Gruber Kristóf
YCN70J

Mesterséges Intelligencia házifeladat

Programs with Common Sense (John McCarthy cikke)

2006.12.10.

John McCarthy-ról

A cikk leírása előtt mindenképpen érdemes pár szót ejtenünk a szerzőről, aki munkásságával jelentősen hozzájárult a mesterséges intelligencia szakterületének fejlődéséhez. Érdekességképpen megemlítjük, hogy John McCarthy jegyzi az „*Artifical Intelligence*" (mesterséges intelligencia) kifejezést is.

John McCarthy 1927-ben született Bostonban, 1948-ban a Caltech-en szerzett matematikus diplomát, majd 1951-ben a Princeton Egyetemen doktorált. Ezután tanárként helyezkedett el a Darthmouth College-ban, amit máig a szakterület szülőhelyének tartanak. 1956 nyarán Minsky (Harvard), Nathaniel Rochester (IBM), és az információelméletet kidolgozó Claude Shannon (Bell) segítségével a neurális hálók, az automaták, valamint a kapcsolódó területek kutatói számára két hónapos munkatalálkozót szervezett, „*Kutatási projekt a mesterséges intelligenciáról"* (*Research Project on Artificial Intelligence*) címmel. A találkozó történelminek bizonyult: a modern tudomány a Darthmouth Konferencia óta jegyzi ezen a néven külön tudományágként a mesterséges intelligenciát. Az ott részt vevő előadók és tanítványaik lettek később azok, akik az elkövetkező évek MI kutatásának fő irányvonalait megszabták.

Az ötvenes évek végén McCarthy és Minsky megalapította az MIT MI Laboratóriumát, majd 1963-ban, immáron egyedül a Stanford-on az MI-labort, melyet 1980-ig igazgatott.

1970-ben megkapta a Turing díjat, 2003-ban pedig elnyerte a Számítógépes és Kognitív Tudományok Benjamin Franklin Emlékérmét, melyet a szakterület legnagyobbjai közt osztanak ki.

Fontosabb munkái

McCarthy 1958-ban kidolgozta a LISP *(List Processing Language, Listafeldolgozó Nyelv)* nyelvet, amely egy függvényalapú, számítási műveleteket számok helyett inkább szimbolikus kifejezésekkel végző programozási nyelv. A LISP a második legidősebb, még ma is használatban lévő programozási nyelv (A FORTRAN után), bár azóta rengeteg változáson ment már keresztül.

Saját munkájának megkönnyítésére munkatársaival közösen kifejlesztették az akkori nagyszámítógépekre az időosztásos *(time-sharing)* operációs rendszer alapelveit. Ez lehetővé tette, hogy több felhasználó egyidejűleg használhassa ugyanazt a számítógépet, úgy, hogy közben az legyen az érzésük, hogy mindegyikük egyedül használja csak a gépet. Ezt a számítógép úgy érte el, hogy az értékes processzor időt szeletekre osztotta, és egy-egy felhasználó azt csinálhatott a saját időszeletében, amit éppen akart. Ha a gép kellőképpen gyorsan váltott a szeletek között, akkor a programok futása folyamatosság látszatát keltette. Ezzel sokszorosára növelve az akkori drága gépidő kihasználtságát.

Szintén 1958-ban jelent meg a *Programs with Common Sense* című dolgozata (jelen írás témája), melyben a sokak által első teljes MI-rendszernek tekintett hipotetikus *Advice Taker* programot vázolta fel.

Az Advice Taker

A projekt fő célja a következő volt: egy „*tanács elfogadó"*-nak *(advice taker)* nevezett program létrehozása, amely képes az emberhez hasonlóan, intelligensen viselkedni a tudása alapján, ezt a tudást pedig képes tanulással bővíteni. A program imperatív vagy deklaratív mondatokat fog kikövetkeztetni a környezetéről; ha imperatív egy mondat, akkor végrehajtja a megfelelő utasítást, ha pedig deklaratív, akkor bővíti vele a tudását. A program és a felhasználói közt (mai szemmel természetesnek mondhatjuk) a számítógép perifériáin keresztül folyik a párbeszéd.

Úgy képzelték, hogy az *advice taker* egy olyan program lesz, amely az elé állított problémákat formális nyelvek mondatainak manipulálásával fogja megoldani. Több hasonló rendszer is létezett már a tanulmány írásakor, például Allen Newell és Herbert Simon Logic Theorist-je (1955), illetve Herbert Gelernter Geometriai tételbizonyítója (1959). Az *advice taker* és elődei között viszont az volt a fő különbség, hogy az eddigi programokban a formális rendszer maga volt a kutatás fő célja, a levezetés pedig a programban testesült meg, ezzel ellentétben az *advice taker* annyi procedúrát próbál meg leírni magában a nyelvben, amennyi csak lehetséges, valamint bizonyos esetekben még magukat a levezetéseket is igyekszik így reprezentálni.

Az *advice taker* fő előnye, hogy nagyon egyszerűen fejleszthető lesz: elegendő csupán állításokat kimondani, leírni azok szimbolikus környezetét, és hogy mit várunk tőle. Egy új állítást hozzáadni a rendszerhez minimális előzetes tudást igényel csak a programtól. Az *advice taker* a tudásából logikai következtetéseket von le, ezek segítségével kialakít magában egy olyan tudást, ami hasonlít az emberek esetében a „józan észhez", vagyis elmondhatjuk, hogy *egy program „józan ésszel" gondolkodik, ha önállóan következtetéseket von le a saját tudásából és tapasztalataiból.*

Az egyik legfőbb cél olyan programok megalkotása volt, amik képesek tanulni a saját tapasztalataiból az emberhez hasonlóan. Erre az egyik legjobb példa Samuel sakkprogramja. A program sok mindent figyelembe vesz a lépésének meghatározása előtt, például visszaemlékszik az előző játékokra is, ahol esetleg egy adott lépés rossz következményekkel

járt. Saját előzetes tudása alapján változtatni tud a viselkedésén, és sokkal célravezetőbb lépéseket tud tenni.

Ahhoz, hogy elvárhassuk egy géptől, hogy egy absztrakciót felfedezzen, ezt az absztrakciót valahogy kellőképpen egyszerűen kell leírnunk.

Az alábbi dolgokat tartottuk a legfontosabb kérdéseknek, amikre egy ilyen rendszernek képesnek kell lennie:

- Az egész intelligenciának modellezhetőnek kell lennie egy számítógépes rendszerben. Ez lehet egy automata, vagy egy általános célú programozási nyelv.
- A működés fontos változásainak egyszerűen kifejezhetőnek kell lenniük.
- A működés minden formájának fejleszthetőnek kell lennie, beleértve magát a fejlesztést is.
- A gépnek rendelkeznie kell, vagy ki kell tudnia fejleszteni terveket a részleges sikerről, mert az összetettebb problémák megoldásakor a teljes sikerek vagy sikertelen próbálkozások túl sűrűn következnek be.
- A rendszernek tudnia kell szubrutinokat készíteni, amik procedúrákban egységekként felhasználhatóak. A szubrutinok megértését az bonyolítja, hogy egy szubrutin kimenete általában nem „jó” vagy „rossz” önmagában, ezért a mechanizmusnak, ami a szubrutinok közül választ, képesnek kell lennie a szituációhoz legjobban illő, ill. leghasznosabb szubrutint kiválasztani.

A projekt a fenti 5 pontból leginkább a másodikra koncentrál. Úgy gondolták, hogy ahhoz, hogy egy program meg tudjon tanulni valamit, először is képesnek kell lennünk közölni vele dolgokat, és ez az elsődleges feladat, amire az első verzióknak főként koncentrálnia kell.

Az első nagy probléma, amibe a projekt ütközött, hogy a programnyelvek többnyire imperatív módon (felszólító módú utasításokból) épül fel, míg az emberi elme inkább deklaratívan (leíróan) működik. Alább láthatóak a két eltérő módszer előnyei összefoglalva:

Az imperatív mondatok előnyei

1. Egy imperatívan leírt mondat mindig gyorsabban végrehajtható és értelmezhető
2. Előzetes tudás nélkül is végre tudja hajtani az utasításokat a gép

A deklaratív mondatok előnyei

1. Az előzetes tudásból előny kovácsolható
2. A deklaratív állítások közt logikai összefüggés van, amit a gép átrendezhet és értékelhet úgy, hogy viszonylag egyszerű következtetéseket tudjon levonni belőlük.
3. Sokkal függetlenebb a deklaratív állítások értelme azok sorrendjétől, mint az imperatív állítások esetében, ezáltal sokkal könnyebb utánagondolni a dolgoknak.
4. A deklaratív állítások hatása függetlenebb a rendszer előző állapotától, így sokkal kevesebb információ is elégséges róla.

Az absztrakciók kifejezésének egyetlen általunk ismert módja a *nyelv*, ezért döntöttek úgy, hogy egy olyan rendszert fognak programozni, amely verbálisan (vagy legalábbis ahhoz nagyon hasonlóan) következtet.

Az Advice Taker működése

Az *advice taker*-nek a következő fő tulajdonságai vannak:

1. Kifejezések számítógépben reprezentálásra a következő rekurzív definíciót használjuk: A kifejezés elemek egy osztálya, ahol egy elem is egy kifejezés. Kifejezések sorozata is kifejezés. A kifejezéseket számítógépben listákkal reprezentáljuk (Newell és Simon, 1957).
2. Néhány kifejezést felfoghatunk deklaratív mondatként egy bizonyos logikai rendszerben, ami hasonlít az univerzális Post-féle egyszerűsített rendszerhez. A bizonyos kiválasztott rendszer programozási szempontoktól is függhet, de lehetséges, hogy egy egyszerű következtetési szabály alapján kombinálja a változókat a modus ponens-el. A kombinálás oka az, hogy megakadályozzuk a gép túlterhelését már kikövetkeztetett általános problémák speciális eseteivel.
3. Létezik egy *azonnali következtető rutin*, amely bizonyos előfeltételek teljesülése esetén azonnali következtetést tud levonni. Kezdetben ez az azonnali következtető rutin fogja leírni az előfeltételek minden egy lépésben kikövetkeztethető eredményét. Később, ahogy tudásbázis egyre bonyolultabb lesz, ez a rutin hozhat még néhány számunkra érdekes következtetést. Mindezek ellenére a rutin nem használja fel a levezetések jelentését.

Az *advice taker* intelligenciája nem fog csak az azonnali következtető rutin segítségével kifejlődni. Az intelligencia a procedúrákban fog majd kifejlődni, amik kiválasztják majd az előfeltételek listáit, amikre az azonnali következtető rutint alkalmazzuk. Természetesen a programnak sohasem szabad úgy használnia az azonnali következtetést, hogy minden általa ismert dologra megpróbálja meghívni, mert ez túl sok erőforrást igényelne.

4. Nem minden kifejezést értelmez a rendszer deklaratív mondatnak. Néhány formula egy objektumot reprezentál. Számunkra egy elem objektum, ha ki akarunk róla fejezni valamit, ami nem következtethető ki az elem nevéből. Pl. a 3812-es szám a legtöbb ember számára nem egy objektum, mert nincs mit mondaniuk róla azon kívül, ami magából a számból következik. Ellenpéldaként felhozhatjuk az 1848-as számot, ami a legtöbb magyar számára nem csak egy szám, hanem egyben azt az évet is jelenti, amikor a forradalom lezajlott, tehát objektum. Az *advice taker*-ben minden egyes objektum rendelkezik egy attribútum listával, amelyben szerepelnek az objektumra jellemző tulajdonságok, amennyiben a következtetés létrejött és elég bonyolult volt ahhoz, hogy a rendszer ne akarja legközelebb is újra kiértékelni.
5. Azokat az elemeket, amik nem olyan deklaratív mondatok, amik felírhatóak a rendszerben formulákkal, egyéneknek, funkcióknak és programoknak nevezzük.
6. A program működése ciklikus futásra korlátozódik, ami a következőképpen néz ki: Az azonnali következtető rutint alkalmazzuk az előfeltételek és az egyének listájára. A következtetések közül néhány imperatív formájú mondat lesz, ezeket végrehajtjuk. A kiértékelendő imperatív mondatok közé tartozik az a rutin is, ami kiértékel és végrehajt.

Lássunk egy példát az *advice taker* működésére: Képzeljük el, hogy én otthon ülök az asztalomnál és el szeretnék menni a reptérre. Az autóm szintén otthon van. A problémára a megoldás a következő: odamegyek az autómhoz, és elvezetem az autót a reptérig.

Először formális állításokat kell konstruálnunk az *advice taker* számára, hogy felvázolja a következtetéseit. Ezután beszélhetünk csak arról, hogy milyen levezetéseken keresztül jutott el az *advice taker* odáig, hogy összeállította az előfeltételekből a rendelkezésre álló tények összességét. Az előfeltételek csoportokba vannak foglalva, mi pedig minden egyes csoport jelentését el fogjuk magyarázni.

1. Először vegyünk fel egy „*ottvan*” előfeltételt az alábbi formában: ottvan(x,y), ez azt jelenti, hogy *x y* közelében van. Ezt felhasználva a következőket kapjuk:

$$ottvan(én, asztalom) \quad (1)$$

$$ottvan(asztalom, otthon) \quad (2)$$

$$ottvan(autó, otthon) \quad (3)$$

$$ottvan(otthon, környék) \quad (4)$$

$$ottvan(környék, reptér) \quad (5)$$

   Szükségünk lesz arra a tényre, hogy az ottvan reláció tranzitív az alábbi módon:

$$ottvan(x,y),\ ottvan(y,z) \rightarrow ottvan(x,z) \quad (6)$$

   Vagy ehelyett használhatunk egy sokkal absztraktabb relációt:

$$tranzitív(ottvan) \quad (7)$$

$$tranzitív(u) \rightarrow (u(x, y),\ u(y, z) \rightarrow u(x, z)) \quad (8)$$

   amiből a (6)-os állítást már le lehet vezetni.

2. Most két szabály következik a megvalósíthatóságra módjára vonatkozóan: sétálhatunk is és vezethetünk is:

$$járható(x),\ ottvan(y, x),\ ottvan(z,x),\ ottvan(én, y) \rightarrow tud(menni(y, z, gyalog)) \quad (9)$$

$$vezethető(x),\ ottvan(y, x),\ ottvan(z, z),\ ottvan(én, y) \rightarrow tud(menni(y, z, autóval)) \quad (10)$$

   Ezeken kívül van két speciális tény:

$$járható(otthon) \quad (11)$$

$$vezethető(környék) \quad (12)$$

3. Következzék egy szabály a „menés” tulajdonságairól:

$$csinált(menni(x, y, z)) \rightarrow ottvan(én, y) \quad (13)$$

4. A problémát magát szintén egy előfeltétel írja le:

$$akar(ottvan(én, reptér)) \quad (14)$$

5. A fenti előfeltételek erre a konkrét példára vonatkoztak. Most pedig következzen egy olyan, amely szinten minden hasonló problémánál hasonló:

$$(x \rightarrow akar(y)), (csinált(y) \rightarrow z) \rightarrow következmény(x, y, z) \quad (15)$$

A „*következmény(x, y, z)*” előfeltétel azt jelenti, hogy egy szituációban, amiben *x* érvényes az *y* cselekvést lehet végrehajtani, hogy egyenesen egy olyan szituációba érkezzünk, amire érvényes z. A tranzitivitást a következők írják le:

$$következmény(x, y, z), következmény(z, u, v) \rightarrow következmény(x, prog(y, u), v) \quad (16)$$

Itt a *prog(u, v)* egy program, ami először *u*-t hajtja végre, majd *v*-t. (Az *u* akció és az egylépéses *prog(u)* program azonosítása nyilvánvalóan szükséges, de a részletekre most nem térünk ki.)

Az utolsó előfeltétel miatt hajtódik végre a a cselekvés:

$$x, következmény(x, prog(y, z), w), akar(w) \rightarrow csinál(y) \quad (17)$$

Az *advice taker* mechanizmusa az alábbi következtetéseket vonja többé-kevésbé ebben a sorrendben a probléma megoldásához:

1. *ottvan(én, asztalom) → tud(menni(asztalom, autó, gyalog)*
2. *ottvan(én, autó) → tud(menni(otthon, reptér, autóval)*
3. *csinált(menni(asztal, autó, gyalog)) → ottvan(én, autó)*
4. *csi*nált(menni(otthon, reptér, autóval)) → ottvan(én, reptér)
5. *következmény(ottvan(én, asztalom), menni(asztalom, autó, gyalog), ottvan(én, autó))*
6. *következmény(ottvan(én, autó), menni(otthon, reptér, autóval), ottvan(én, reptér))*
7. *következmény(ottvan(én, asztalom), prog(menni(asztalom, autó, gyalog), menni(otthon, reptér, autóval))*
   → ottvan(én, reptér)
8. *csinál(megy(asztalom, autó, gyalog))*

Az utolsó előfeltétel indítja el magát a cselekvést.

A fenti előfeltételek felvetnek két fontos kérdést a levezetéssel kapcsolatban: Az első, hogy hogyan gyűjtjük össze a 17 előfeltételt, a második pedig hogy a levezetés hogy fog működésbe lépni, ha már összegyűjtöttük őket. Ezekre a kérdésekre nem adhatunk konkrét választ ezen dokumentumban, de ezek nyilvánvalóan nem teljesen elkülöníthető problémák, mivel néhány levezetés már elkészülhetett volna mielőtt az előfeltételeket összegyűjtöttük volna. Először is tisztázzuk, hogy hogyan gyűjtöttük össze a 17 előfeltételt!

Mindenekelőtt kijelenthetjük, hogy a 14. előfeltételt „*akar(ottvan(én, reptér))*”, ami célt állítja be, és az 1. előfeltételt „*ottvan(én, asztalom)*”, amit egy olyan rutintól kell megkapnunk, ami azt a kérdést válaszolja meg, hogy „hol vagyok?” kivéve minden előfeltételről feltételezhetjük, hogy már a gép memóriájában van, mert az emberhez hasonló módon felderíti saját környezetét. Egyik sem probléma-specifikus és jelenlétük a memóriában nem sejteti előre ezt a konkrét problémát, vagy olyan problémák egy osztályát, amelyek szűkebbek ennél és amelyektől elvárható, hogy ember előzőleg már megoldotta őket. Elvárhatjuk ezt a követelményt, ha azt elmondhatjuk, hogy az *advice taker* rendelkezik „józan ésszel”.

Másrészről viszont, mivel feltételezzük, hogy az előfeltételek már a memóriában vannak, még mindig meg kell határoznunk, hogy hogyan tudjuk ezeket alkotóelemeikből épített listákra szétbontani, amikre a következtető rutint már lehet alkalmazni. Például mi azt várjuk az *advice taker*-től, hogy a következőképpen hajtsa ezt végre: először vesszük az „*akar(ottvan(én, reptér))*” mondatot, amit egy bizonyos, főlistának nevezett L listára bontunk szét. A program egy megfigyelő rutinnal kezdődik, ami a főlistát kezdi el tanulmányozni, és bizonyos állításokat helyez el annak tartalmáról egy a „főlista megfigyelése” nevű listában. Még nem specifikáljuk ennek a listának a lehetséges kinézetét, de azt elmondhatjuk, hogy ebben az esetben meg fogja figyelni, hogy „L egyetlen állítása akar(u(x)) formájú”. A „következtető és végrehajtó” rutint a „főlista megfigyelései” és a „várakozó parancsok” lista kombinációjára fogjuk alkalmazni. Ez a lista többnyire kicsi, és soha nem változik, vagy legalábbis csak akkor, ha fontos változások következnek be az *advice taker* életciklusában. A „várakozó parancsok” lista tartalma még nincsen kidolgozva, de ami amit mindenképpen ki kell következtetni, az néhány állítás tulajdonság listája. Tulajdonképpen a program először megnézi az „*akar(ottvan(én, reptér))*” mondatot és megpróbálja kimásolni az állításokat egy tulajdonság listába. Tételezzük fel, hogy ez nem sikerül neki, mert az „*akar(ottvan(én, reptér))*” nem egy objektum, így nincs tulajdonság listája sem. (Itt azt várhatnánk, hogy a

reptérre menni akarás problémája már korábban felmerült, így az „*akar(ottvan(én, reptér))*” objektum lesz, de ez attól függ, hogy voltak-e olyan rutinok már korábban, amik lehetővé teszik a korábbi tapasztalatok általánosításának felhasználását.)

Ezután az általánosítást elősegítendő agép azt fogja megnézni, hogy be volt-e már valami sorolva az „*akar(ottvan(én, x))*” formába, ami a valahová menni akarás általános problémája. Elvárhatjuk, hogy a (6) (ill. (7) és (8)), (9), (10), (13) előfeltételek be vannak sorolva. Léteznie kell még továbbá az

akar(ottvan(én, x)) → csinál(megfigyel(hol vagyok én?))

formulának, amit az (1)-es előfeltétel ad. Kell továbbá lennie egy utalásnak az egyel magasabb absztrakciós szinten lévő cél állításra, ami azt okozhatja, hogy az „*akar(x)*” tulajdonság listáját is megnézzük. Ez vezet a (15), (16) és (17) állításokhoz.

Nem fogjuk tovább követni a megoldás menetét, viszont megjegyezzük, hogy az „*akar(ottvan(én, x))*” tulajdonság listája egy szabály kéne hogy legyen, ami az „*ottvan(én, y)*” és az „*akar(én, x)*” előfeltételekkel kellene, hogy kezdődjön és tartalmaznia kéne egy következtetést a „*megy(y, x, z)*” tulajdonság listájának keresésére. Ez feltételezhetően meghiúsulna, majd olyan levezetések következnének, ahol egy olyan y-t keresnénk, amire igaz, hogy „*ottvan(én, y)*” és „*ottvan(reptér, y)*”. Ez megtörténhetne a kiinduló- és a célpont tulajdonság listájának átnézésével és feldolgozásával. Ezek után a (10)-es előfeltétel adódna, aminek egyik előfeltétele az „ottvan(én, autó)”. A fentiek megismétlése megtalálná a (9)-es előfeltételt is, ami teljessé tenné az előfeltételek halmazát, mivel a többi „ottvan” előfeltételeket már korábbi keresések melléktermékeként megtaláltuk volna.

Reméljük, hogy gyakorlatban látva a tulajdonság listákban említett levezetési szabályokat kézenfekvővé tettük az olvasó számára a dolgokat. Meg kell, hogy jegyezzük, hogy magasabb absztrakciós szinten sok állítás reflex-szerű. Gyaníthatjuk, hogy az ember esetében a tudatos és tudatalatti gondolatok közti elkülönülés egy olyan határvonal, ami az itt említett reflex-szerű levezetések (amiket nem kell megmagyarázni, csak az eredményt kell végrehajtani) és a többi levezetés (melyek előfeltételeket és következtetéseket eredményeznek) közt lévő határvonalhoz hasonlítható.

Felhasznált irodalom

- John McCarthy: Programs with common sense (1959)

  http://www-formal.stanford.edu/jmc/mcc59/mcc59.html

- John McCarthy (computer scientist) (Wikipedia)

  http://en.wikipedia.org/wiki/John_McCarthy_%28computer_scientist%29

- John McCarthy (Agent Portal)

  http://www.agent.ai/?folderID=151&articleID=508&ctag=&iid=