Te hájas disznó!", ezt különböző szinteken lehet elemezni:

1. magukat a szavakat (te, hájas, disznó);
2. a szavak grammatikai szerepét a mondatban (hájas: melléknév, disznó: főnév);
3. a mondat szemantikáját; a benne kifejeződő gondolatokat (hogy az illető egy hájas disznó);
4. a mondat mögött rejlő, azt irányító koncepciót (inzultus, a beszélő támadni akar).

Hogy pragmatikai értelemben is megértsük a „Te hájas disznó!" mondatot, a természetesnyelvfeldolgozó rendszernek nemcsak a nyelvtanról és a szótárról kell tudással bírnia, de sok információja kell legyen magáról a világról is. Ezt a fajta tudást persze nagyon nehéz „betáplálni a komputerbe".

Lásd még: Természetes nyelv feldolgozása

PREDIKÁTUM

Logika

A predikátum kifejezés eredetileg a matematikai logikából származik, ennek gondolatát azonban a programnyelvek is kiterjedten használják az MI alkalmazásokban.

A predikátum olyan függvény, amelynek értéke vagy IGAZ (TRUE) vagy HAMIS (FALSE). Például tekintsük az alábbi Prologban írt kérdést:

?szereti (mackó, méz)

Ami azt jelenti:

Teljesül-e a „szereti" predikátum a mackó és méz argumentumokkal?

A Lispben vannak olyan függvények, amelyek TRUE vagy FALSE értéket adnak vissza, tehát predikátumok. Vegyük például a LESSTHAN (KISEBBMINT) predikátumot. Ez két számot hasonlít össze. Ha az első kisebb, mint a második, akkor a predikátum teljesül, és a Lisp a T (True), egyébként pedig a NIL (SEMMI – jelen esetben HAMIS) értéket adja vissza.

Minden Prolog szabály predikátum (noha nem túl gyakran használjuk őket ebben az értelemben).

Lásd még: Prolog, Logikai programozás

PREDIKÁTUM-KALKULUS

Logika

A predikátum-kalkulus olyan eszköz, amelynek révén a világra vonatkozó gondolatokat a matematikai képletek precizitásával fejezhetjük ki. Az elsőrendű logika egyik fajtája, és mint ilyen, bonyolultabb, mint rokona, a kijelentés-kalkulus.

Hogy megértsük a predikátum-kalkulus lényegét, gondoljunk először a kijelentés-kalkulusra és annak korlátaira. Ezzel a módszerrel egyszerű kijelentések igazságát és hamisságát tudjuk kiszámítani. A kijelentés-kalkulusban egyszerű igen-nem típusú kijelentések alkotják ismereteink alapjait, és ezekből kiindulva vezethetjük le más kijelentések igazságát-hamisságát.

Kijelentés-műveletek használatával (lásd *kijelentés-kalkulus*) egyszerű kijelentésekből összetett kijelentéseket alkothatunk:

John szereti a szirupot ÉS John szereti a mézet ÉS Johnnak van kocsija.

Azonban az összetett kijelentéseknél is a komponensek igazságán vagy hamisságán van a hangsúly: a kijelentés-kalkulus logikája ugyanis korlátok közé szorítja, hogy mit lehet kifejezni és mit nem. Az egyik nyilvánvaló korlát, hogy nem írhatunk szabályokat és tételeket. Ennek az az oka, hogy nem használhatunk ún. kvantorokat. Nem mondhatjuk, hogy ez és ez teljesül „minden elemre" egy adott osztályban. Nem mondhatjuk például, hogy

Minden John nevű ember szereti a szirupot.

A predikátum-kalkulus ezzel szemben már *megengedi* változók és kvantorok használatát. A kibővített logika formuláiból kijelentés-műveletekkel alkotott „mondatok" segítségével már sok gondolatot meg tudunk fogalmazni.

Például:

$\forall x$.Macska(x) → VanBajsza(x).

A világ minden x individuumára érvényes, hogy ha x macska, akkor x-nek van bajusza. $\forall$ az *univerzális kvantor*, ami azt fejezi ki, hogy a kijelentés igaz lesz, bármilyen individuum nevét helyettesítjük a változó helyébe.

$\exists x$. Macska(x) and Fekete(x) and Mohó(x).

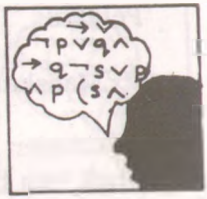
Van olyan x, amely macska, fekete és mohó.

$\exists$ az *egzisztenciális kvantor*, amely azt jelzi, hogy vannak olyan individuumok, amelyekre a kijelentés igaz.

A predikátum-kalkulus népszerű a logikusok körében, ám ha a valóságos gondolatokat megpróbáljuk lefordítani rá, rendkívül hosszú és érthetetlen kifejezéseket kapunk. Emiatt a predikátum-kalkulust ritkán alkalmazzák „tiszta" formában. Ennek ellenére ez alkotja számos olyan MI-rendszer alapját, amely logikai alapon ábrázolja a tudást. Ilyenre példa a STRIPS rendszer, és a logikai programozási nyelvek, köztük a Prolog.

Lásd még: Kijelentés-kalkulus

PROCEDURÁLIS PROGRAMNYELV



Logika

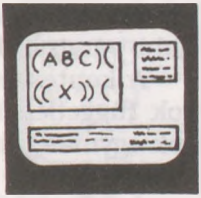
Egy procedurális (eljárásokra alapozott) programnyelv „utasításoknak" vagy „parancsoknak" nevezett eljárások és tevékenységek sorozatából épül fel. Például:

X-et szorozd meg Y-nal.
Ellenőrizd ezt grammatikailag.
Ezt a szót vedd hozzá a listához.

A procedurális programnyelvekkel szemben egy *deklaratív programnyelv* programjai ténykijelentésekből állnak, és futás közben a program ezekből következtetéseket von le.

Lásd még: Deklaratív programnyelvek

PRODUKCIÓS SZABÁLY



Programozási technikák

A „produkciós szabály" kifejezést elég kiterjedten használják az MI-ben tudásreprezentációs rendszereknél. A produkciós szabály egy feltétel–tevékenység párból áll. Például:

If robbanás esélye nagyobb, mint 1:1 000 000 ...
then ... következmény-listába beírni: „rendkívül veszélyes".

A produkciós szabálynak van úgynevezett „bal oldala" és „jobb oldala". Ha a bal oldal feltételei teljesülnek, a szabály működésbe lép, és a megfelelő tevékenység végrehajtódik.

A MYCIN szakértői rendszer közismert a produkciós szabályok alkalmazásáról. A MYCIN a vér betegségeivel foglalkozik. Ahogy az alábbi (egyszerűsített) szabályból is láthatjuk, a MYCIN produkciós szabályai Lispben íródtak. A szabály egy premisszából (alkalmazási feltételből) és a végrehajtandó tevékenységből áll.

```
PREMISSZA:      (AND (KONTEXTUS AZONOS HELY VÉR)
                     (KONTEXTUS AZONOS SZENNYEZŐDÉS
                                GRAMM_NEGATÍV)
                     (KONTEXTUS AZONOS MORFOLÓGIA PÁL-
                     CIKA)
TEVÉKENYSÉG:    (KONKLÚZIÓ (KONTEXTUS AZONOS ECOLI
                     MÉRTÉK 0.6))
```

Amikor a szabály működésbe lép (nevezetesen, ha a betegség helye a vér, a baktérium gram-negatív, és pálcika alakú), a (KONTEXTUS) kontextus helyébe az új információ lép. Ebben az esetben a baktérium azonosítója is felkerül a listára, és azt is jelzi, hogy mi a helyes azonosítás esélye.

A kontextuslista tehát a beteg minősítését tartalmazza, és az újabb vérvizsgálatokkal változik.

A kontextuslisták használatával az egyes produkciós szabályok függetlenek maradnak egymástól. Ha a produkciós szabályok egymásra hivatkoznak, például

A if B and C
C if D and E
B if F and H

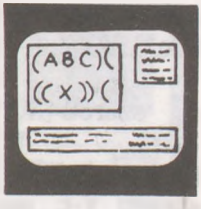
akkor a rendszer átszerkesztése bonyolult feladat, különösen, ha több száz szabályról van szó.

Miután egy *előre láncoló* szakértői rendszer nagyon sok szabályt használ, egy szabályértelmező szükséges, hogy ezek helyes sorrendben hajtódjanak végre. A szabályértelmező az alábbi szempontok szerint működhet:

1. Nézd végig a szabályokat, és amelyiknek tejesül a feltétele, azt hajtsd végre.
2. Ha egy szabály végrehajtásával ismétlés keletkezne a kontextuslistán, azt ne vedd figyelembe.

Lásd még: Metaszabály, Tudásbázis

PROLOG



Programozási technikák

A Prolog a matematikai logikai elvein alapuló programnyelv (lásd *logikai programozás*).

A logikában a tényeket egyszerű, formális módon fogalmazzuk meg, és így képezhetik egy logikai gondolatmenet alapját. A logikus helyesen megfogalmazott tényekből logikai módszerekkel új tényekre tud következtetni.

Eleanor szereti a mogyoróvajas szendvicset
Ha Eleanor szeret valamit, azt megeszi

Eleanor megeszi a mogyoróvajas szendvicset

A Prologot logikusok fejlesztették ki az *automatikus következtetés* céljaira, a számítógép lényegében a logikus szerepét játssza, és megvizsgálja a programban felsorolt tényeket.

Az a fajta következtetés, amelyet különösen alkalmas volt automatizálásra, a Bob Kowalski logikus által 1972-ben megalkotott *SL-rezolúció* volt. Az SL-rezolúció a logika *predikátumkalkulus* szarv-klóz részhalmazából következtet új tényekre. Az SL-rezolúcióból azután Colmerauer fejlesztette ki a Prologot.

A Prolog klasszikus demonstrációja az emberek közti viszonylatokkal foglalkozik. Tekintsük az alábbi példát:

felettese(Ádám, Klára)
felettese(Richard, Eleanor)

ami azt jelenti, hogy

Ádám Klára felettese,
Richard Eleanor felettese.

Ezek ugyan konkrét ténykijelentések, de a Prolog általánosabb kijelentéseket is tud alkotni.

szereti (X,Y) :- felettese (X, Y)

Ez azt jelenti, hogy ha van egy X és egy Y nevű ember, akkor X szereti Y-t, amennyiben X Y felettese. Ha most feltesszük a Prolognak a kérdést:

? szereti (Ádám, Klára)

vagyis

szereti-e Ádám Klárát?

akkor a Prolog ki tudja következtetni a választ. Ha ugyanis Ádám felettese Klárának, akkor Ádámnak szeretnie is kell őt. Ezt persze más programozási nyelvek is meg tudják tenni, de távolról sem ilyen elegánsan.

Tehát miben különbözik a Prolog a hagyományos programnyelvektől? A hagyományos programozó algoritmusokban, azaz valamilyen feladat megoldására irányuló, lépésenként végrehajtandó utasításokban gondolkodik. A Prolog programozó arra koncentrál, hogy milyen formális relációkat lehet definiálni az adott problémában: (ki kinek a főnöke, ki kit szeret stb.) és milyen formális relációk jellemzik a keresett megoldást. Következésképpen a Prologban való programozás rendkívül módszeres gondolkodást és szinte szőrszálhasogató alaposságot igényel.

Egy program lényegét alkotó logikai relációk igen absztraktak és nehezen megragadhatók lehetnek. Emiatt a Prolog és a logikai programozás tankönyvei sokszor családfákkal és rokonsági relációkkal kapcsolatos kimerítő példákat sorolnak fel, ahelyett, hogy valódi programozási feladatokat mutatnának be. Ezek a relációk ugyanis ismerősebbek és könnyebben érthetők, mégis jól példázzák a valódi logikai programozást.

Jó példa egy igen nehéz, mégis kiterjedten alkalmazott Prolog-fogalomra az APPEND predikátum. Ez explicite és maradéktalanul definiálja, mi játszódik le két lista összekapcsolásakor. Ez az első ránézésre triviális feladat sokkal izgalmasabb, mint képzelnénk. Lásd *logikai programozás.*

Lásd még: Logikai programozás

PROTOKOLL

Általános
számítógépes
kifejezés

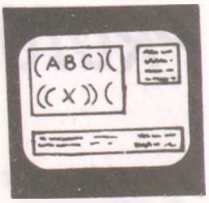
A robottechnikában (és még sok helyen) a protokoll valamilyen feladat leírása, hasonló egy recepthez. A részletesség foka azon múlik, hogy kinek készült: a robotnak nagyobb részletességre van szüksége, mint az embernek, akinek elég, ha ennyit mondunk: „vegyél egy fogót". A robotnak egy ilyen protokollt mozdulatról mozdulatra meg kell tanítani (ezt a fajta tanítást *végigvezetésnek* nevezik), vagy a szükséges lépéseket előre be kell programozni.

Hasonló értelemben szerepel a szó az adattovábbításban, ahol protokollnak neveznek egy eljárás- és egy adatformátumhalmazt, amelyek segítségével kapcsolatot lehet teremteni és adatokat továbbítani gépek között. Erre részletes protokollok széles skálája áll rendelkezésére.

A tudáselsajátítás kontextusában a protokoll írott feljegyzése mindannak, amit egy tevékenység során tulajdonképpen csinál az ember, többek között verbális jelzéssel arra is, hogy mire gondol a probléma megoldása közben. Még a „hmm" és „izé" is beletartozik, akárcsak a nyilvánvaló zsákutcák, mert ezek is lényegesek lehetnek a logikai folyamatban, kizárhatnak bizonyos választásokat, amelyek addig ígéretesnek tűnhettek. A második értelemben vett protokoll tehát sokkal szerteágazóbb, mint az első.

Lásd még: Tudáskigyűjtés

REKURZIÓ



Programozási technikák

Tegyük fel, hogy valakinek úgy magyaráznánk el, kik a rokonai, hogy azt mondanánk: „rokonaid a szüleid, testvéreid, illetve az ő rokonaik". Ez teljesen rendben van, de aki már eleve nem tudja, hogy mit jelent a „rokon", annak nem sokat mond. Ennek az az oka, hogy a magyarázatban az a fogalom is szerepel, amit éppen meg kívánunk magyarázni: a magyarázatban ördögi kör rejlik. Matematikai megfogalmazással: *rekurzív definíciót* adtunk.

A matematikában és a programozás bizonyos feladatainál a rekurzív definíció rendkívül hasznos lehet. A rekurzió klasszikus *logikai programozásbeli* demonstrációjában a számítógép egy családfán belüli kapcsolatokról gondolkodik. Kiderült, hogy az ilyen relációk kezelése számos párhuzamot mutat a valóságos logikai-programozási alkalmazásokkal, tehát nem a szerzők eredetiségének a hiánya az oka, hogy mindenütt ugyanez a fajta példa bukkan fel.

Az „elődje" relációt az alábbi módon definiálhatjuk:

1. X elődje Y-nak, ha
   X szülője Y-nak
2. X elődje Y-nak, ha
   X szülője Z-nek és
   Z elődje Y-nak.

Ha végigkövetjük ezt a definíciót, az elődeink között először a saját szüleinkre találunk. Azután a szüleink szülei, vagyis a nagyszüleink következnek, majd a dédszüleink, az ükszüleink, és így tovább.

A 2. rekurzív szabályban rejlik a trükk, hogy minden elődünk előfordul: a (fordított) családfát a gyökértől az ágak hegyéig végigkeressük, amerre a távolabbi elődök találhatók. Az 1. (nem

rekurzív) szabály állítja meg a folyamatot, amikor a legtávolabbi elődünkre bukkantunk.

A logikai programozásban a rekurzió igen gyakori alkalmazása az APPEND predikátum. Ez határozza meg, mi történik, ha két szimbólumlistát egymáshoz kapcsolunk. Ha az „egy macska" és a „mászott a matracra" szavakat akarjuk összekapcsolni, ezt úgy is megtehetjük, hogy az APPEND predikátum pontosan akkor legyen kielégítve, ha a két szólista helyes összekapcsolásával kijön a teljes mondat.

A Lisp *függvényszervezésű nyelv*, számokat, szavakat, neveket vagy más szimbólumokat függvények segítségével dolgoz fel. Gyakran megesik, hogy egy új függvény definíciójában magát a definiálni kívánt függvényt is felhasználjuk. Más szóval a függvények rekurzívak is lehetnek.

Az alábbi példa Hope függvényszervezésű nyelven írja le a „faktoriális" függvény rekurzív definícióját (a pozitív egész számok szorzatát 1-től az adott számig. Például 5 faktoriális = 1 × 2 × 3 × 4 × 5).

```
dec factorial: num → num;
   factorial (n)
          <= if n = 1
                then 1
                else factorial (n – 1) * n
```

Ha n = 5, akkor n faktoriális = 5-ször (n – 1) faktoriális (a program utolsó sorából); ennyi tehát világos. Ez a definíció azonban az (5 – 1) azaz 4 faktoriálisára támaszkodik. Tehát az egész cikluson még egyszer, még egyszer és még egyszer végig kell mennünk, sorban meghatározva 4, 3 és 2 faktoriálisát. A program nem rekurzív része „ha n = 1 akkor 1" leállítja az eljárást, mihelyt az 1-et elértük.

RISC

Hardver

RISC a *reduced instruction set computer* (csökkentett utasításkészletű komputer) rövidítése, egy új stratégia a számítógép jelentős felgyorsítására. A számítógép működési sebességének korlátait az szabja meg, hogy a processzorai milyen gyorsan hajtják végre az alapműveleteket (gépi kódot vagy gépi műveleteket). Az alaputasítások teljes gyűjteményét nevezik „utasításkészletnek". A magas szintű nyelveken, például Fortranban vagy Lispben írt programokat először le kell fordítani gépi utasításokra, hogy a gép le tudja futtatni őket.

A számítógép-tervezők eddig arra törekedtek, hogy bonyolult utasításokat tegyenek a gép utasításkészletébe, néha több alaputasítást sűrítve egyetlen utasításba, hogy a fordítóprogramok munkáját megkönnyítsék, és ezáltal gyorsítsák a gépet. Ezek a bonyolult utasítások azonban csökkentik az egész központi processzor hatékonyságát. Miután a legtöbb fordítóprogram csak egy töredékét használja a rendelkezésre álló gépi kódú utasításnak, a számítógép-tervezők egy csökkentett utasításkészletű számítógéppel álltak elő, amelynek sokkal kevesebb gépi kódú utasítása van, azok viszont gyorsabban működnek. A RISC gépek gyorsabban működnek a viszonylag egyszerű programokkal.

Nemsokára kijön a MISC, vagyis „minuscule instruction set computer" (minimális utasításkészletű komputer) egyetlen gépi kódú utasítással, és elképesztően gyors lesz, már csak arra van szükség, hogy valaki megmondja, mi legyen az az egyetlen utasítás.

ROBOTTECHNIKA

Hardver

A robottechnika robotok készítését és programozását jelenti, amelyek ma már bonyolult, rugalmasan használható, változatos alakú és méretű gépek lehetnek. A tervezés és elkészítés részleteivel itt nem foglalkozunk. A robotok programozása mindenesetre sokszor MI-technikákra támaszkodik, különösen a látórendszereknél (amelyek segítségével a robot meglátja, hol vannak a tárgyak) és a tervező rendszereknél (amelyek megmondják a robotnak, mikor és hogyan kell más helyre elmozdulnia).

SAKK

Játékok
és játék-
tartományok

A sakkozó automatáról szóló legkorábbi jelentések az 1800 körüli évekből származnak, amikor Kempelen Farkas bemutatta sakkasztal mellett ülő, török ruhába bújtatott automatáját. A török Európa-szerte számtalan játszmát nyert, és mindenkit lenyűgözött. Később azonban lelepleződött, hogy az automatában egy törpe rejtőzött.

A sakkozó programok kutatása már az 50-es években elkezdődött, amikor Lex Bernsteinnek sikerült egy 500 pontot elérő programot készítenie (az alábbi pontozási rendszer szerint):

Szabályosan játszó kezdő	500
Elég jó játékos	1200
Nemzetközileg rangsorolt játékos	2000
Nagymester	3000+

1967-ben Richard Greenblatt programja körülbelül 1100 pontos szintet ért el. 1972-ben David Slate és Larry Atkin az amerikai Northwestern Universityn készített Chess 3.6 nevű program 1400 pontot teljesített. A program egy későbbi változata, a Chess 4.0 egy gyorsabb számítógépen 2070 pontot ért el. Az 1982-es kompütersakk-világbajnok, a Belle pontszáma 2300 volt. A Deep Thought (Mély gondolat) nevű számítógépes rendszer (amelynek hardverét és szoftverét is speciálisan a sakkozás céljára készítették) 2550 pontos teljesítményt ért el. A Deep Thought a Carnegie Mellon Universityn készült, F. H. Hsu és Thomas Anantharaman munkája.

Hogyan működnek ezek a sakkozó programok? Hiszen a játék bármely állásában olyan elképzelhetetlenül nagy a lehetséges folytatások száma, hogy a számítógép aligha képes mindegyiket generálni, és megvizsgálni a lehetséges következményeket.

Igen fontos, hogy a sakkozó programok valamilyen módszerrel limitálják a számítógép által végrehajtandó keresés mennyiségét. A hihetetlenül nagy keresési térben a megfelelő lépés megtalálása olyan feladat, amelynek megoldása továbbra is az MI-kutatók előtt áll.

Az első meggondolandó szempont, hogy a játék fájában hány csúcsot kívánunk végigkeresni. Két dolgot lehet szabályozni: egy szint szélességét és a mélységet. Ha a program a teljes szélességben keres, akkor hatalmas, több százezer csúcsból álló keresési fákat kell végignéznie. Ez csak méregdrága, nagy számítógépeken lehet praktikus. A kisebb komputerek *legyezőszerűen* paraméterezik a szélességet és a mélységet, azaz szintenként eltérően korlátozzák a vizsgálandó fa szélességét.

Az alfa-béta nyírás egy másik olyan technika, amellyel korlátozni lehet a vizsgálandó csúcspontok számát. Ez lényegében arra utasítja a számítógépet, hogy a keresési fa egyes ágait „nyírja le", ha bizonyos számítások azt mutatják, hogy azok nem tartalmaznak egyetlen jó lépést sem.

A *higgadt keresés* csak azokat az állásokat vizsgálja, amelyeknél bizonyos abban, hogy a játék végső kimenetele szempontjából is jól becsülte meg az értékét. A beláthatatlan következményekkel járó lépéseket (amelyek hirtelen óriási előnyhöz juttathatják a komputert, de kedvezőtlen esetben visszafelé is elsülhetnek) nem is veszi figyelembe a keresésnél.

A második mérlegelendő dolog a keresésnél, hogy melyik az a legígéretesebb csúcs, amelynek a további elágazásait érdemes kifejteni. Ebben *heurisztikák*, vagyis olyan ökölszabályok segítenek, amelyek segítségével megbecsülhetjük, hogy egy lépés jó-e vagy sem. Például annak alapján lehet megítélni a kialakuló pozíciót, hogy (a komputer vagy az ellenfél szempontjából) lehet-e figurákat nyerni, nem kerülnek-e tisztek kötésbe vagy villába, lehet-e vezért, bástyát vagy futót játékba hozni és így tovább. A heurisztikákkal működő komputerek sok lépés kiiktatásával képesek csökkenteni a keresési teret.

A keresés korlátozásának egy másik technikája múltbeli játszmák ismeretére alapozódik. Egy sakkjátszma mondjuk száz lépésből áll. Viszonylag könnyű egy egész lépéssorozatot tárolni az adatbázisban: elég a figurák koordinátáját megadni. Lehetséges minden állást egy-egy számmal (úgynevezett *hash-kóddal*) ábrázolni egy táblázatban, ezekből azután kapunk egy indexet a játszmák pillanatfelvételeihez, és ennek segítségével a gép gyorsan végig tudja nézni, hogy az aktuális pozíció megegyezik-e valamelyik korábban ismert játszma egyik állásával.

Eszerint a számítógép képes felismerni bármely állást, amely az utolsó száz évben nagymesterek játszmáiban szerepelt. Azt is könnyen ki tudja deríteni, hogy egy adott nehéz állásban mit lépett valamelyik nagymester. Ha a nagymester mást lépett, ez támpont arra, hogy a felmerülő lépést kizárja.

1974 óta a számítógépes sakk fejlődése legalább felerészben a puszta hardverteljesítménynek köszönhető. Manapság a számítógépek gyorsabb és mélyebb keresésre képesek, mint korábban bármikor. A Deep Thought sakkozó rendszerben a keresés minden feladatára külön specializált chipet fejlesztettek ki. Hét VLSI (nagyon nagy mértékben integrált) áramkör kezeli a táblázatokat, amelyek a sakkjáték alapjellegzetességeit tárolják. Például a gyalogok elhelyezkedését a táblán, illetve ezek jelentőségét. Noha a Deep Thought a taktikai előrelátás mélységét és pontosságát illetően messze felülmúlja a nagymestereket, az ember magasabb rendű hosszú távú stratégiai elképzelése révén mégis felül tud kerekedni.

Lásd még: Keresés, Alfa-béta nyírás, Minimax technika

SEJTAUTOMATA

Elmélet/
Filozófia

Ez olyan automata, amely memória és processzorelemekként egyaránt sejtek egy tömbjét használja. A legismertebb John Conway híres Életjátéka, amelyet egy végtelen négyzetrácsos táblán kell játszani. Minden négyzet, vagyis sejt vagy üres, vagy egy elemet tartalmaz. Minden menetben (generációban) a játékos három egyszerű szabály alapján kiürít vagy betölt sejteket. Amikor ugyanezt a komputer játssza, másodpercenként több generáció sebességgel, lenyűgöző képek alakulnak ki, és ennek nem kis része van a sejtautomata népszerűségében.

Van olyan sejtautomata, amely teljes számítási képességgel rendelkezik, így a számítógép-tudomány elméleti problémáinak elemzésére is alkalmas. A neuronhálózatokhoz is közel állnak, hiszen nagyszámú egyszerű elemből épülnek fel, amelyek kölcsönösen meghatározzák egymás működését, és így a Neumann-féle, illetve a közönséges számítógépek elméleti modelljeként is szolgálnak.

Lásd még: Automata

SOROS MŰKÖDÉS

Általános számítógépes kifejezés

A soros vagy szekvenciális számítógép egyszerre egy utasítást hajt végre. A legtöbb Neumann-féle, azaz közönséges számítógép soros működésű, köztük a mini- és mikrogépek többsége is. A soros működés alternatívája a *párhuzamos*, illetve az *osztott feldolgozás*.

Lásd még: Párhuzamos feldolgozás, Neuronhálózatok

SZAKÉRTŐI RENDSZER SHELL

Szakértői rendszerek

A szakértői rendszer shell szakértői rendszer készítésére szolgáló alkalmazáscsomag. Lehetséges ugyan a „semmiből" is szakértői rendszert írni valamilyen programnyelven, erre azonban nincs feltétlenül szükség. A szakértői rendszer shell kész eszközöket ad egyszerű szakértői rendszerek készítéséhez a megfelelő tartományokban.

Mit nyújt egy szakértői rendszer shell? Először és elsősorban eszközt ad a programozónak a tudás reprezentációjára. Erre többféle technika létezik, a legismertebb a *produkciós szabály*, részben némiképp átalakított formában, ugyanis ez a „legtermészetesebb" a laikusnak, mert a szabályokat a természetes (emberi) nyelvhez közel álló formában írják le.

Például:

IF: rendelés nagy tétel
and vásárló magas számlát fizet
and vásárló nem Jones és tsa.

THEN: engedmény 10%

Persze több produkciós szabályból szakértői rendszert készíteni egész más feladat, mint hogy egymás után leírjuk őket. Mondjuk 50 szabály megfelelő együttműködését elérni meglehetősen trükkös feladat. Jól meg kell értenünk, hogy a rendszer hogyan fogja használni ezeket a szabályokat, hogy következtetéseket tudjon levonni a segítségükkel, és pontosan tudnunk kell, milyen fajta szaktudást akarunk rögzíteni benne (lásd *tudástervezés*).

Nem a szabályok jelentik az egyetlen módszert a tudás reprezentációjára. A fejlettebb szakértői rendszer shellek (vagy „esz-

közkészletek") a programozónak további eszközöket biztosítanak (lásd *Objektumorientált programozás, Keret, Forgatókönyv*).

Miután megírtuk egy szakértői rendszer szabályait, egy programmal át kell alakítanunk a számítógéppel futtatható formába. Például a szabályokat először Prologba kell átírnunk, majd a compilerrel végrehajtható kódra lefordítanunk. Eszerint a szakértői rendszer shell egy *compilert* vagy *interpretert* is tartalmaz, attól függően, melyik felel meg jobban a célnak.

A rendszerben a következtetőgép az a rész, amely a szabályok alapján a tényleges következtetéseket levonja. Nélküle a szakértői rendszer nem tudná intelligens módon felhasználni mindazt, amit tud. Tehát a szakértői rendszer shellel együtt egy következtetőgépet is vásárolunk. További részleteket lásd a *Következtetőgép* címszónál.

A legtöbb esetben a következtetőgép valamilyen módon számot ad arról, hogy egy következtetéshez milyen szabályokat használt fel, így a szakértői rendszer viselkedését elemezni is lehet. Ezt a gondolatot részletezi a *magyarázó rendszer* címszó.

Lásd még: Magyarázó rendszer, Szakértői rendszerek

SZAKÉRTŐI RENDSZEREK

Szakértői rendszerek

A szakértői rendszerek a mesterségesintelligencia-kutatásnak a piacon leginkább megjelenő megnyilvánulásai. Ezek a programok valamilyen terület „szaktudását" sűrítik magukba, és lehetővé teszik, hogy a felhasználó ezzel kapcsolatban kérdéseket tegyen fel. Ez a tudás persze mindig egy nagyon szűk ismeretterületről, vagy ahogy mondani szokták, *tartományból* származik.

Az egyik legelső tartomány, amellyel kereskedelmi forgalomba hozott szakértői rendszer foglalkozott, nagyszámítógépes rendszerek konfigurációnak rendelése volt. Egy nagy konfiguráció olyan sok összetevőből áll, hogy a megfelelő elemek megrendelése rendkívül bonyolult feladattá vált. A Digital számítógépes cég a kereskedők támogatására, kifejlesztette az XCON elnevezésű szakértői rendszert, hogy a megfelelő vevőnek a megfelelő konfigurációt tudják eladni. Ez a rendszer R1 néven is ismertté vált. Ez volt a maga nemében az első szakértői rendszer, és 1980 óta sikeresen segít VAX-konfigurációrendelések összeállításában.

A szakértői rendszerek történetében újabb mérföldkövet jelentett az 1970-es években a Stanford Universityn Edward Shortcliff által kifejlesztett MYCIN, amely a vér bakteriális fertőzéseit diagnosztizálta oly módon, hogy a baktériumok bizonyos jellegzetességeit megtárgyalta a felhasználóval. A MYCIN, más hírneves szakértői rendszerekhez hasonlóan (lásd A. függelék) az IF ... THEN alakú úgynevezett *produkciós szabályok* formalizmusán alapul:

```
IF (alak=kerek) and (szín=zöld) THEN név := ALMA
```

Sok szakértői rendszer ma is ilyen szabályokon alapul, de ma már számtalan másfajta tudásreprezentációt is alkalmaznak (lásd *Objektumorientált programozás*, *Keret*, *Forgatókönyv*).

Egy szakértő tudásából számítógépes rendszert készíteni távolról sem könnyű feladat. Ne feledjük, hogy egy komoly szakértői rendszer elkészítése emberek többéves munkáját igényelheti. És nem a kódolás (magának a programnak a megírása) tart ilyen sokáig, hanem annak a módnak a megtalálása, hogy a szakértő tudását a számítógépbe tudjuk „sűríteni", és hozzáférhetővé tenni azt a felhasználó számára. Ezeket a feladatokat magyarázzuk el a *Tudástervezés* és *Tudás-kigyűjtés* címszavak alatt.

A 70-es években már számos szakértői rendszer képes volt a bizonytalan információk kezelésére. Ennek szokásos technikája, hogy a bizonyosság fokát mesterségesen skaláris értékkel jellemzik: „Esni fog, 0,8 bizonyossággal", vagy „0,6 a valószínűsége, hogy ez az információ helyes." Ahol a rendszernek egyszerre sok bizonytalan kijelentést kellett magába olvasztania, és belőlük új információt generálnia, bonyolult numerikus módszereket, „kalkulusokat" alkalmazott. Néhány ismertebb példa:

1. Bayes tétele, a PROSPECTOR szakértői rendszer alkalmazta;
2. a Dempsey–Shafer elmélet;
3. az evidens érvelés;
4. megerősítési elmélet, a MYCIN szakértői rendszer bizonyossági tényezőit ezzel számították ki. Egy adott hipotézis bizonyossági tényezője két mérték különbségeként adódott, ahol az első a hipotézis igazságára, a második a hipotézis cáfolatára vonatkozott.

Ezekkel a numerikus módszerekkel azonban sok baj van. Nagyjából azt lehet mondani, hogy ezek a kalkulációk *ad hoc* módszerek: egy-egy tartományban gyakran nem ismeretes az előforduló események valószínűségi eloszlása. Ezenkívül igen nehéz figyelembe venni, hogy egy esemény előfordulása miként befolyásolja a másikat. Azt sem lehet tudni, hogy bizonyos tények összefüggenek-e és hogyan.

Talán nem kevésbé lényeges az a tény, hogy mind a szakértők, mind a felhasználók számára szokatlan, hogy állításaik igazságfokát ilyen numerikus precizitással adják meg. Egy szakértő nehezen szánja rá magát, hogy az ismereteihez valószínűségi értéket rendeljen. Ezek a kalkulációk tehát erősen korlátozott lehetőséget adnak az emberi következtetések feldolgozására.

A 80-as években kezdték a bizonytalanságot úgy kezelni, hogy a tartományról további ismereteket szereztek be, hogy még több szabállyal jellemezhessék a szakismereteket. Teljesen eltérő megközelítés a *neuronhálózatok* potenciális alkalmazása, ahol

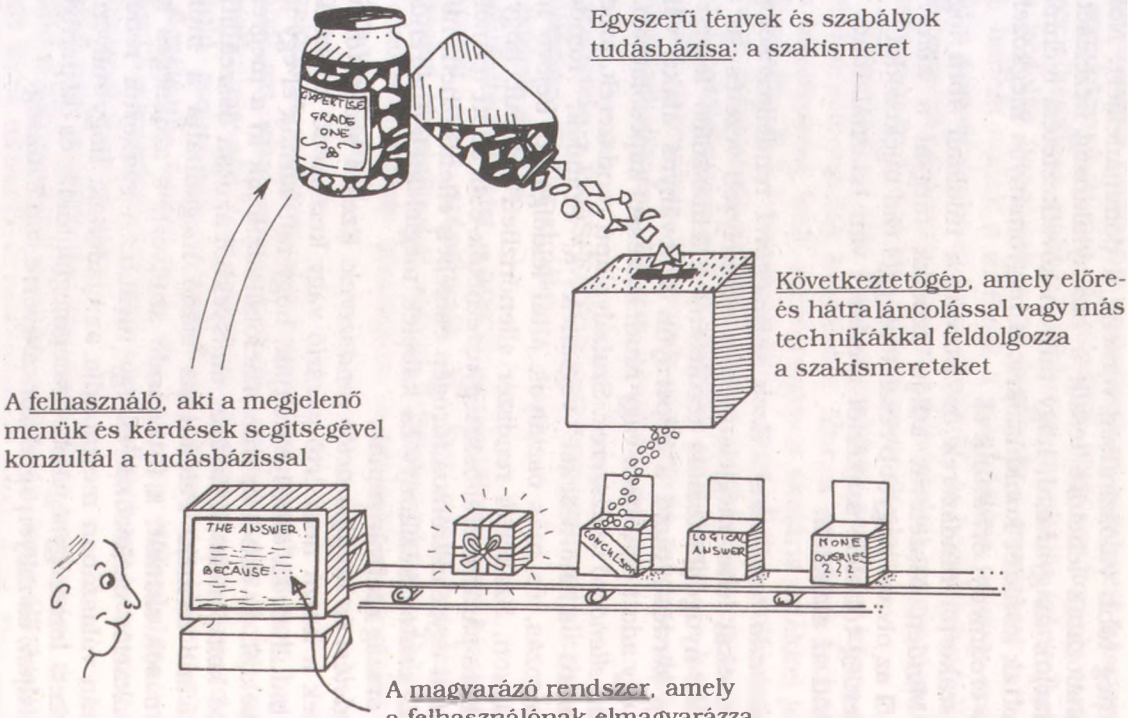
egyidejűleg több valószínűség verseng a dominanciáért. Noha a neuronhálózatok lehetővé teszik a bizonytalanság kezelését, ha meg akarjuk magyarázni, hogy milyen következtetési technikákkal jutottak konkrét konklúziókra, a hagyományos megközelítéssel jobb eredményt érhetünk el.

A szakértői rendszerek összetevőit a túloldali ábra foglalja össze. Minden összetevőt külön szócikk tárgyal a könyvben, amelyből az olvasó nagyobb részletességgel tud tájékozódni.

Jelenleg többféle szakértői rendszer van használatban, többek között az alábbiak:

1. *Tanácsadó rendszerek.* Ezek segítségével rendelkezéseket és eljárásokat lehet megjeleníteni a személyzeti vezetés, az adózás és nyugdíjbiztosítás területén. A felhasználó feltesz néhány kérdést, majd a képernyőn szabványos alakú választ kap egy adott eljárással vagy rendelkezéssel kapcsolatban.
2. *Irodai ellenőrző rendszerek.* Szabályalapú rendszerek, amelyek egyszerű ellenőrzéseket végeznek a Számlák, Rendelésfeldolgozás, és más osztályok által feldolgozott bejövő információkon. Szakértői rendszer ellenőrizheti például, hogy egy rendelés paraméterei összeegyeztethetők-e az adott termékkel. Például egy elektromos fúrógép esetében ellenőrizheti, hogy a hozzá csatolt kézikönyv és kábelek megfelelnek-e a rendeltetési ország szabványainak.
3. *Rendelési és konfigurációs rendszerek.* Ezek a szakértői rendszerek a hozzá nem értő vásárló vagy kereskedő információiból indulnak ki, amellyel leírják, hogy mit várnak el egy adott terméktől, és ebből az információból alakítják ki a megrendelendő komponensek listáját, amelyekből azután összeállítható a kívánt termék. Például az eladó megadhatja a munkaállomások számát, a futtatandó szoftvert, a szükséges kommunikációs eszközöket és így tovább. A szakértői rendszer azután válaszában megmondja az eladónak, hogy milyen kábeleket, lemezegységeket, szalagmeghajtókat és képernyőket rendeljen, és milyen rendszerszoftverre van szükség.

A szakértői rendszer összetevői



Egyszerű tények és szabályok tudásbázisa: a szakismeret

Következtetőgép, amely előre- és hátralláncolással vagy más technikákkal feldolgozza a szakismereteket

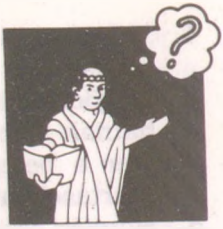
A felhasználó, aki a megjelenő menük és kérdések segítségével konzultál a tudásbázissal

A magyarázó rendszer, amely a felhasználónak elmagyarázza, a következtetőgép hogyan jutott egy konkrét következtetésre

4. *Valós idejű felügyelő rendszerek.* A valós idejű felügyelő rendszerek több változót figyelnek folyamatosan, és tőlük függően állítanak bizonyos paramétereket. Példaként szolgálhatnak az orvosi felügyelő rendszerek, ahol a szakértői rendszer a beteg fiziológiai folyamatait felügyeli, illetve az ipari gyártásellenőrző rendszerek.
5. *Harctéri rendszerek.* Ezek a rendszerek nagy mennyiségű alacsony szintű adatot fogadnak (radaron vagy rádión keresztül, például repülőgépek típusáról és helyzetéről) amiből értelmes modellt állítanak fel a valóságos helyzetről. A beérkező adattömeget ezáltal a taktikai döntéseket hozó emberek számára jobban érthető formába önti. Az ilyen rendszerek azonban nem tévedhetetlenek: előfordulhat, hogy rosszul interpretálják az adatokat.

Lásd még: Bizonyosság, Elasztikus logika, Előre láncolás, Hátra láncolás, Következtetőgép, Magyarázó rendszer, Szakértői rendszer shell

SZÁMÍTÓGÉPPEL SEGÍTETT OKTATÁS



Elmélet/
Filozófia

A számítógépeket háromféle módon használják az oktatásban. Vannak kifejezetten az oktatás (általában a programozás oktatása) céljaira kifejlesztett nyelvek és számítógépes környezetek: Seymore Papert Logója az egyik klasszikus példa. Vannak azután olyan programok, amelyekkel valami megtanulandó, ám az osztályterembe be nem hozható anyagot lehet szimulálni: példa lehet az űrhajózás és a genetika. Ezeket a programokat szimulációra szánták, az oktatásban való felhasználásuk csupán másodlagos. Az MI csak a harmadik kategóriában játszik lényeges szerepet: az intelligens komputerrel segített oktatásban (ICAI), ahol az „intelligencia" arra szolgál, hogy a számítógép kideritse, a tanuló miben szorul gyakorlásra, és ennek megfelelően alakítsa a tanulás további menetét.

Az első kísérletek, amelyeket inkább számítógéppel segített tanulásként (CAL) lehet leírni, keretorientáltak voltak. Mintapéldákat állítottak fel, és minden lehetséges választ előre elképzeltek. Például a következő kérdésre: „2 + 3 = ?", a lehetséges válaszokat: „< 4", „5", „6", „ >6". Ebben esetben előre programozott intelligenciáról beszélhetünk.

A későbbi programok MI-technikák segítségével próbálták kideríteni, hogy a tanuló miért adott hibás választ. Ehhez szükség volt mind a megtanulandó ismereteknek, mind a tanuló tudásának belső ábrázolására. Az utóbbiba a tanuló hiedelmei, illetve ismereteinek bizonyossági fokai is beletartoztak. Mindkét tudástípus ábrázolására különféle technikákat alkalmaztak. A 70-es évek végén a program vezérlésére bevették a tanítói módszerek szabályait is (szemben a tárgyra vonatkozó szabályokkal), ezáltal pontosabban sikerült modellezni a tanár viselkedését. Ez a terület azonban még most is kezdetleges stádiumban van.

Az ICAI-programoknak foglalkozniuk kell a diákok tanulási problémáival is, akárcsak az általuk beszélt (természetes) nyelvvel és a sok területen megfigyelhető informális érveléssel. Az ember sokszor még egészen formális tárgyakról, például a matematikáról is informális nyelven beszél. Tehát minden tanító programnak képesnek kell lennie arra (amire az EXCHECK képes a matematikai logikában), hogy összeegyeztesse a tudás formális belső ábrázolását a pusztán emberi megértésre szolgáló kevésbé formálisakkal. Az ICAI-programok saját maguk állítják elő a tanulónak feladandó problémákat (és nem egy előre elkészített készletből veszik elő). És nem pusztán azért elemzik a válaszokat, hogy a helyességüket ellenőrizzék, hanem hogy a további problémákat, a tanuló gyenge pontjainak tekintetbe vételével, módosíthassák. Alakfelismerési és induktív tanulási technikákat alkalmaznak, hogy megismerjék a tanuló tudását (vagy hiedelmeit) a tárgyban. A program azt feltételezi, hogy a tanuló rendelkezik az ő ismereteinek egy részével vagy az egésznek egy hibás változatával, (vagy mind a kettővel). Az utóbbi egyébként az automatikus programozás hibajavításához is kapcsolódik, hiszen a hibás tudással rendelkező tanuló ahhoz hasonlítható, mint amikor egy program az adott terület információját dolgozza fel, ám hiba van benne.

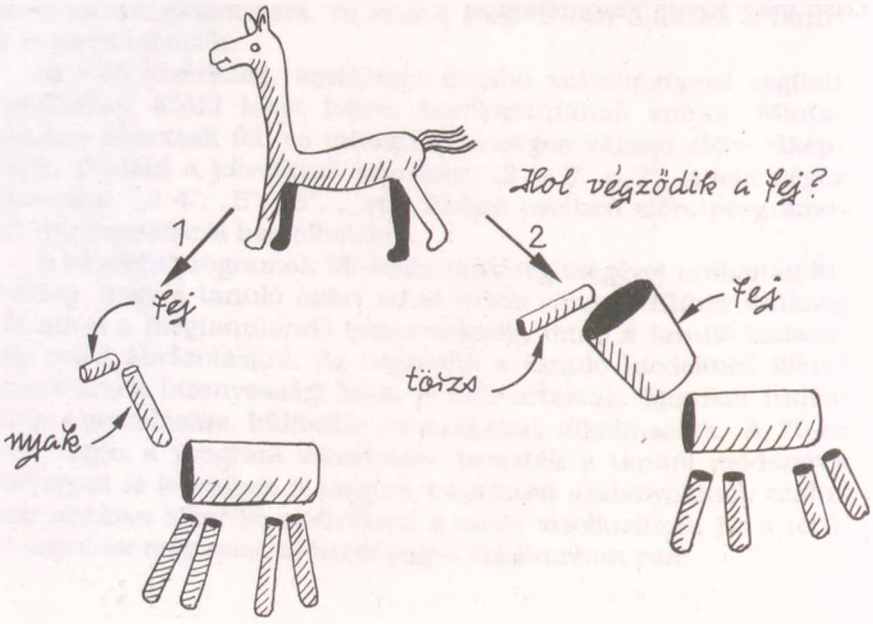
Lásd még: Keret, Forgatókönyv

SZEGMENTÁLÁS

Látás

A szegmentálás kifejezés, ahogy a látás kutatásában használják, arra vonatkozik, hogy a számítógép hogyan bontja fel a tárgyakat értelmes egységekre. Egy képen két szinten is végrehajtható a szegmentálás.

Alacsonyabb szinten a kép különböző árnyalatú vagy textúrájú *régiókra* bontható. Ez a folyamat az *elemi vázlat* készítésével párhuzamosan játszódhat le, amely egyszerű információt szolgáltat a képen látható síkok számáról, esetleg azok irányáról is. Az így definiált régiók és az élek adatainak elemzéséből kiderül, hogy azok valódi síklapoknak, vagy árnyékoknak felelnek-e meg.



Magasabb szinten a szegmentáció egy tárgy részletei közötti *határok* megtalálásával és így végső soron a tárgy azonosításával foglalkozik. Az elefántot például ormányos állatként definiálhatjuk, de ha a számítógép nem képes helyesen orra-fejre-nyakra-törzsre tagolni az állatot, esetleg összetéveszti egy zsiráffal. Tehát a szegmentálásnak ez a (sokkal nehezebb) változata azzal foglalkozik, hogy foltok bonyolult együtteseit hogyan lehet azonosítható részekre tagolni.

A tárgyakat rendszerint *általánosított hengerekre* szokták tagolni. A henger az a test, amelyet egy körvonal pontjai súrolnak, ha a síkjára merőlegesen egyenes vonalban elmozdítjuk. Az általánosított hengernél kör helyett bármilyen zárt görbét vehetünk. Az elefánt formája például közelíthető egy „törzsként" jelölt nagy hengerrel, illetve ebből kinyúló kisebb, fejként, lábként stb. címkézett hengerekkel is. Ezt az egyszerűsített leírást mint grafikont vagy listát azután elrakhatjuk valamilyen adatbázisban. A részletgazdagságot további kisebb hengerek hozzávételével növelhetjük. A probléma ott lép fel, amikor azt akarjuk eldönteni, hogy ezek a hengerek hogyan csatlakoznak egymáshoz. Hibás döntés esetén zsiráfot lőhetünk az agyaráért. A képek szegmentálására vannak heurisztikus szabályok, de nincs általános módszer: hogy mi magunk hogyan csináljuk, az valószínűleg a tapasztalaton, illetve az agyunk idegsejtjei közötti kapcsolatokon múlik.

Lásd még: Textúra

SZEMANTIKA

Természetes nyelv feldolgozása

A szemantika a nyelvi jelentéssel foglalkozik, ellentétben a szintaxissal, amely a mondat szavainak a grammatika szabályainak megfelelő elrendezését írja le. Például az alábbi mondat:

Az almának kék íze volt,

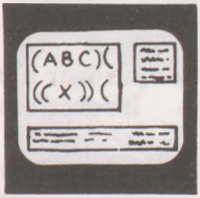
szintaktikailag helyes, szemantikailag azonban hibás: megfelel a nyelvtan szabályainak, mégis értelmetlen.

Az MI-ben a jelentés problémája komoly akadályt gördít a nyelvmegértésre vállalkozó rendszerek sikeressége elé. Mi ugyan bármiről könnyen meg tudjuk mondani, hogy informálisan mit jelent, mögöttünk azonban hatalmas köznapi tudás és tapasztalat áll. Ha egy teljes értelmező szótárt táplálunk is a számítógépbe, amely minden szó jelentését megmagyarázza, a mögöttük húzódó gondolatok tökéletesen rejtve maradnak. A bekezdés jelentése több, mint az alkotóelemek puszta összege.

Vannak olyan MI-kutatók, akik optimisták abban a tekintetben, hogy a párhuzamos feldolgozás, és a neuronhálózatok technikája meghozza a megoldást. Talán az ilyen architektúrájú számítógépek képesek párhuzamosan több ezer jelentést megvizsgálni, hogy megtalálják, melyik értelmes az adott kontextusban.

Lásd még: Szintaxis, Pragmatika

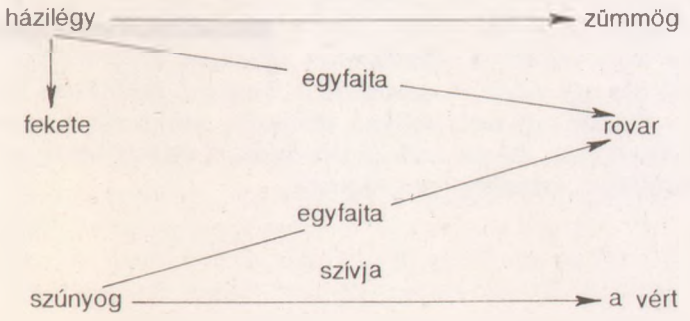
SZEMANTIKAI HÁLÓ



Programozási technikák

A szemantikai háló fogalmak és a közöttük levő kapcsolatok formális ábrázolására szolgál. A szemantikai hálót Quillian vezette be 1969-ben az adatbázis alternatív struktúrájaként, és ezzel új korszakot nyitott a természetes nyelvi jelentés reprezentációjában.

A szemantikai háló olyan gráf, amelynek csúcsai egyszerű fogalmaknak felelnek meg, élei pedig a fogalmak közötti kapcsolatokat jelölik. Például rovarok jellegzetességeit jeleníti meg az alábbi szemantikai háló:



Hogyan festhet mindez számítógépes programként? A szemantikai háló csupán az információt írja le, nem a programot. Ahhoz, hogy egy szemantikai hálóból működőképes programot készíthessünk, további szabálykészlet szükséges, hogy a számítógép fel tudja használni az ismert tényeket. Ha például hozzátesszük,

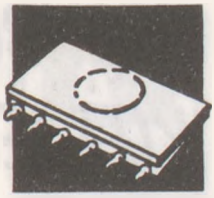
hogy

a rovar házilégy, ha zümmög és fekete;
a rovar szúnyog, ha vért szív;

akkor már szemantikai hálónk nem egyszerűen a rovarokat és a jellegzetességeiket leíró adatgyűjtemény: lehetővé teszi, hogy az adatbázisban rákérdezzünk, hogy egy beadott kifejezésről tartalmaz-e azt alátámasztó adatokat. Más szóval a szemantikai háló lehetővé teszi, hogy ellenőrizzük egy adott tény „igazságát".

Lásd még: Tudásbázis, Szemantika

SZILÍCIUM-RETINA



Hardver

Ez olyan hardver, amely a kép elemzésével alakfelismerésre képes. A retina idegsejtrétege a szemgolyó falán érzékeli a fényt. Alapvető feldolgozási funkciókat is ellát, így például éleket keres. Carver Mead készített egy ehhez hasonló félvezetőt, amelyet szilíciumretinának nevezett el, és amely annak több funkcióját ellátja, számos vonatkozásban lemásolva felépítési és működési elveit. Ez a neuronhálózat chipképek előfeldolgozására alkalmazható, és legfőbb gyakorlati példa a neuronhálózatok alkalmazására az alakfelismerésben, amely feladatban – a nagyfokú párhuzamosság miatt – a hagyományos számítógépek hatékonysága viszonylag alacsony.

Lásd még: Neuronhálózatok

SZIMBOLIKUS KIFEJEZÉS

Általános számítógépes kifejezés

Szimbolikus kifejezés az, amelyben konkrét számok, betűk, szavak helyett szimbólumok állnak. Például a

2 * 3

egy aritmetikai kifejezés, ami az aritmetika nyelvén fejez ki valamit. Ha ennél általánosabban akarjuk kifejezni magunkat,

x * y-t

írunk, ahol az x és y szimbólum számokat helyettesít, ám ők maguk nem számok. (Természetesen a „2" meg a „3" is szimbólumok, amelyek a „két dolog" illetve "három dolog" helyett állnak.) Ide kapcsolódik a *szimbolikus számítás* kifejezés is. A fenti első kifejezés egy konkrét számítás: a 2-szer 3 értékének kiszámítása. A második egy szimbolikus kifejezés, szimbolikus számítást jelöl. Az iskolában ezt nevezik *algebrának*, a szimbolikus számítás tehát valójában számítógépes algebra.

SZIMBOLIKUS PROGRAM-NYELV

Általános
MI-kifejezés

A Fortran, Basic és hasonló programnyelvek különösen alkalmasak aritmetikai számításokra. Ugyanezek a nyelvek jók egy adott szó vagy kifejezés gyors megkeresésére egy szövegben vagy listában. Jól használhatók tehát könyvelő és táblázatkezelő programok írására.

Vannak olyan számítógép-nyelvek, például a Lisp vagy a Prolog, amelyek egy másik kategóriába esnek. Ezek kevesebb lehetőséget adnak matematikai, könyvelő, tudományos alkalmazások készítésére, viszont sokkal alkalmasabbak a gondolatok, nevek, szavak szimbólumokként való kezelésére. Ezért ezeket szimbolikus programnyelveknek szokás nevezni.

SZIMULÁCIÓ VAGY EMULÁCIÓ

Elmélet/
Filozófia

Az egyik legsúlyosabb kritika az MI-vel szemben, hogy az általa elért eredmények valójában nem „intelligensek", csak annak látszanak. Más szóval az MI-programok nem emulálják (közelítik meg), csak szimulálják (utánozzák) az emberi tevékenységeket. A klasszikus szimuláló program az ELIZA, amely jellegzetes motívumokat (szavakat) keresett az input szövegben, és attól függően adott ki valamely másik szöveget. Az eredmény nagyon emlékeztetett a beszélgetésre, de persze semmit nem jelentett a komputernek: szó sem volt a programban a kérdés megértéséről, csak a fő motívumok megismétléséről.

Azt a tézist, hogy minden szabályokra alapozott rendszer legfeljebb arra képes, hogy szimulálja az emberi tevékenységet, John Searle adta elő a legnagyobb ékesszólással az MI-ről alkotott *Kínai szoba* analógiájával.Ez a kritika valószínűleg akkor igazán találó, amikor a hagyományos számítógépes látórendszerekre vonatkoztatjuk, amelyeknek csak kevés kapcsolatuk volt a mi látórendszerünkkel (és egyébként is gyengén működtek); kevésbé érvényes a szakértői rendszerekre. Az utóbbiak olyan szabályokat használnak, amelyek maguknak a terület szakértőinek a definíciója szerint megegyeznek azzal, ahogy *ők* valójában gondolkodnak. Természetesen ez nem szükségképpen ábrázolja jól azt, ahogyan *valójában* gondolkodnak: inkább azoknak a tudatos folyamatoknak a leírása, amelyeket meg tudnak fogalmazni. Kimarad belőle az implicit tudás nagy része, amit a szakértői rendszerek csak szimulálni képesek. A kritika valószínűleg legkevésbé a neuronhálózatokra érvényes.

Lásd még: Kínai szoba

SZINTAXIS

Természetes
nyelv
feldolgozása

A szintaxis a mondatban levő szavak formális elrendezésével foglalkozik, tekintet nélkül azok jelentésére. A mondatban a szavak csak bizonyos kombinációkban fordulhatnak elő. Például

A disznó átrepült a kerítés fölött

mondat szintaktikailag helyes, de

A disznó kerítés átrepült fölött

már ellentmond a grammatika szabályainak, tehát szintaktikailag hibás. A *szemantika* a szintaxissal ellentétben a jelentéssel foglalkozik. Még egy szintaktikai szempontból helyes mondat is lehet teljesen értelmetlen.

Az almának rendkívül sárga íze volt

mondat szemantikája legalábbis kétséges, jóllehet szintaktikai szempontból kifogástalan.

Lásd még: Grammatika, Elemző

SZINTBEN INDULÓ KERESÉS

Keresés

A számítógépes megoldáshoz számos problémát keresési fával lehet ábrázolni. A keresési fa egy probléma megoldási fázisait „csúcsokkal" ábrázolja. A csúcsokat összekötő élek „operátorokat" jelképeznek, vagyis eljárásokat és tevékenységeket, amelyek alkalmazásával a megoldás egyik fázisából átjutunk a másikba.

A szintben, illetve mélységben induló keresés két olyan technika, amelyek segítségével módszeresen végigjárhatjuk egy keresési fa csúcsait. A keresés illusztrálására képzeljük el, hogy a komputer repülőgépen San Franciscóból Nizzába kíván eljutni.

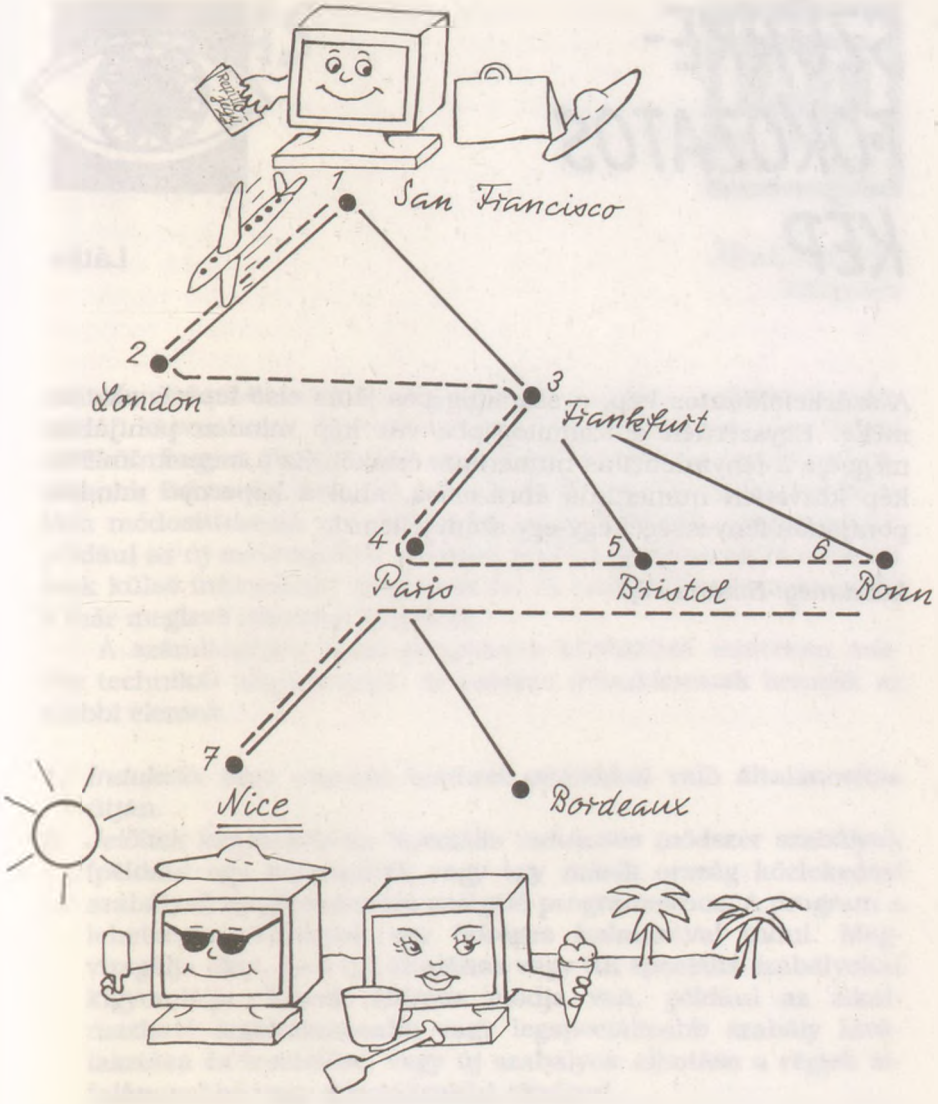
A számítógép információját a városok közötti útvonalakról, egy keresési fában ábrázolhatjuk. A keresés San Franciscóval indul. Első lépésként a gép generálja az első csúcs (a „gyökér") következő csúcsait. Ezek az 1. szintű csúcsok. Köztük van London és Frankfurt, egy lépésre a gyökértől. Az 1. szintű városok között Nizza nem szerepel, tehát a számítógép generálja a 2. szintű csúcsokat, amelyek két repülőútnyira esnek San Franciscótól.

Ez az eljárás, amellyel szintről szintre haladva generáljuk az új csúcsokat, addig folytatódik, amíg meg nem találjuk Nizzát. Ezután a számítógép kijelölheti a legrövidebb, legkevesebb átszállást igénylő útvonalat a mediterrán célállomásig.

Sok ágat tartalmazó keresési fánál a szintben induló keresés nagyon hosszú időbe telhet. Ami még fontosabb, túl sok memóriát igényel, mivel az eljárás során az egy szinthez tartozó összes csúcsot egyszerre kell tárolni. Példánkban minden 3. szintű csúcsot el kell menteni, miközben a 4. szintű csúcsokat generáljuk.

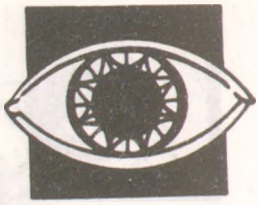
Jelentős memóriaigénye miatt a gyakorlatban előfordulhat, hogy a szintben induló keresés másodpercek alatt a teljes memóriát lefoglalja.

Lásd még: Keresés, Mélységben induló keresés


1
San Francisco
2
London
3
Frankfurt
4
Paris
5
Bristol
6
Bonn
7
Nice
Bordeaux

Keresési út

SZÜRKE-FOKOZATOS KÉP



Látás

A szürkefokozatos kép, a számítógépes látás első lépésének terméke. Egyszerűen a számítógépbe vitt kép minden pontjához megadja a fényintenzitás numerikus értékét. Ez a monokróm TV-kép közvetlen numerikus ábrázolása, ahol a képernyő minden pontjának fényességét egy-egy szám jellemzi.

Lásd még: Bináris kép

TANULÁS

Általános MI-kifejezés

Számos olyan technika létezik, amelynek révén a számítógép „tanulni" képes, azaz új információkat és eljárásokat elsajátítani anélkül, hogy a programozó új programot írna. A tanulóprogramoknak információhoz kell jutniuk, hogy azután ennek megfelelően módosíthassák a viselkedésüket. A felfedező programoktól, például az új matematikai tételeket felállító programoktól eltérően ezek külső információt dolgoznak fel és asszimilálnak, nem pedig a már meglevő adatokat tárják fel.

A számítógépes tanulóprogramok különböző szinteken sokféle technikát alkalmaznak, és sokszor felbukkannak bennük az alábbi elemek.

1. *Indukció*, azaz tanulás konkrét példákból való általánosítás útján.
2. *Jelöltek kigyomlálása.* Speciális indukciós módszer szabályok (például egy kártyajáték vagy egy másik ország közlekedési szabályai) megtanulására szolgáló programokhoz. A program a lehetséges szabályok egy bőséges halmazával indul. Megvizsgálja őket, és a túl általános vagy túl speciális szabályokat kigyomlálja. Ennek számos módja van, például az alkalmazható legáltalánosabb vagy legspeciálisabb szabály kiválasztása és tesztelése, vagy új szabályok alkotása a régiek általánosabbá vagy speciálisabbá tételével.
3. *Genetikus algoritmusok.* Random additív keresési technikával, ismételt „mutációkkal" és kiválasztással keresnek egyre jobb szabályváltozatokat (vagy programokat, vagy tetszőleges stringeket).
4. *Neuronhálózatos tanulás.* A neuronhálózatok nem hagyományos algoritmusokkal működnek, így az algoritmusok optimalizálására szolgáló módszerek sem alkalmazhatók. Tanuló al-

goritmusokra azonban mégis szükségük van, vagyis előírásokra, hogyan módosítsák a neuronok közötti kapcsolatokat aszerint, hogy az output mennyire felel meg az elvártnak. A *visszatáplálás* többek között ilyen módszer.

A számítógépes tanulás, amely során a számítógép a saját programját módosítja, az automatikus programozás egy válfaja. Ebben azonban egyelőre kevés előrehaladás történt. A számítógép képessége, hogy tanuljon a tapasztalatból, a legszűkebb formális kontextusoktól (például kártyajátékok elsajátításától) eltekintve alig meggyőző, még a legegyszerűbb állatokkal, mondjuk legyekkel vagy csigákkal összehasonlítva is. Ami azt illeti, éppen az ilyen primitív állatok tanulmányozásából indult ki a neuronhálózatokkal megvalósítható tanulás ötlete, azzal a reménnyel, hogy ez lényegesen eredményesebb lesz a hagyományos, algoritmikus megközelítésnél.

Lásd még: Automatikus programozás

TÁRGYI TUDÁS

Általános
számítógépes
kifejezés

A tárgyi tudás a szimbólumok, gondolatok és tárgyak legális (lehetséges) viselkedési módjaival kapcsolatos ismeret. Ebben szemben áll a *metatudással*, amely a bennünk meglevő tárgyi tudás felhasználására vonatkozó stratégiákból áll. A tárgyi tudás az, amit tudunk, a metatudás pedig arról szóló tudás, amit tudunk.

Vegyük a következő problémát:

Hogyan állítjuk elő a 8 számot a 12-ből 2 kivonásával és hozzáadásával?

A tárgyi tudás a megengedett aritmetikai lépéseket tartalmazza:

Hogyan kell 2-t hozzáadni
Hogyan kell 2-t levonni.

A metatudás azt mondja meg, ezt a tudást hogyan lehet felhasználni, hogy megkapjuk a választ:

Ha 2-t adunk egy pozitív számhoz, akkor megnöveljük
Ha 2-t levonunk egy pozitív számból, akkor csökkentjük
A 8 pozitív szám
12 nagyobb, mint 8
Ha egy számból nagyobb számot akarunk előállítani, akkor azt növelni kell
És így tovább ...

TARTOMÁNY

Általános
számítógépes
kifejezés

1. Tartománynak nevezünk a tudás vagy szakismeret egy-egy olyan területét, amelyben Mi-technikákat lehet alkalmazni. Például a szakértői rendszerek technológiája alkalmazható az olajfúrás tartományára vagy a keresési technikák a sakk tartományára.
2. Egy függvény értelmezési tartománya azoknak az értékeknek a halmaza, amelyekre alkalmazni lehet. Például

   '3 páratlan' igaz;
   '6 páratlan' hamis;
   '(„hello") páratlan' sajtóhiba.

A 'páratlan' függvény értelmezési tartománya az egész számok halmaza, értékkészlete pedig az {igaz, hamis} halmaz.

TECHNIKAI MUNKA-ÁLLOMÁS

Általános számítógépes kifejezés

A technikai munkaállomás olyan számítógép, amelyet főként kutatásokra és szoftverfejlesztésre használnak.

Az 1970-es évek végén és 80-as évek elején volt egy tendencia, hogy nagy teljesítményű számítógépeket használjanak egy-egy konkrét alkalmazás kifejlesztésére. Például a Symbolics cég felvásárolta a Symbolics 3600 nevű, kizárólag Symbolics Lispben írt programok fejlesztésére specializált Lisp-számítógépet. Ezek a specializált gépek igen drágák voltak: így azután olcsóbb alternatívaként divatba jöttek az általánosabb technikai munkaállomások.

A tipikus technikai munkaállomás UNIX operációs rendszer alatt dolgozik. Nagyméretű, magas felbontású képernyővel és több megabyte-os RAM-mal rendelkezik. Sok MI-alkalmazás futtatásához nagy memóriára van szükség: a szimbolikus számítások ugyanis a hagyományos programoknál jóval memóriaigényesebbek.

Lásd még: Fejlesztői eszköz

TERMÉSZETES NYELV FELDOLGOZÁSA

Természetes nyelv feldolgozása

A természetesnyelv-feldolgozás a mesterségesintelligencia-kutatás egyik kutatási területe, olyan eszközök kifejlesztésére irányul, amelyek segítségével emberi (tehát nem számítógépes) nyelven kommunikálhatunk a számítógéppel. Ez a kommunikáció történhet írásban és szóban is.

Miután nyelvünkben az intelligenciánk is kifejeződik, aligha meglepő, hogy a számítógéppel csak úgy értethetjük meg a mondanivalónkat, ha teljesen intelligenssé tesszük. Az MI kutatóinak, miközben a természetesnyelv-feldolgozás kérdéseivel bajlódnak, szembe kell nézniük ezzel a ténnyel, és abba is bele kell törődniük, hogy az ő életükben valószínűleg nem jön létre az igazi természetes nyelvű ember–gép-párbeszéd.

Eközben a természetesnyelv-feldolgozás továbbra is izgalmas kutatási terület marad a nyelvészet, filozófia, pszichológia és számítógép-tudomány találkozási pontján.

Ahhoz, hogy a számítógép értelmezni tudja az emberi nyelvet, a következő típusú tudásra van szüksége:

1. a mondatban szereplő szavak elfogadható sorrendjének ismerete (a *grammatika* ismerete);
2. annak ismerete, hogy az emberek hogyan használják a szavakat gondolataik és véleményeik kifejezésére;
3. a beszélgetés törvényeinek ismerete, ezen belül egy modell a személyiségről, attitűdjeiről, a beszélgetőpartner megértésének szintjéről;
4. a világ általános (hétköznapi) ismerete.

A grammatika problémájával már elég alaposan foglalkoztak. Számos grammatikai modell programját készítették el, amelyek megadják, hogyan fűzhetők össze mondatokká a szavak. Egy „szin-

taktikai" *elemző* segítségével a számítógép a mondatot mondatrészekre (igékre, főnevekre, melléknevekre) tudja bontani és megkonstruálja a nyelvtani szerkezetét. Ha már a mondatot sikerült részekre bontani, viszonylag könnyű feladat a mondatrészek összeszerkesztésével grammatikailag helyes válaszokat előállítani. Noha abban szinte mindenki egyetértene, hogy a mondat szintaxisa is fontos, a legtöbben úgy gondolják, hogy ezt az információt a szemantikai tudással is ötvözni kell. Más szóval a mondat „felszíni" sorrendjén túl valahogy a jelentését is ábrázolni kellene.

Azt a tudást, hogy az emberek hogyan alkalmazzák a szavakat gondolataik és véleményeik kifejezésére, sokkal nehezebb ábrázolni a számítógépben, mint egy mondat szavainak típusát és sorrendjét. Általában bizonytalanok vagyunk afelől, hogy mit is akarunk mondani, és ritkán állítunk valamit feketén-fehéren. Pszichológusok feltevése szerint amikor egy kimondott mondatot hallunk, valószínűleg megpróbálkozunk párhuzamosan többféle értelmezéssel. Például tekintetbe vesszük bizonyos szavak divatos vagy kulturált használatát. Még a hétköznapi beszélgetésekben is mondunk olyasmit, aminek a megértése az emberek számára könnyű, de a legfejlettebb komputer se lenne rá képes.

Tegyük fel, hogy egy dolgozót írunk le a számítógépnek. A leírásban az alábbi is szerepel:

Ő az a fajta ember, akit az édesanyád kedvelne: nagyon jól elüldögél egy kényelmes fotelban, és bámulja a tévét.

A gép két pozitív tulajdonságot jegyezne meg ebből:

1. hogy az édesanyád kedvelné őt;
2. hogy egy bizonyos tevékenységet nagyon jól tud.

Ez pontosan az ellenkezője annak, amire gondoltunk. Más mondatok egyszerűen csak többértelműek:

Láttuk a Wembley stadiont útban a londoni repülőtér felé.

Csak a világról szóló általános tudásunk alapján tudhatjuk, hogy nem a Wembley stadion volt útban a repülőtér felé.

Sok módszert fejlesztettek ki, amely korlátozott módon képes a mondatok jelentésével foglalkozni. Például az esetgrammatikák keretjellegű reprezentációt adnak a mondatoknak, a különböző szócsoportok funkcióját feltüntetve (ki, mikor, mit csinált).

Egy igazi természetes beszélgetésnél arról a személyről is tudnunk kell valamit, akivel szemben állunk. Bizonyos rendszerek már most képesek figyelembe venni, hogy kezdővel vagy szakemberrel állnak szemben, amikor valamit kezdenek elmagyarázni, de még nagyon durván működnek (lásd *Magyarázó rendszerek* és *Kooperatív rendszerek*). Lehet, hogy a jövő számítógépes rendszerei végtelen türelemmel, udvarias és segítőkész modorban fognak tárgyalni a felhasználóval. Ilyen jellegű működő rendszerektől persze még igen messze vagyunk.

A világról szóló hétköznapi tudással gyakorlatilag minden ember rendelkezik:

Minden leesik, ha valami nem tartja.
Ha el akarod érni a vonatot, ki kell menned az állomásra.

A számítógépnek (hogy meg tudja tanulni), pontosan meg kell mondani az ilyesmit. A hétköznapi tudás problémáját kikerülendő a legtöbb jövőbeli rendszer csak nagyon kicsi, pontosan definiált tartományon fog működni.

TERMÉSZETES NYELVI FELÜLET

Természetes nyelv feldolgozása

A természetes nyelvi felület lehetővé teszi, hogy emberi nyelven kommunikáljunk a számítógéppel. Természetes nyelvi homlokzatnak is nevezik. Ma még csak korlátozott nyelvi inputot fogadnak el. A természetes nyelvi felületek elsődleges feladata adatok visszakeresése nagy adatbázisokból. Az adatbázis jól körülhatárolt tartomány, és bár több ezer tényt tartalmazhat, nem sok kérdést szoktak róluk feltenni. Persze az ember, még a specialista is, különbözőképpen fogalmazhatja meg a kérdéseit. Hogyan képes a természetes nyelvi felület kezelni ezt a változatosságot?

Erre többfajta technikát használnak. Talán a legkönnyebben érthető a Gary Hendrix (Stanford Research Institute) által tervezett LIFER. A LIFER grammatikája nem igéket, mellékneveket és más „beszédrészeket" keres (lásd *Grammatika*), hanem jelentés egységek alapján próbálja megérteni a mondatot.

Képzeljünk el egy felületet egy nagy raktár festékkészlet-adatait tartalmazó adatbázishoz. Az alapvető jelentésegységek „szín", „festéktípus", „fedés" és „ár" lehetnek. A LIFER-alapú felület ehhez az adatbázishoz három programozási trükköt használhat:

1. sémák;
2. parafrázis;
3. ellipszis.

A *séma* olyan szabványos mondatforma, amelyre más nem szabványos mondatformákat lehet leképezni. Például az alábbi két mondat:

Mi az ára a lila emulziós festéknek?
Milyen a fedési tulajdonsága a vörös enyves festéknek?

egyaránt a következő sémára képezhető le:

(MI A) TULAJDONSÁGA (AZ X) FESTÉKNEK?

A *parafrázis* egy adott mondat átfogalmazása változatlan jelentéssel. Például:

Hány doboz sárga festékünk van?
HELYETT: Mennyi a raktárkészlet sárga festékből?

Az *ellipszis* nem teljes mondatokban megfogalmazott kérdésekre vonatkozik. A kérdések jelentése az előző kérdés alapján deríthető ki.

Milyen a fedési tulajdonsága a sárga enyves festéknek?
És az ára?
És a rendelkezésre álló mennyisége?

A LIFER a sajtóhibákkal is megbirkózik, képes a névmások és szinonimák értelmezésére is.

A LIFER viszonylag egyszerű felület. Ezért a következő hiányosságokkal rendelkezik:

1. A párbeszéd szigorúan csak egy szűk tartományra vonatkozik;
2. A rendszer nem tudja kezelni a diszjunkciót (A vagy B), a mennyiségi meghatározásokat, a következtetést és az okságot.
3. Szigorú kötöttségek érvényesek az adattípusokra és a lehetséges kapcsolatokra közöttük.

A LIFER több mint tíz éve készült, azóta nagyon sokat fejlődtek a természetes nyelvi felületek, bár az említett ötletek közül sok még ma is használatos. A természetes nyelvi felületekre egy egészen különböző megközelítést tesz lehetővé az ATN, a kiterjesztett átmenetháló.

Még ha ismernénk is a mondatok grammatikai szerkezetét, egészen más feladat ebből kivonni a jelentést. Noha vannak olyan felületek, amelyek a grammatikai elemzési technikákat valamilyen, a mondat jelentését reprezentáló rendszerrel egészítik ki, ezek sokat hibáznak, és csak egy szűken definiált tartományban tudnak dolgozni.

Lásd még: Homlokzat, Természetes nyelv feldolgozása

TERMÉSZETES OSZTÁLY

Elmélet/
Filozófia

Ezt a kifejezést a tudással és tudásalapú rendszerekkel kapcsolatban használjuk. Természetes osztálynak nevezzük a dolgok egy csoportját, ha „természetesen" egy kategóriába sorolhatók. Így például a „kutya" egy „természetes" csoport: mindenki tudja, hogy mi tartozik bele és mi nem. Más kategóriáknak viszont, mint például az „intelligencia" vagy a „mikrokomputer", nincsenek univerzálisan elfogadott megfelelőik a valós világban.

Ideális esetben egy szakértői rendszer tudásbázisának természetes osztályokba kellene tagozódnia, tehát a „Cerberus" alatt tárolt adatok elérésével automatikusan fel kell idéződjön a kutyaság minden attribútuma, a hegyes fogak, szőrösség, az ugatás, de irreleváns információ, például hogy ugyanazzal a betűvel kezdődik, mint a „káposzta", már nem. Az *objektumorientált programozási* technikák pontosan erre valók. Az egyetlen probléma, hogy nem világos, vajon a valóságban léteznek-e természetes osztályok. Miért van az, hogy egy farkas nem számít kutyának, de a „Buksi" nevű puha rózsaszín szőrös játékállatka igen? Hogy lehet az ENIAC számítógép, ha egy modern zsebkalkulátor, amely több memóriával és számítási teljesítménnyel rendelkezik, nem az?

A filozófus Wittgensteinnek sok mondanivalója volt az ehhez hasonló ellentmondásos problémákról, a modern filozófusok és szociológusok is lelkesen foglalkoznak ezzel a területtel. Bár a számítógéptudósokkal izgalmas beszélgetéseket lehet folytatni róla, legjobb, ha egyébként távol tartja tőle magát az ember.

Lásd még: Explicit és implicit tudás

TERVEZÉS

Keresés

Az MI sokszor foglalkozik olyan tervezési problémákkal, ahol a feladatvégzésnek sokféle módja van, mint például dobozok tárolása egy gyárban. A tervezőprogramok részletes sémákat adnak arra, hogy valamilyen jól leírt végállapotot hogyan kell elérni. Nem triviális dolog úgy megtervezni a feladatokat, hogy mindegyik a kellő időben és helyen kerüljön sorra, és egyik se zavarja a másikat.

A tervezőrendszereknek képesnek kell lenniük következtetéseket levonni tárgyuk korlátozó feltételeiből, oksági viszonyairól és a stratégiákról. Az outputban a rendszer megmondja a felhasználónak, hogy egy adott tevékenység mennyire illeszkedik egy meglevő tervhez, vagy milyen választásai vannak egy feladat végrehajtásában.

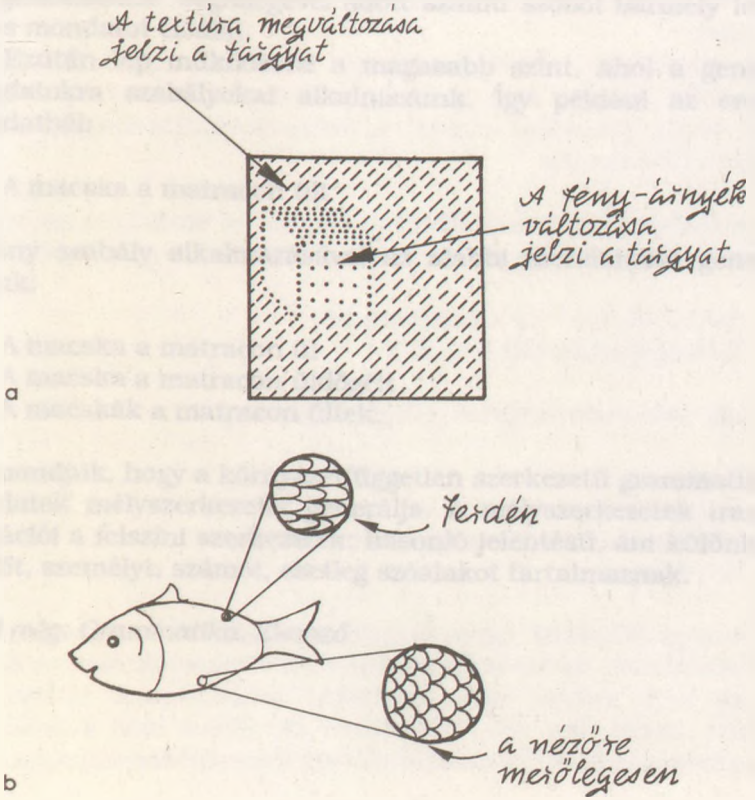
A tervezőprogramokat *építőkocka-világokban* tesztelték, ahol a célállapotot egyértelműen le lehetett írni, és a program teljes tudással bírt a „világról". Az SHRDLU természetes nyelvi programban volt egy tervező, amely alapján a számítógép ki tudta számítani, hogy a pillanatnyi helyzetből hogyan lehet eljutni a felhasználó által megkívánt állapotba. Vannak erre sokban emlékeztető gyakorlati alkalmazások is, például dobozok elhelyezése automatizált raktári rendszerekben.

A tervezést ebben az értelemben elsősorban a robotvezérlésben alkalmazzák, de használják ezenkívül ipari projektek, utazások, munkakiosztás, műholdas missziók és tudományos kísérletek megtervezésére is, váltakozó sikerrel.

TEXTÚRA

Látás

Egy tárgy felületi textúrája a számítógépes látórendszernek értékes támpontot ad a tárgy mibenlétére és térbeli elhelyezkedésére vonatkozóan. Ha a szín, vagy fényesség nem elég árulkodó, a textúra alapján a legtöbb tárgy elkülöníthető a háttértől, amelyen



nyugszik. A textúrák megkülönböztetése azonban nehéz feladat. Textúrája csak egy területnek lehet. Tehát ha egy adott képponthoz valamilyen textúrát akarunk rendelni, valamilyen tulajdonság alapján definiálni kell tudnunk a területe, ahova esik.

Ha már sikerült azonosítani, a terület textúrája a tárgy térbeli elhelyezkedéséről is elárul valamit, ha összevetjük a velünk pontosan szembenéző felület textúrájával. (Ez persze feltételezi, hogy tudjuk, annak milyen a textúrája. Feltételezhetjük, hogy a legtöbb felület izotróp (a mi látórendszerünk is él ezzel a feltételezéssel): vagyis minden részterületük egyforma. Ha tehát valamilyen tulajdonság láthatóan megváltozik, annak a térbeli elfordulás lehet az oka. Másfelől feltételezhetjük, hogy minden ellipszis, amit látunk, valamilyen szögben elforduló kör, minden téglalap megdöntött négyzet és így tovább. Végül, a látórendszerben lehet egy adatbázis a „lapos" felületek textúráiról.

Lásd még: Mélység

TRANSZ-FORMÁCIÓS GRAMMATIKA

Természetes nyelv feldolgozása

A transzformációs grammatika fogalmát Chomsky, a nyelvész alkotta meg 1957-ben az MIT-n, azóta is folyamatos tanulmányozás és fejlesztés tárgya.

A transzformációs grammatika két fokozatban működik. Az elsőben egy nagyon egyszerű elméleti eszköz, a „kontextus-független grammatika" segítségével adott számú szóból bármely lehetséges mondatot előállít.

Ezután lép működésbe a magasabb szint, ahol a generált mondatokra szabályokat alkalmazunk. Így például az eredeti mondatból:

A macska a matracon ült

néhány szabály alkalmazásával az alábbi mondatokat generálhatjuk:

A macska a matracon ül
A macska a matracon üldögélt
A macskák a matracon ültek.

Azt mondjuk, hogy a környezetfüggetlen szerkezetű grammatika a mondatok mélyszerkezetét generálja. A mélyszerkezetek transzformációi a felszíni szerkezetek: hasonló jelentésű, ám különböző igeidőt, személyt, számot, esetleg szóalakot tartalmaznak.

Lásd még: Grammatika, Elemző

TUDÁSBÁZIS

Szakértői rendszerek

A tudásbázis az MI-rendszereknek az a része, amely egy adott szakterületen a tényeket, szabályokat és fogalmakat tárolja. Például egy növényi megbetegedéseket diagnosztizáló rendszer tudásbázisa a növényi megbetegedések tüneteiről szóló információt tartalmazná. A konzultáció során a következtetőgép azután az ebben a tudásbázisban tárolt információ alapján vonna le következtetéseket.

A tudásbázis, mint gyanítható, jól strukturáltan tárolja az ismereteket, hogy a következtetőgép könnyen hozzá tudjon férni, és fel tudja használni azokat. A tudás ábrázolásának szokásos módjai a következők:

1. Az ismereteket bemutathatjuk tények és szabályok sorozataként, amelyek vagy *procedurálisak* (eljárásjellegűek), például:

   tedd X-et, ha Y és Z bekövetkezik
   X elvégzéséhez előbb tedd Y-t

   vagy *deklaratívak*, (leíró jellegűek) például:

   X igaz
   X igaz, ha Y és Z igaz
   Minden Y Z tulajdonsággal rendelkezik
   X Z-típusú

2. Összegyűjthetünk egyes tárgyakra és eszmékre vonatkozó információkat, és *keret* formájában ábrázolhatjuk őket. A keretnek több nyílása van, amelyeket attribútumok értékei (pl. szín, méret, kor stb.) tölthetnek ki, illetve más keretekhez, szabályokhoz és utasításokhoz való kapcsolódás információi.

3. A tudásbázist összefüggő fogalmak szemantikai hálójára is alapozhatjuk.

Ha a tudást szabályokkal és nem „hagyományos" programozási technikákkal reprezentáljuk, ennek előnye, hogy a tudásbázis könnyen karbantartható. Egy nagyméretű, mondjuk Basicben vagy Cobolban írt rendszert nagyon nehéz követni. A program akár több száz oldalon keresztül tarthat, és egy adott ponton nagyon nehéz megmondani, hogy mi is történik. Hasonlóképpen nehéz megválaszolni olyan jellegű kérdéseket, hogy a program hogyan viselkedik egy-egy konkrét esetben („Hogyan kezeli a program a nagybani szállítóknak feladott megrendeléseket?" vagy „Mit tud a program a pöttyös pizsamákra vonatkozó árengedményekről?), különösen, ha nincs tisztességesen karbantartva.

Ezzel szemben egy szabálysorozat alakjában megírt program még a nem programozó számára is érthető lehet.

TUDÁS-KIGYŰJTÉS

Szakértői rendszerek

A tudáskigyűjtés az a folyamat, amelynek során emberi szakismereteket gyűjtünk, össze, hogy felhasználásukkal szakértői rendszert készithessünk. Része a tudástervezés folyamatának, amit speciális számítógépes elemzők, a *tudástervezők* végeznek.

A sikeres tudás-kigyűjtéshez egyrészt szükséges megtudni, hogy az elképzelt szakértői rendszerhez milyen szakismeretekre van szükség, másrészt meg kell beszélni a klienssel, hogy az ő szervezetének kontextusában hogyan lehet azokat felhasználni. Egyik feladat sem könnyű: egy komoly rendszernél hónapokba telhet, amíg a tudástervezők és a kliens számára is világos, hogy a rendszer mire lesz és mire nem lesz képes. Ezalatt a tudáskigyűjtőnek, akinek talán fogalma sincs arról a területről, amellyel dolgoznia kell, a szakismeretek egész útvesztőjében kell kiigazodnia, és azokat olyan fajta módon reprezentálnia, ami azután a későbbi program alapjául szolgálhat.

A szakemberrel való beszélgetés során a tudástervező diplomata és detektív egy személyben. Az emberi szaktudást ugyanis soha nem olyan könnyű kigyűjteni.

Először is a szakértő által mondottak teljes értelmetlenségével kell megbirkóznia. Minden szakértő azt gondolja, hogy egyszerűen és világosan beszél a tárgyáról. Mindent megtesz, együtt érez a számítógépessel, és ezért a (legalábbis a számára) lehető legegyszerűbb kifejezéseket és fogalmakat használja. Egyszerűen hihetetlennek tűnik, hogy valaki mindezek után sem érti, amit mond. Például minden könyvelő meg van lepve, hogy az egyszerű halandó a legegyszerűbb könyvelési fogalmakat sem képes felfogni. Hiszen még egy számítógépesnek is kell legyen valami fogalma a „naplófőkönyvről", „napi bizonylatról" és „részletes revízióról". Hát nem feltétlenül. A tudástervezőnek tehát el kell játszania a diák szerepét, aki felteszi a kezét, és kérdez.

És nem csupán a terminológiából származik zavar: a dolgok összefüggéseiből is. Kemény feladat rájönni, mi az igazán fontos, és mi a jelentéktelen részlet. Egy teljesen érdektelennek tűnő témában egyetlen ártatlan kérdés egy egész lelkes kiselőadást indíthat el. Máskor, ha olyasvalamire kérdezünk rá, ami addig kulcskérdésnek tűnt, egyszavas választ kapunk.

Az igazán nehéz helyzetekben persze különböző formális tudáskigyűjtő technikákra hagyatkozhatunk. Például használhatunk kártyákat, amelyekre felírunk egy-egy alapfogalmat vagy kifejezést. Összekeverjük őket, majd megkérjük a szakértőt, hogy ossza két csoportra. Aztán megpróbáljuk kideríteni, mire alapozta a csoportosítást. Ezáltal világosabb képet kaphatunk arról, hogy a terület különböző fogalmai hogyan függenek össze.

Egy másik módszer az, hogy egy tudástervező-teamet felkérünk, játssza el az elképzelt szakértői rendszer szerepét. A „szakértői rendszer" papíron kommunikál a felhasználóval, ahogy végleges formájában a képernyőn tenné. Ez a gyakorlat jó elképzelést ad a tervezett rendszer pontosságáról, ahogy a kérdéseket kezeli, és hogy megfelelő módon reprezentált-e a tartomány.

A nap végeztével szeretnénk valamelyes áttekintést nyerni a tárgyról, és jól megérteni azokat a fogalmakat, amelyekkel az elkészült rendszer dolgozni fog. Számos módja van annak, ahogy a gondolatainkat megvitatásra előterjeszthetjük: elem-relációs diagramok, összetevő-diagramok, szemantikus háló stb. Ezek a féligmeddig formális módszerek alkalmasak a gondolatok kritikus megvitatására: kijelölik a megbeszélés fő szempontjait. Később azután a tartomány még formálisabb reprezentációjává alakíthatók.

Lásd még: Tudástervezés

TUDÁS-TERVEZÉS

Szakértői rendszerek

A tudástervezés az emberi szakértelem kigyűjtésének és szakértői rendszerré alakításának a folyamata. Tehát ide tartozik a tudáskigyűjtés is, amelynek során a számítógépes elemző szakértőket hallgat meg, és kutatja a vizsgált tartományt.

Egy nagy szakértői rendszeren a tudástervező-team akár két évig vagy még tovább is dolgozhat. Egy ilyen teamben különféle szakemberek lehetnek a felhasználói felületért felelős pszichológustól (lásd *emberi tényezők kutatása*) a programkódot író programozóig. Mellettük dolgoznak a csoportban a kliens igényeinek felmérésében gyakorlott elemzők és azok, akik a kikérdezésben, a lelkes bemutatásban és az üzletpolitikában gyakorlottak.

Hogy némi képet adjunk a tudástervezésről, leegyszerűsítve felvázolunk egy képzeletbeli projektet egy szakértői rendszer megtervezésére. A szóban forgó rendszer feladata, hogy eladóknak segítsen egy nagy vállalat számára konfigurálni egy bonyolult számítógépes rendszert. Az eladó konzultál a vevővel, majd a szakértői rendszer diagramot készít a vásárló igényeinek megfelelő számítógéprendszerről, és segít megrendelni az alkotórészeket.

1. hónap A tudástervezők elmagyarázzák, hogy egy szakértői rendszer hogyan segítené az eladógárdát számítógéprendszerek konfigurálásában. A kereskedelmi igazgató csak homályosan érti a dolgot, de úgy ítéli meg, hogy érdemes tovább foglalkozni vele.

2. hónap Az elemzés kezdeti stádiuma. A tudástervezők számos látogatást tesznek, kereskedőkkel és vezetőikkel beszélgetnek a projektről. Az alábbi kérdésekre keresnek választ:

1. Mi a feladata a szakértői rendszernek? Hol és hogyan szerezhetjük be a szükséges ismereteket a tartományról?

2. Milyen hardveren futtatná a kliens a szakértői rendszert?
3. Milyen legyen a felhasználói felület? Legyen menüvezérelt, vagy elég, ha a rendszer igen-nem válaszokat ad? Van-e szüksége a rendszernek grafikára?
4. Kell-e közös felület táblázatkezelőkhöz, adatbázishoz vagy más szoftverhez?
5. Milyen jellegű legyen a rendszer outputja?
6. Ki lesz a rendszer felelőse telepítés után? Ki végzi majd a karbantartást? A szabályokat szakember vagy „laikus" fogja felfrissíteni?
7. Valószínű-e a rendszer jövőbeli kiterjesztése?

3. hónap A tudástervezők modellt készítenek a rendelésfeldolgozás és eladás jelenlegi folyamatáról a számítógépes cégnél. A modell még az egyes vevőkről, nagykereskedelmi árengedményekről, illetve a valutaárfolyamokról is tartalmaz adatokat.

A csoport a számítógép-alkotórészek listáját is felállítja, feltüntetve, hogy a cég által forgalmazott több száz féle elem hogyan kapcsolódik egymáshoz. Ebben az ábrázolásban világosan megjelenik, hogy melyik lemezegységhez milyen kábelek tartoznak, mi a termékszáma a spanyol billentyűzetnek stb: lényeges ismeretek, ha számítógép-rendszert akarunk szállítani egy nagy cégnek.

6. hónap Elkészül a rendszer prototípusa, meggyőzve a klienst, hogy jó úton haladnak a dolgok. Alábbhagy a gyártók félelme, hogy a projekt finanszírozása bármikor leállhat, mindenki boldog.

9. hónap Elkészül a szakértői rendszer részletesebb specifikációja. Ebből azután egy még jobb prototípus készül. Ezt a kliens is megbírálja, változtatásokat javasol a felhasználói felületen, és rámutat a rendszer ismereteinek hiányosságaira. A tervezés és bírálat fázisa így váltja egymást, amíg végül ...

20. hónap A kliens meg van elégedve a prototípussal. A specifikáció megfelel az igényeinek.

Most következik a tervezői fázis. Ennek során a felhasználói felület, a karbantartási eszközök, a hatékonyság, és a futtató környezet részleteire koncentrálnak. A termék végső átadása előtt természetesen még egy tesztelési fázis is következik.

Lásd még: Tudáskigyűjtés, Szakértői rendszerek

TURING-GÉP

Elmélet/
Filozófia

A Turing-gép egyfajta automata, egyszerű elméleti számítógép, amellyel tesztelhetjük, mit lehet és mit nem lehet számítógéppel elvégezni. A Turing-gépnek nagyon kis memóriája van, (ezt nevezzük a gép „belső állapotának"), és egy kis utasításkészlete. Inputja és outputja egyaránt egy író-olvasó „fejen" keresztül továbbítódik egy karaktereket tartalmazó papírszalagra. A fej karaktereket tud olvasni a papírszalagról, illetve új karaktereket írni rá. A gép arra is képes, hogy a szalagon jobbra vagy balra lépjen, ha a programja és a „memóriája" erre utasítja.

Alan Turing, aki a gépet a 30-as években feltalálta, kimutatta, hogy elég hosszú szalaggal a Turing-gép bármit ki tud számolni, amit „determinisztikus módszerekkel" (ma inkább diszkrét elektronikának hívjuk, és ide tartozik a digitális számítógép is) el tudunk végezni. Tehát teljes számítási képességgel rendelkezik, és így a digitális számítógép modelljeként szolgál.

Lásd még: Automata

TURING-TESZT

Elmélet/
Filozófia

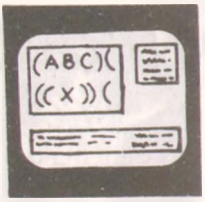
Ezt a tesztet vagy „gondolatkísérletet" Alan Turing találta ki annak eldöntésére, hogy lehet-e értelmesen „intelligens számítógépről" beszélni. Elképzelt két külön szobát, mindkettőben egy-egy távvezérléssel összekötött írógéppel. A tesztelő személy az egyik szobában ül, és kérdéseket ír le az írógépével, amelyekre válaszokat is kap. Ha egy előre megállapodott idő után nem tudja eldönteni, hogy a kapott válaszok embertől vagy a géptől származnak-e, akkor a válaszolót joggal tekinthetjük intelligensnek.

Turing úgy gondolta, hogy ez a teszt az intelligencia meglétére koncentrál, és ezzel segített meghatározni, mit kell tudnia egy intelligens számítógépnek (vagy, ahogy ma mondanánk, intelligens programnak). Ami azt illeti, ez csak formalizált változata annak, ami a bíróságok előtt vagy munkaerő-felvételeken történik, ahol az egyik ember kérdések segítségével megpróbálja kitalálni, mi zajlik le a másik fejében.

Ezt a tesztet is becsaphatják olyan programok, amelyek inkább szimulálják, mintsem emulálják (megközelítik) az emberi gondolkodást. Alapfeltevése, hogy a *Homo sapiens* az intelligencia ideális mércéje, amit talán még Turing is (akit embertársai öngyilkosságba kergettek) megvitatásra érdemesnek tartana.

Lásd még: Szimuláció vagy emuláció, Kínai szoba

ÚJRAÍRÁSI SZABÁLY



Programozási technikák

Az újraírási szabály sok alkalmazásban használható programozási technika. A szabály azt mondja ki, hogy valahányszor a program beolvas valamit, azt helyettesítenie kell valami mással. Például, az az újraírási szabály, hogy „a TEHEN-et írd újra DISZNÓként",

A tehén átugrotta a Holdat

input mondatot

A disznó átugrotta a Holdat

output mondattá alakítaná át.

Általános felhasználási területei az automaták és a grammatikák. Sok *automatának* nevezett egyszerű elméleti komputer újraírási szabályokkal működik, amennyiben egy „szalagról" beolvas egy karaktersorozatot, és az átalakított karaktersorozatot kiírja ugyanarra a szalagra.

Bizonyos (elméleti) nyelvek, így az úgynevezett környezetfüggetlen nyelvek *grammatikája* is újraírási szabályokkal adható meg. Itt az M mondatszimbólumból indulunk ki. Az elemző egyik szabálya: „írd újra M-et FcsIcs-ként", ahol Fcs a „főnévi csoport" és Ics az „igei csoport" szimbóluma. A másik szabály: „írjuk újra Fcs-t NévelőFőnévként", az elemző tehát Névelő-Főnév-Ics-t ír és így tovább. Végül az M szimbólum helyén már egy kifejtett szimbólumsorozat áll. Mindez az elemző eljárásának egy másfajta leírása, de a számítógép tényleges működésére is kihatással van.

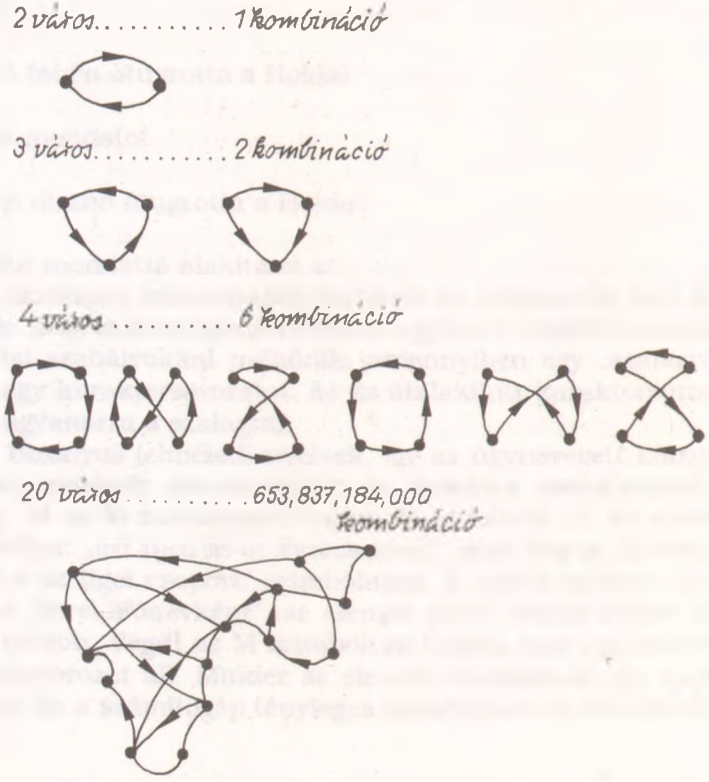
A függvényszervezésű nyelvek feldolgozása is történhet ilyen módon. A „faktoriális" függvényt egy sor újraírási szabállyal dolgozhatjuk fel a következőképpen:

3 faktoriálist írd újra
3 * 2 faktoriálisként, ezt írd újra
3 * 2 * 1 faktoriálisként, amit írj újra
3 * 2 * 1-ként, ami = 6.

Lásd még: Grammatika

AZ UTAZÓ ÜGYNÖK PROBLÉMÁJA

E közismert probléma megoldásával számos MI-probléma-megoldó módszert szokás tesztelni. Egy kereskedelmi ügynöknek körbe kell utaznia néhány várost, a kiindulóhelyre visszaérkezve, de a



többi városban csak egyszer járhat. Melyik a legolcsóbb útvonal, ha ismert az út költsége bármely két város között?

A nehézséget itt a probléma potenciális nagysága jelenti: ha nagyszámú városnál minden lehetséges útvonalat meg akarunk vizsgálni, kombinatorikai robbanás lép fel. A problémát nem lehet kisebb részproblémákra felosztani, mert az, hogy legközelebb melyik városba menjünk, függ attól is, hogy eddig melyekben jártunk. Ez a kölcsönös összefüggőség azt is sejteti, hogy a neuronhálózatok ideálisak erre a problémára, szemben a hagyományos játéklejátszó megközelítésekkel. Erre utal az is, hogy J. J. Hopfield Utazó Ügynök Probléma neuronhálózata néhány gépi ciklus alatt majdnem optimális megoldást talál a problémára.

VIRTUÁLIS GÉP

Általános számítógépes kifejezés

A virtuális (látszólagos) gép azt jelenti, ahogy a számítógép a felhasználó előtt megjelenik, szemben azzal, ahogy ténylegesen működik. Például egy időosztásos gép felhasználója, miközben a valóságban 46 másik felhasználóval osztozik, megeshet, hogy ebből semmit nem érzékel, gépe látszólag megszakítatlanul dolgozik. Hasonlóképpen, minden felhasználónak nagy „virtuális memória" áll rendelkezésre, míg a valóságban a kis memóriaterületet egy sokkal nagyobb „álmemória" egészíti ki a mágneslemezen.

Hasonló, bár kissé távol eső értelemben virtuális gépnek nevezik az egyik számítógép szimulálását egy másikon. A felhasználó azt hiszi, hogy a szimulált géppel dolgozik, miközben egy egészen másfajta gép előtt ül, amely viszont logikailag azonos módon működik, hiszen a másik teljes gépi kódú utasításkészletét szimulálja.

VISSZA-KÖVETKEZTETÉS

Logika

A visszakövetkeztetés az a fajta gondolatmenet, amely a következményeket az okokkal magyarázza, szemben a *dedukcióval* vagy *levezetéssel*, amely az okból kiindulva jut el a szükségszerű következményekig. Például tudjuk, hogy ha egy autóval valamekkora utat teszünk meg, a motor felmelegszik. Ha tehát felnyitjuk a kocsi motorháztetőjét, és tapasztaljuk, hogy a motor meleg, visszakövetkeztethetünk arra, hogy valaki vezette. Ez persze nem csalhatatlan következtetés, mert más okok is vezethettek a motor felmelegedésére. Bayesi logika segítségével az ilyesfajta magyarázat valószínűségét numerikusan is megbecsülhetjük, ilyen következtetési módszert a szakértői rendszerek is sokszor alkalmaznak.

A visszakövetkeztetés leghíresebb szakembere a Baker Street 221B szám fiktív lakója volt. Sherlock Holmes szinte sohasem alkalmazott dedukciót; mindig a következményektől jutott el a (sokszor rendkívül valószínűtlen) eseményekig, amelyek hozzájuk vezettek. Amint Dr. Watson rámutatott, csoda, hogy soha nem tévedett: az igazi következtetés távolról sem ennyire csalhatatlan.

Lásd még: Levezetés, Bayesi logika

VISSZALÉPÉS

Keresés

A visszalépés a kereséssel kapcsolatos kifejezés. Számos programban, ahol a komputernek kell bejárnia egy keresési teret, és eközben döntéseket hoznia, a visszalépés teszi lehetővé, hogy „újra próbálkozhasson", ha egyszer hibázott.

A logikai programozásban különösen fontos a visszalépés, amikor konjunkcióval összekapcsolt feltételekhez keresünk minden lehetséges megoldást. Vegyük például a következő szabályt:

X lánya Y ha:

Y apja X

1. feltétel

és Y nőnemű

2. feltétel

Az adatbázisból tudjuk, hogy

Edward anyja Katherine ;
Rose anyja Katherine;
John apja Henry;
John anyja Ann;
Edward apja Henry;
Sally anyja Rose;

Mary nőnemű;
Katherine nőnemű;
Ann nőnemű;

Rose nőnemű;
Henry hímnemű;
John hímnemű;
Edward hímnemű;
Sally nőnemű.

A komputer, miközben anya–lány kapcsolatokat keres, X-et és Y-t az adatbázisból vett szimbólumokhoz „köti". A konjunktív feltételek mindegyik részét sorra kell venni; elképzelhetjük tehát, hogy első kísérletre X-et Katherine-hez, Y-t pedig Edwardhoz köti.

A második feltétel kiértékelésekor azonban ez a kombináció nyilvánvalóan elesik, mivel Edward hímnemű, nekünk pedig nőnemű Y-ra van szükségünk. A program ekkor visszalép, és felszabadítja az Edwardhoz kötött Y változót.

Következő próbálkozásra X-et Katherine-hez, Y-t Rose-hoz kötjük. Most a konjunktív feltétel mindkét része teljesül. X és Y lekötésére azonban minden kombinációt ki kell próbálni, mielőtt az eljárás véget érne.

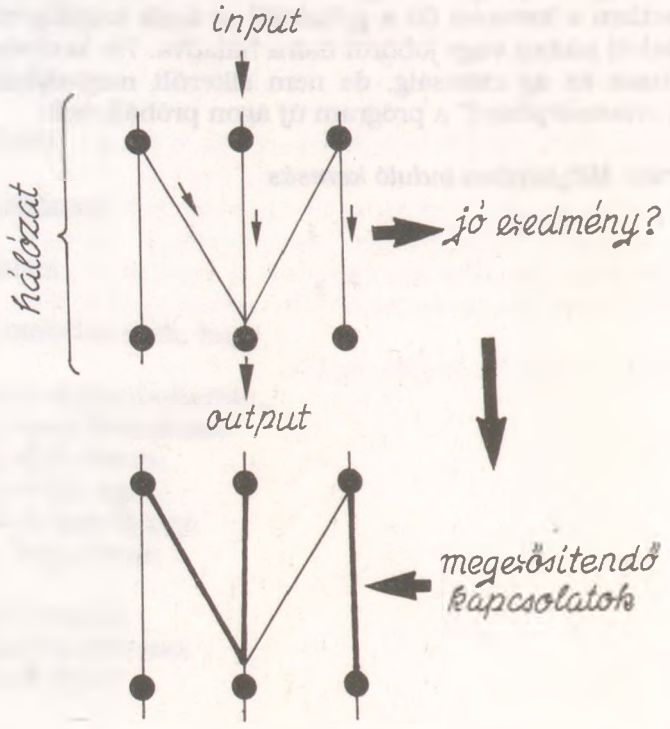
A keresésben általánosan használjuk a visszalépést. A keresési fát vagy mélységben vagy szintben indulva lehet bejárni. Az első esetben a keresési fát a gyökértől az ágak hegyéig vizsgáljuk meg, balról jobbra vagy jobbról balra haladva. Ha keresés közben eljutottunk az ág csúcsáig, de nem sikerült megtalálni a megoldást, „visszalépéssel" a program új ágon próbálkozik.

Lásd még: Mélységben induló keresés

VISSZA-TÁPLÁLÁS

Neuron-
hálózatok

A visszatáplálás a legalább két neuronréteggel (tehát „rejtett rétegekkel" is) rendelkező hálózatok tanulási módszere. Adott input mintázat mellett a valóságos outputot egybevetik az elvárt outputtal. A kettő közti különbséget azután „visszatáplálják" azokba az összeköttetésekbe, amelyek ezt az outputot szolgáltatták. Jó egyezésnél megerősítődnek a kapcsolatok azok között a sejtek



között, amelyek hozzájárultak az outputhoz. Ha rossz az egyezés, a szóban forgó sejtek összeköttetései gyengülnek; legközelebb, ha ugyanez az inputminta alakul ki, ezeknek kisebb hatásuk lesz a rejtett rétegekre és az outputra, mint eddig. Egy újabb változatban a rejtett réteg automatikus visszacsatolást küld az inputrétegbe, amely így végrehajtja ezeket a változtatásokat. Ezt nevezzük „recirkulációs" algoritmusnak.

Ez a megerősítésekkel működő tanulási módszer az emberi tanulás bizonyos jellegzetességeit mutatja, például azt, ahogyan az anyanyelvet elsajátító gyermek túlzottan általánosítja a nyelvtani szabályokat. „Én is jöszök a boltba" mondhatná egy két-három éves kisgyerek, aki még nem tudja, hogy a „jönni" igét rendhagyó módon kell ragozni.

A visszatáplálás csak arra az esetre szorítkozik, amikor a neuronhálózat már közel jutott a „céljához", vagy jó úton van hozzá. Amikor ez a feltétel nem teljesül, a hűtés módszerét kell alkalmazni.

Lásd még: Connection-elv, Neuronhálózatok

ZÁRTVILÁG-FELTEVÉS

Keresés

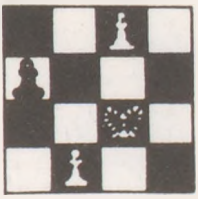
A zártvilág-feltevés azt mondja ki, hogy egy rendszerben ábrázolt információ abszolút teljes. Mint amikor egy detektívregény végén már minden lehetséges tényt megismertünk, pusztán logika kérdése, hogy megvizsgáljuk a rendelkezésre álló bizonyítékokat, és levonjuk a szükségszerű következtetést.

A matematikai logikában azt feltételezzük, hogy egy gondolatmenet axiómái minden szükséges tényt tartalmaznak, amelyből el lehet jutni a következtetésekig: a matematikai logika világa „zárt". A valós életproblémák egészen bizonyosan nem ilyenek: ritkán vannak pontosan meghatározva, így nem könnyű őket logikailag ábrázolni. Nem meglepő tehát, hogy a számítógépek, amelyek közvetlenül logikai elveken alapulnak, nehezen birkóznak meg a valós életproblémákkal. Sok esetben a problémák köznapi megoldása pusztán azért nem jöhet szóba, mert kívül esik azon a zárt világon, amelyben a számítógép működik.

Sok adatbázis is egy „zárt világban" működik. Egy bizonyos légitársaság járatait tartalmazó adatbázis csak azokat a városokat sorolja fel, amelyeket a gépek érintenek, a többit nem. Az adatbázis kénytelen a zárt világ szemléletét elfogadni, mert messze ez a legpraktikusabb. A negatív tények száma sokszorosan meghaladja a pozitív tényekét. Nem lenne ésszerű minden negatív tényt kimondani.

Lásd még: Logikai programozás, Explicit és implicit tudás, Alapfeltevéses következtetés

ZÉRÓ ÖSSZEGŰ JÁTÉKOK



Játékok
és játék-
tartományok

„Zéró összegű" vagy „konstans összegű" játéknak nevezik a játékok egy fontos típusát. Ez a kifejezés a játékelméletből származik. A zéró összegű játékban az elnyerhető összeg konstans, ha tehát az egyik játékos többet nyer, a másik szükségképpen többet veszít. Például a választás zéró összegű játék: ha az egyik jelölt nyer, a másik *mindenképpen* veszít. A zéró összegű játék gondolata lényeges szerepet játszik sokféle problémánál, a populációgenetikában, a tömegek irányításában, a választásokban, de leginkább játékokra alkalmazzák.

Ehhez kapcsolódik a *szimmetrikus játék* (és ellentéte, az *aszimmetrikus játék*) fogalma. A szimmetrikus játékban a játékosok teljesen egyenlőek, vagyis mindkettőjük előtt ugyanazok a választások állnak, tehát mindegy, melyikük választ. A sakk például ilyen játék: mindkét játékosnak ugyanazok a lépései vannak, tehát mindegy, melyiküket nevezzük fehérnek és melyiküket feketének. A póker viszont aszimmetrikus játék, mert az egyik játékos, mint osztó, előnyben van. (Valójában a sakk sem teljesen szimmetrikus, hiszen az egyik játékosnak kell kezdenie.)

A. függelék: Alapvető MI-programok

Vannak olyan MI programok, amelyek alapvetőek az MI valamelyik területén. Számtalanszor hivatkoznak problémamegoldó módszereikre, ezért itt felsoroljuk őket keletkezési dátumukkal, főbb szerzőikkel együtt (ahol egy nagyobb projekt eredményeként jöttek létre, a közreműködő szervezetet adjuk meg), feltüntetve azt az MI területet is, amelyre jellemzőek.

Program	*Dátum*	*Szerző(k)*	*Leírás*
ACRONYM	1981.	Brooks	Modellalapú, tartománytól független értelmező rendszer. Félig program, félig nyelv tárgyak azonosítására és osztályozására, adatbázissal vagy grafikus leírással való összehasonlítására.
CHI	1981.	Kestrel Institute	Automatikus programozási eszköz, Különböző módszerek alkalmazásával „barátságos", interaktív természetesnyelvi programozási környezetet szolgáltat. A PSI utódja.

Program	*Dátum*	*Szerző(k)*	*Leírás*
CONGEN	1976.	Stanford University	A ciklikus kémiai struktúrákkal foglalkozó DENDRAL felújított változata.
DEDELUS	1975.	Richard Waldinger és Zohar Manna	Automatikus programozási rendszer, magas szintű, teljes logikai programspecifikációból azt kielégítő Lisp-programot készít.
DENDRAL	1965.	Stanford University	Heurisztikus szakértői rendszer nem ciklikus kémiai vegyületek struktúrájának meghatározására a tömegspektrum alapján. Viszonylag egyszerű algoritmus generálja az összes lehetséges struktúrát: a program a tömegspektrográfia alapján leszűkíti a szóbajöhető struktúrák számát.
ELIZA	1966.	Joseph Weizenbaum	Korai természetesnyelv-feldolgozó program, egy nemdirektív pszichoanalitikus modorában a beírt szövegből kiválasztotta a kulcsszavakat, és közkeletű frázisok módosított változatait nyomtatta ki.
EMYCIN	1980.	Stanford University	Szakértői rendszer shell, a MYCIN vezérlő struktúrájával működött, a fertőző betegségek adatbázisa nélkül.

Program	*Dátum*	*Szerző(k)*	*Leírás*
GPS	1957.	Newell, Shaw és Simon	Általános problémamegoldó, eszköz-cél elemzéssel oldott meg alkalmasan megfogalmazott problémákat. Amikor tárgyának belső reprezentációját általánosítani próbálta, nehézségekbe ütközött.
GUIDON	1978.	William Clancey	CAI (Számítógéppel segített oktatás) program fertőző betegségekhez, a MYCIN adatbázisára épült, és egy 200 betanító szabályt tartalmazó külön adatbázis segítette az oktatási funkciókat. Belső modellt állított fel a tanuló részleges tudásáról is.
HACKER	1975.	Gerald Sussman	Tanulórendszer, amely maga is megtanul térképet készíteni (hogy segítsen egy robotnak építőkockákat egymásra tenni). Szimulált próbálkozások alapján „szabad–tilos" listát állított össze. Modellként szolgál a programozás tanulására.
HARPY	1975.	Carnegie Mellon University	Beszédmegértő program, a felismerendő mondatokról előre felhalmozott szemantikai, szintaktikai és fonetikai tudást tartalmaz.

Program	*Dátum*	*Szerző(k)*	*Leírás*
HEARSAY	1976.	ARPA által finanszírozott program	Táblastruktúrán alapuló beszédmegértő program független modulokat tartalmazott az akusztikai/ fonetikai, szintaktikai és szemantikai tudásról.
INTERNIST	1974.	H. Pople és J. Myers	Szakértői rendszer belgyógyászati konzultációhoz. A betegek panaszaiból kiindulva a betegségeket és tüneteket tartalmazó fastruktúrájú adatbázis alapján összeállítja a lehetséges betegségek listáját.
LEX	1981.	Thomas Mitchell	Egyszerű matematikai (integrálási) feladatokat tanul meg megoldani egyszerű heurisztikus szabályokat fedez fel, hogy mikor kell alkalmazni a beépített alapvető integrálási szabályokat.
LUNAR	1972.	Bolt, Beranek és Newman Inc.	Természetes nyelvi rendszer, amely egy ATN elemző segítségével angol kérdéseket formális kérdőívnyelvre fordít, és információt szerez az Apollo 11 Hold-kőzetmintáit leíró adatbázisból. Ez tehát egy természetesnyelvi „homlokzat".

Program	*Dátum*	*Szerző(k)*	*Leírás*
MACIE	1986.		Mátrixvezérlésű következtetőgép. Connection-elvű tudásbázis modelleknél használható önálló következtetőgép.
MACSYMA	1971.	MIT és felhasználók	Nagyméretű interaktív szimbólum-manipulációs program matematikai problémamegoldáshoz. A program több mint 600 matematikai műveletre képes, így lineáris egyenletmegoldásra, Taylor-sorfejtésre, mátrix- és vektorműveletekre. Saját pogramnyelve van, és óriási tudásbázissal rendelkezik.
META-DENTRAL	1978.	Stanford University	A DENDRAL által használható szabályok előállítására szolgáló program.

Program	*Dátum*	*Szerző(k)*	*Leírás*
MYCIN	1976.	Stanford University	Konzultatív szakértői rendszer fertőző betegségek diagnosztizálására. Valószínűleg a leghíresebb szakértői rendszer. A tudás rögzítésére produkciós szabályokat használ, amelyek segítségével bizonyossági értékeket rendel az egyes diagnózisokhoz. A következtetéshez hátraláncolást alkalmaz, szabálybázisában kiterjedt mélységben induló kereséssel kapja meg a releváns szabályt.
PROSPECTOR	1978.	SRI International	Konzultációs szakértői rendszer geológusok támogatására. A tudást olyan hálózatban tárolja, ahol a csúcsok: kijelentések és az élek: következtetési szabályok. 1982-ben lett híressé, egy terepkutatás értelmezésében túltett a geológusokon.

Program	*Dátum*	*Szerző(k)*	*Leírás*
PSI	1979.	Cordell Green és mások	Automatikus programozási rendszer, módszerek széles skáláját foglalja össze egy „támogató programozási környezetben". A program specifikációja vegyes kezdeményezésű dialógussal történik, a különböző tudásterületek „szakértőinek" szoros együttműködésével.
QMR	1980.	Popple	Az INTERNIST mikroszámítógépes változata. Keretalapú adatbázist használ.
SCHOLAR	1970.	Bolt, Beranek és Newman Inc.	Tanítórendszer Dél-Amerika földrajzának oktatására. Vegyes kezdeményezésű angol mondatokkal teszteli, és szemantikai hálóban tárolja a tanulók tudását. Nem él a zártvilág-feltevéssel, ezért nem teljes adatbázist is elfogad.

Program	*Dátum*	*Szerző(k)*	*Leírás*
SHRDLU	1972.	Terry Winograd	Természetesnyelvi rendszer, amely képes a benne reprezentált „építőkocka-világról" beszélgetést folytatni. Az elemző egy rendszernyelvtanon működött. Az SHRDLU a tartomány leszűkítésével és hatékony problémamegoldó technikák alkalmazásával kerülte ki az angol nyelv bonyolultabb jelenségeit: kétséges, hogy bővebb tartományokra ki lehetne-e terjeszteni.
SOPHIE	1975.	Bolt, Beranek és Newman Inc.	Elektronikával foglalkozó CAI-program. A tanár és a szimulált munkapad kombinációjaként működik. Természetes nyelvi felülete egy szemantikus nyelvtan segítségével mondja meg a tanulónak, hogy az általa tervezett áramkör mit csinál, valamint teszteli és megbírálja a tanuló hipotéziseit, hogy mi okozza a kapott eredményeket.

Program	*Dátum*	*Szerző(k)*	*Leírás*
STRIPS	1971	Richard Fikes és Nils Nilsson	Problémamegoldó program, amely predikátum-kalkulus segítségével old meg több termen keresztül mozgó robotokkal kapcsolatos problémákat. A szobák tartalmának tényleges és megkívánt elrendezését állapotgráfban ábrázolja, és ebben keres alkalmas utat a pillanatnyi, illetve a célállapot között.
SYSTRAN	1978.		Gépi fordítással foglalkozó programcsomag. A mondatokat részlegesen egy belső ábrázolásra, majd innen egy másik nyelv felszíni szerkezetére fordítja.
TEIRESIAS	1975.	Randall Davis	Adatok nagy adatbázisokba való bevitelét támogató rendszer. A TEIRESIAS tanulóprogram, amely egy szakértői rendszerből egy adatbázisba (eredetileg a MYCIN-be) gyűjti a tudást. Belső metaszabályok irányítják az adatbáziselemek elsajátítását.

Program	*Dátum*	*Szerző(k)*	*Leírás*
WUSOR	1979.	Ira Goldstein és Brian Carr	A WUMPUS számítógépes játék betanítására szolgáló gyakorlóprogram. Belső modellel rendelkezik a tanulóról, amely egy játék szakértői modulra támaszkodik, és a tanulás irányításához így szolgáltat információt a játék, a pszichológia és a tanár modulokról.
XCON	1978.	Digital Equipment Corporation	A VAX 11/780 gép konfigurációjára szolgáló szakértői rendszer. R1 néven is ismeretes. Megszerkesztésében számos technikát felhasználtak (és 1986-87-ben teljesen átírták).

B. függelék: MI-nyelvek és környezetek

Van néhány nyelv és programozási környezet, amely sokszor előfordul az MI-vel összefüggésben. Alább felsoroljuk a leggyakoribbakat néhány jellemzőjükkel együtt.

Nyelv/környezet	*Leírás*
ART	Szakértői rendszerek kifejlesztésére szolgáló programozási környezet, „shell"-je kész modulokat tartalmaz. A tudást keretekkel, szabályokkal és eljárásokkal ábrázolja, sok nyelvben objektumorientált „szótárral" integrálódik. A grafikus felület valós időben logikai következtetéseket végez.
Conniver	Adatbázisrendszer, az információt kontextusfával ábrázolja, amely egyidejűleg mutatja be a lehetséges történések különböző hatásait. Mintaillesztő képességekkel is rendelkezik.
Expert	Szakértőirendszer-készítő eszköz, szabályalapú rendszerekhez. Külön eszközöket tartalmaz tesztesetek bevitelére és elemzésére, illetve ezek alapján új szabályok felállítására. Gyors prototípuskészítésre is alkalmas, és magas szintű szabálybeviteli nyelvet tartalmaz. Fortranban készült.

Nyelv/környezet	*Leírás*
InterLisp	A Lisp egyik gyakori változata, a BBN és Xerox fejlesztése. Külön módszereket tartalmaz a felhasználó parancsainak, eljárásainak és filecsomagjainak kezelésére.
IPL	Az első listafeldolgozó nyelv, 1957-ben fejlesztették ki; itt jelent meg a sejtekből álló lista, és a lista minden elemét előállító generátor gondolata. A GPS projektet IPL-ben írták.
KEE	„Knowledge engineering environment" („Tudástervezői környezet"). Fejlett tudás- és adatbázis-modellező és -következtető eszköz. A keretalapú tudásreprezentációt, a szabályalapú következtetést, a Lispet és az interaktív grafikát kombinálja, eközben kiterjedten alkalmazza az objektumorientált programozási modelleket is.
Linda	Programnyelv a Connection Machine-nel kapcsolatos kutatásokhoz, Reprezentációit n-esek terében helyezi el, a program elemei szabadon és egymástól függetlenül párhuzamosan működnek együtt.
Lisp	Listastruktúrákon alapuló nyelv. Lásd a vonatkozó szócikket.
Loops	Általános eszköz tudásbázis-programok készítésére. A Loop eljárásorientált, objektumorientált és hozzáférésorientált programozási gondolatokat kombinál szabályalapú programozási környezetben.
Occam	Az Inmos transzputer-chip párhuzamos számítási képességeinek kihasználására kifejlesztett nyelv.

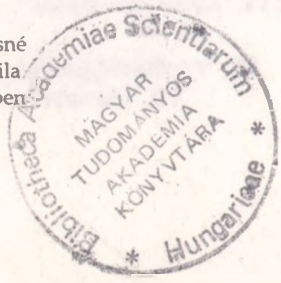
Nyelv/környezet	*Leírás*
OPS5	Általános célú produkciós szabályokon alapuló nyelv. Egyetlen munkamemóriában tárolja a bevitel és módosítás ideje szerint rendezett adatokat, és ennek révén oldja fel az ütközéseket. A szabályértelmező rendkívül gyors, de általában elég nehéz felületet írni hozzá. Az R1 is OPS5-ben készült.
Planner	Az adatbázisokban ábrázolt tudás eljárásalapú ábrázolására szolgáló nyelv. A programozó a tudást megadott körülmények között követendő eljárásokról szóló kijelentések halmazával („tételekkel") fejezi ki, Automatikus visszalépésre képes. Megvalósítására Terry Winograd SHRDLU-jában került sor Micro-Planner néven.
POP-2	Angliában népszerű MI-nyelv, másutt alig használják. A Lisp több jellegzetességével rendelkezik, de generátorként működő (lásd IPL) dinamikus listákat is képes kezelni, vannak bizonyos vezérlőstruktúra kibővítései is.
Prolog	Predikátum-kalkuluson alapuló nyelv. Lásd a vonatkozó szócikkben.
Sail	„Stanford Artificial Intelligence Language" („Stanford MI-Nyelv"). Az Algol 60-on (a Fortran mára kihalt kortársán) alapuló nyelv, amelynek „hagyományos" vezérlő struktúrája már képes listák, makrók, korutinok, hatékony asszociatív adatvisszakereső eszközök és interaktív hibajavító eszközök használatára.

Címszavak angol mutatója

Chinese room	Kínai szoba 96
Classic puzzles	Klasszikus fejtörők 100
Closed world assumption	Zártvilág-feltevés 234
Combinatorial explosion	Kombinatorikai robbanás 102
Compiled knowledge	Felhalmozott tudás 50
Compiler	Compiler 22
Computer aided instruction	Számítógéppel segített oktatás 186
Concurrency	Együttfutás 29
Connection machine	Connection machine 25
Connectionism	Connection-elv 23
Cooperative system	Kooperatív rendszer 104
Cybernetics	Kibernetika 91
DARPA	DARPA 26
Dataflow	Adatfolyamat 1
Declarative programming language	Deklaratív programnyelvek 28
Deduction	Dedukció 27
Default reasoning	Alapfeltevéses következtetés 2
Delivery system	Futtató rendszer 53
Depth	Mélység 124
Depth first search	Mélységben induló keresés 126
Derivation tree	Levezetési fa 109
Development environment	Fejlesztői környezet 47
Development tool	Fejlesztői eszköz 46
Distributed processing	Osztott feldolgozás 152
Domain	Tartomány 204
Edge detection	Élkeresés 42
Elastic logic	Elasztikus logika 30
Epistemology	Ismeretelmélet 79
ESPRIT	ESPRIT 39
Experimental system	Kísérleti rendszer 97
Expert system shell	Szakértői rendszer shell 179
Expert systems	Szakértői rendszerek 181
Explanation system	Magyarázó rendszer 122
Explicit & tacit knowledge	Explicit és implicit tudás 40
Fifth generation projects	Ötödik generációs projektek 153
Forward chaining	Előre láncolás 34

Frame	Keret 89
Frame store	Képkocka tároló 86
Front end	Homlokzat 72
Game tree	Játékok fája 80
Genetic algorithm	Genetikus algoritmus 54
Grammar	Grammatika 58
Grey level image	Szürkefokozatos kép 200
Hamming net	Hamming-háló 61
Heuristic	Heurisztika 66
Hill-climbing	Hegymászás 64
Hopfield-net	Hopfield-háló 73
Human factors research	Emberi tényező kutatása 36
Hypercube	Hiperkocka 69
Hypermedia	Hipermédia 70
Induction	Indukció 75
Inference engine	Következtetőgép 106
Interpreter	Interpreter 77
Intuition	Intuíció 78
Knowledge base	Tudásbázis 216
Knowledge elicitation	Tudáskigyűjtés 218
Knowledge engineering	Tudástervezés 220
Learning	Tanulás 201
Lexicon	Lexikon 111
Lighthill report	Lighthill-jelentés 112
Lisp	Lisp 113
Lisp chip (lisp machine)	Lisp chip (Lisp-gép) 115
List processing	Listafeldolgozás 116
Logic programming	Logikai programozás 119
Machine translation	Gépi fordítás 56
Meta knowledge	Metatudás 129
Meta rule	Metaszabály 128
Minimax technique	Minimax technika 131
Model based vision	Modell alapú látás 133
Monotonic reasoning	Monoton következtetés 134
Naive physics	Naiv fizika 137
Nanotechnology	Nanotechnológia 139

A kiadásért felelős az Akadémiai Kiadó és Nyomda igazgatója
A nyomdai munkálatokat az Akadémiai Kiadó és Nyomda végezte
Felelős vezető: Zöld Ferenc igazgató
Budapest, 1994
Nyomdai táskaszám: 23381
Felelős szerkesztő: Rátz Miklós
Műszaki szerkesztő: Nyárádi Tamásné
A fedéltervet készítette: Lőrincz Attila
Megjelent: 24,3 (A/5) ív terjedelemben



A könyv 13 témacsoportban 160 címszó tömör ismertetését tartalmazza a mesterséges intelligencia tárgykörében. A témacsoportok a következők: elmélet, hardver, játékok, képfeldolgozás, keresési eljárások, logika, a mesterséges intelligencia általában, neuronhálózatok, a pénzügyi támogatás intézményei, programozási módszerek, szakértői rendszerek, a számítástechnika általában és a természetes nyelvek kezelése. Ez a „kislexikon" azokhoz szól, akik ugyan nem foglalkoznak hivatásszerűen a mesterséges intelligenciával, de munkájukban lépten-nyomon beletkőznek e szakterület problémáiba, s akarva-akaratlan használják – gyakran pontatlanul – ezeket a fogalmakat, kifejezéseket. Egyszóval: a munka hasznos lehet minden számítógép-használónak, legyen bár tíz- vagy ötvenéves, ügyviteli dolgozó, orvos vagy gazdasági szakember.

Ára: 905,– Ft áfával

1828