

A mesterséges intelligencia alapjai

Mihálydeák Tamás

Számítógéptudományi Tanszék,
Informatikai Kar
Debreceni Egyetem
e-mail: mihalydeak.tamas@inf.unideb.hu
<https://arato.inf.unideb.hu/mihalydeak.tamas/>

2017. március 26.

Bevezetés

Mesterséges (?) intelligencia (!) [Artificial intelligence (AI)]

A mesterséges intelligencia (MI) néhány értelmezése

- 1 Emberi módon gondolkodó rendszerek
- 2 Emberi módon cselekvő rendszerek
- 3 Racionálisan gondolkodó rendszerek
- 4 Racionálisan cselekvő rendszerek

1. Emberi módon gondolkodni

- Meg kellene határozni, hogy az emberek hogyan gondolkodnak:
 - önelemzés
 - pszichológiai kísérletek
- Az MI számítógépes modelljeit és a pszichológia kísérleti technikáit a kognitív tudomány kapcsolja össze.

2. Emberi módon cselekedni

- Alan Turing (1912–1954)
- Turing teszt (1950): Valami intelligens, ha megkülönböztethetetlen egy vitathatatlanul intelligens entitástól.
- A számítógép akkor állja ki a próbát, ha az emberi kérdező néhány írásos kérdés feltétele után nem képes eldönteni, hogy az írásos válaszok egy embertől vagy egy géptől érkeznek-e.
 - természetes nyelv feldolgozása
 - tudásreprezentáció
 - automatizált következtetés
 - gépi tanulás
 - teljes Turing teszt:
 - gépi látás
 - robotika

3. Racionálisan gondolkodó rendszerek

- Arisztotelész (Kr.e. 384-322) logikája: a helyes következtetés törvényszerűségeinek első rendszerbe foglalása
- XIX.–XX. század: a logika modern elméleteinek létrejötte.
- Az MI logicista felfogása:
 - A hangsúly teljes egészében a helyes következtetéseken van.

4. Racionálisan cselekvő rendszerek: a racionális ágens

- Szemléletesen egy ágens nem más, mint valami, ami cselekszik.
- Egy racionális ágens a legjobb (várható) kimenetel érdekében cselekszik.
- A helyes következtetés része a racionális cselekvésnek: vannak olyan racionális cselekvések, amelyeknél a következtetésnek nyoma sincs (pl. reflexszerű cselekvések).
- Az MI mint a racionális ágensek tervezése:
 - általánosabb a gondolkodás törvényére koncentráló megközelítésnél;
 - tudományosan kezelhető: a racionalitás mértéke definiálható és általános.
- A továbbiakban a racionális ágensek általános elveire és a létrehozásukhoz szükséges komponensekre koncentrálunk.
 - Repülnek-e a fecskék?
 - Repülnek-e a repülőgépek?

Cihan H. Dagli

- "A gépi intelligencia emulálja, vagy lemásolja az emberi ingerfeldolgozást (érzéketfeldolgozást) és a döntéshozó képességet számítógépekkel. Az intelligens rendszereknek autonóm tanulási képességekkel kell bírniuk és alkalmazkodniuk kell tudni bizonytalan, vagy részlegesen ismert környezetekhez."

Aaron Sloman

- "A számítógéptudomány egy alkalmazott részterülete. A mesterséges intelligencia egy nagyon általános kutatási irány, mely az intelligencia természetének kiismerésére és megértésére, valamint a megértéséhez és lemásolásához szükséges alapelvek és mechanizmusok feltárására irányul."

Yoshiaki Shirai - Jun-ichi Tsujii

- "A mesterséges intelligencia kutatásának célja az, hogy a számítógépeket alkalmassá tegyük az emberi intelligenciával megoldható feladatok ellátására."

Peter Jackson

- "A mesterséges intelligencia a számítógéptudomány azon részterülete, amely az ember olyan kognitív (megismerő) képességeit emuláló számítógépi programok tervezésével és alkalmazásával foglalkozik, mint a problémamegoldás, vizuális érzékelés és a természetes nyelvek megértése."

Filozófia (Kr.e. V. századtól)

- Általános kérdések
 - Lehet-e formális szabályok révén igaz konklúzióhoz jutni?
 - Hogyan emelkedik ki a mentális elme a fizikai agyból?
 - Honnan ered a tudás, és miképpen vezet cselekvéshez?
- Arisztotelész (Kr.e. IV.): helyes következtetések elmélete
- Ramon Lull (XIV.): a hasznos következtetést egy gépezetre lehetne bízni
- Leonardo da Vinci (XV.–XVI.): mechanikus kalkulátor megtervezése (ma már tudjuk, hogy működőképes volt!)
- Thomas Hobbes (XVII.): a következtetés olyan, mint egy numerikus számítás
- Schickard, Pascal (XVII.): az első ismert számítógép;
- Leibniz, Descartes (XVII.–XVIII.)
- Boole, Frege (XIX.), Russell, Wittgenstein, Carnap (XX.)

Matematika (IX. századtól)

- Melyek a helyes következtetések formális szabályai?
- Mi az, ami kiszámítható?
- Hogyan vagyunk képesek a bizonytalan információ alapján következtetéseket levonni?

Gazdaságtan(XVIII. századtól)

- Hogyan kell döntenünk, hogy a hasznunk maximális legyen?
- Mit kell tennünk, ha mások esetleg nem segítőkészek?
- Hogyan kell döntenünk, ha a haszonhoz csak a távoli jövőben jutunk el?

Neurális tudományok (XIX. századtól)

- Hogyan dolgozza fel az információt az agy?

Pszichológia (XIX. századtól)

- Hogyan gondolkoznak és cselekszenek az emberek és az állatok?

Számítógépes tudományok (XX. századtól)

- Hogyan lehet hatékony számítógépet építeni?

Írányításelmélet és kibernetika (XX. századtól)

- Hogyan működhet egy mesterségesen létrehozott eszköz a saját irányítása mellett?

Nyelvészet (XX. századtól)

- Mi a nyelv és a gondolat kapcsolata?

Érlelődés (1943–1955)

- McCulloch, Pitts: mesterséges neuron modell (1943)
 - Háttér: agyi neuronok működése, állításkalukulus (Russell, Whitehed), Turing számításelmélete
- Minsky, Edmonds: 40 neuronból álló hálózatot szimuláló első neurális számítógép (1951., SNARC, 3000 elektroncső)
- Turing: Comptuing Machinery and Intelligence: egy teljes elképzelés az MI-ről (Turing teszt, gépi tanulás, genetikusan algoritmusok, megerősített tanulás)

Az MI megszületése (1956)

- 1956: Dartmouth: a mesterséges intelligencia megszületése
- Itt történt a névadás is. (Számítási racionalitás: lehet, hogy jobb név lett volna.)
- Az MI kezdetek óta megkísérli bizonyos emberi tulajdonságok duplikálását: kreativitás, önfejlesztés, nyelv használata.
- Az MI az egyetlen olyan terület, ahol bonyolult, változó környezetben autonóm módon működő gépek építése a cél.

Korai lelkesedés időszaka (1952–1969)

- Ez a "Nézze uram, biz' Isten magától megy" időszaka
- Általános problémamegoldó: General problem solver (GPS)
- Gelernter (1959): Geometry Theorem Prover.
- McCarthy (1958):
 - LISP: elsődleges MI programozási nyelv lett
 - Advice Taker: az első teljes MI rendszer: a tudásreprezentáció és a következtetés leglényegesebb elveinek megtestesítése

Hullámvölgy (1966-1973)

- A korai rendszerek csődöt mondtak, ha szélesebb körben vagy nehezebb problémákr akarták őket bevetni.
- Ok: a korai programok magáról a problémáról nagyon kevés tudást tartalmaztak: csupán egyszerű szintaktikai manipulálással értek el sikereket.
 - A szellem készséges, de a test gyenge. Kétszeres fordítás eredménye: A vodka jó, de a hús romlott.
- Az a tény, hogy egy program egy megoldás megtalálására elvben alkalmas, nem jelenti azt, hogy a program bármi olyan mechanizmust is tartalmaz, amely a megoldás gyakorlati megvalósításához szükséges.
- Kombinatorikus robbanás.
- A használt struktúrák fundamentális korlátai: amit reprezentálni tud egy program, azt megtanulhatja, csak keveset tud reprezentálni.

Tudásalapú rendszerek (1969–1979)

- Heurisztikus programozási projekt

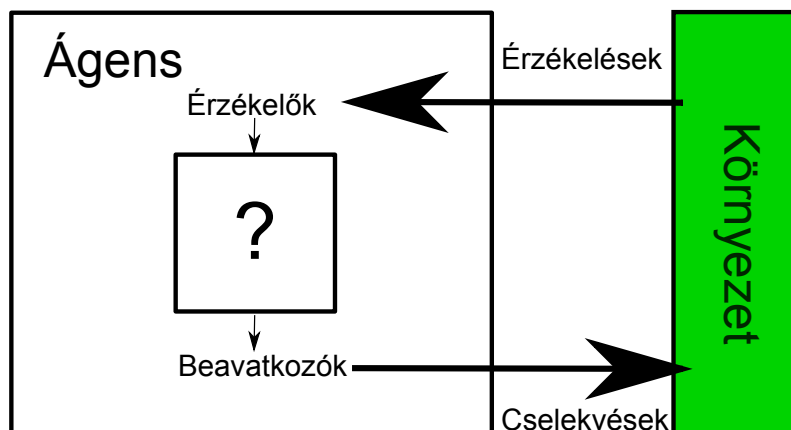
Az MI iparrá válik (1980-tól)

- 1980: néhány millió dolláros forgalom; 1988 2 milliárd dolláros forgalom

Az MI tudománnyá válik (1987-től)

- Az elegánsak győzelme a szakadtak felett

Intelligens ágensek kialakulása (1995-től)



- A mesterséges intelligencia: a környezetüket érzékelő és cselekvő ágensek tanulmányozása
- Ágens: érzékeléseket cselekvésre leképező függvényt valósít meg.

- Az ágens érzékelői segítségével érzékeli a környezetét, és beavatkozási segítségével megváltoztatja azt.
- Az ágens érzékelésének a fogalma:
 - egy tetszőleges pillanatban egy ágens érzékelő bemeneteinek összességét írja le;
- Egy ágens érzékelési sorozata az ágens érzékeléseinek teljes története (minden, amit az ágens valaha érzékelt).
- Általánosságban: egy adott pillanatban egy ágens cselekvése az addig megfigyelt teljes érzékelési sorozattól függhet:
 - Az ágens viselkedését egy ágensfüggvény írja le: az ágensfüggvény érzékelési sorozatot cselekvésre leképező függvény.
 - A mesterséges ágens belsejében az ágensfüggvényt egy ágensprogram valósítja meg.
 - ágensfüggvény: absztrakt matematikai leírás
 - ágensprogram: egy konkrét implementáció, amely az ágens architektúráján működik.
 - Pl.: porszívóvilág

A racionalitás koncepciója

- Racionális ágens: a helyesen cselekvő ágens
 - Helyes cselekedet: az a cselekedet, amely az ágenszt legsikeresebbé teszi.
- Teljesítménymérték: az ágens sikerességének mértéke (az ágens tervezőjének kell meghatároznia)
- A teljesítménymértéket aszerint kell megállapítani, hogy mit akarunk elérni a környezetben, és nem aszerint, hogy miképp kellene az ágensnek viselkednie.

Mitől függ egy ágens racionalitása egy adott pillanatban?

- a siker fokát mérő teljesítménymértéktől;
- az ágens a környezetre vonatkozó tudásától;
- az ágens által végrehajtható cselekvésektől;
- az ágens érzékelési sorozatától (az adott pillanattig).

A racionális ágens értelmezése

- Az ideális racionális ágens minden egyes észlelési sorozathoz a benne található tények és a beépített tudása alapján minden elvárható dolgot megtesz a teljesítménymérték maximalizálásáért.
- A racionalitás nem azonos a mindentudással.
 - Egy mindentudó ágens tudja cselekedetei valódi kimenetelét, és ennek megfelelően cselekedhet.
 - A racionalitás az elvárt teljesítmény maximalizálja.
 - A tökéletesség a tényleges teljesítmény maximalizálja.

A racionalitás néhány alkotó eleme

- Információgyűjtés: a hasznos információk beszerzése, felfedezés.
- Tanulás: az ágens tapasztalata alapján az előzetes tudása módosulhat, átértékelődhet.

Az ágensek autonómiája

- Nem autonóm ágens: nem épít saját megfigyeléseire csak a beépített tudásra.
- Egy racionális ágensnek autonómnak kell lennie (mindent, amit megtanulhat, meg kell tanulnia ahhoz, hogy hibás előzetes tudását kompenzálja).

TKBÉ

- Teljesítmény
- Környezet
- Beavatkozók
- Érzékelők

A környezetek fajtái 1.

- teljesen megfigyelhető — részlegesen megfigyelhető
- determinisztikus — sztochasztikus
 - Determinisztikus a környezet, ha a környezet következő állapotát jelenlegi állapota és az ágens által végrehajtott cselekvés teljesen meghatározza
 - Sztochasztikus: egyébként
- Stratégiai a környezet, ha a környezet más ágensek cselekvéseit leszámítva determinisztikus (a sztochasztikus jelleget csak más ágensek viselkedése jelenti)

A környezetek fajtái 2.

- Epizódszerű — sorozatszerű
 - Epizódszerű: az ágens tapasztalata elemi epizódokra bontható, minden egyes epizód az ágens észleléseiből és egy cselekvésből áll. A következő epizód nem függ az előzőekben végrehajtott cselekvésektől (pl.: alkatrészeket tesztelő ágens)
 - Sorozatszerű: az aktuális döntés befolyásolhat minden továbbit (pl.: sakk, vezetés)
- Statikus — dinamikus
 - Ha a környezet megváltozhat, amíg az ágens "gondolkodik", akkor a környezet dinamikus (pl.: gépkocsi vezetés)
 - Statikus: egyébként (pl. keresztreljtvény)
 - Szemidinamikus környezet: ha környezet időben nem változik, de az ágens teljesítménymértéke igen (sakk, órával).

A környezetek fajtái 3.

- Diszkrét — folytonos
 - A felosztás alkalmazható a következőkre:
 - a környezet állapotára (pl.: diszkrét: sakk)
 - az időkezelés módjára
 - az ágens észleléseire
 - az ágens cselekvéseire (pl.: diszkrét: sakk)
- Egyágenses — többágenses
 - versengő — kooperatív
- Legnehezebb: a részlegesen megfigyelhető, sztochasztikus, sorozatszerű, dinamikus, folytonos, többágenses eset.

A mesterséges intelligencia feladata

- Az ágensprogram megtervezése: egy függvényé, amely megvalósítja a az észlelések és a cselekvések közötti leképezést.

ágens = architektúra + program

Ágensprogramok

- az ágensprogramok váza: bemenetként fogadják az aktuális észleléseket (a szenzoroktól), és visszaküldenek egy cselekvést a beavatkozókhoz.
- az ágensprogram az aktuális észlelést veszi bemenetként, az ágensfüggvény a teljes észlelési történetet fogadja.
- az ágensprogram csak az aktuális észlelést tudja fogadni, mert az érkezik a környezetből.
- Az ágensprogramokat pszeudokóddal fogjuk leírni.

A pszeudokód nyelve

- Statikus változók:
 - kulcsszó: `static`;
 - értékét a függvény első hívásánál kapja meg, és megtartja a függvény minden további hívásánál;
 - hasonlít a globális változóra, de értéke csak a függvényen belül hozzáférhető;
 - a statikus változókat kezelő programok objektumként implementálhatók objektumorientált nyelvekben
- függvények mint értékek:
 - függvények és eljárások: nagybetűs nevek (FN);
 - változók: dőlt kisbetűs nevek (x);
 - függvényhívás: $\text{FN}(x)$;
 - megengedjük, hogy egy változó függvény típusú értéket is felvehessen;
 - a tömbök 1-től kezdődnek;
 - a beljebb szedett bekezdést hurok, vagy feltételes kifejezés hatáskörének megadására használjuk.

Egy példa

```

1: function TABLAZAT-VEZERLESU-AGENS(eszleles)
2:   static eszlelesek                                ▷ egy sorozat, kezdetben üres
3:   static tablazat                                ▷ az észlelési sorozat által indexelt táblázat,
   kezdetben teljesen feltöltött
4:   csatold az eszleles-t az eszlelesek végére
5:   cselekves ← KIKERESES(eszlelesek, tablazat)
6:   return cselekves
7: end function

```

- Az MI alapvető kihívása: hogyan írjunk olyan programot, amely nagyszámú táblázatbejegyzés helyett kisméretű programmal produkál racionális viselkedést.

Egyszerű reflexszerű ágensek

- Az aktuális észlelés alapján választják ki a cselekvéseket.

```
1: function REFLEXSZERU-PORSZIVO-AGENS(helyszin, allapot)
2:   if allapot = Piszkos then
3:     return Felszívás
4:   else if helyszin = A then
5:     return Jobbra
6:   else if helyszin = B then
7:     return Balra
8:   end if
9: end function
```

Feltétel–cselekvés szabályok

- Az észlelések feldolgozása alapján létrejött cselekvéseket lehet ezekkel a szabályokkal vezérelni.

```
1: function EGYSZERU-REFLEXSZERU-AGENS(eszleles)
2:   static szabalyok                                ▷ feltétel–cselekvés szabályok halmaza
3:   allapot ← BEMENT-FELDOLGOZAS(eszleles)
4:   szabaly ← SZABALY-ILLESZTES(allapot, szabalyok)
5:   cselekves ← SZABALY-CSELEKVES(szabaly)
6:   return cselekves
7: end function
```

- Ez az ágens csak akkor fog működni, ha a helyes döntés kizárólag az aktuális észlelés alapján meghozható - azaz csak akkor, ha a környezet teljesen megfigyelhető.

Modellalapú reflexszerű ágensek

- Részleges megfigyelhetőség kezelése: az ágens nyomon követi a világ jelenleg nem megfigyelhető részét:
 - az ágensnek nyilván kell tartania valamiféle belső állapotot, amely az észlelési történeten alapul, és így a jelenlegi állapot nem megfigyelt aspektusainak legalább egy részét tükrözi.

```

1: function REFLEXSZERU-AGENS-ALLAPOT(eszleles)
2:   static allapot                                ▷ a világ jelenlegi állapotának leírása
3:   static szabalyok                                ▷ feltétel–cselekvés szabályok halmaza
4:   static cselekves                                ▷ a legutolsó cselekvés, kezdetben semmi
5:   allapot ← ALLAPOT-FRISSITES(allapot, cselekves, eszleles)
6:   szabaly ← SZABALY-ILLESZTES(allapot, szabalyok)
7:   cselekves ← SZABALY-CSELEKVES(szabaly)
8:   return cselekves
9: end function

```

Célorientált ágensek

- Alapvetően különbözik a feltétel–cselekvés szabályoktól:
 - magában foglalja a jövő figyelembevételét: Mi fog történni, ha ezt és ezt teszem?
- Sokkal rugalmasabb mivel a döntéseit alátámasztó tudás explicit módon megjelenik.

Hasznosságorientált ágensek

Tanuló ágensek

Problémamegoldás kereséssel

Problémamegoldó ágens

- A célorientált ágensek egyik típusa
- Olyan cselekvéssorozatot keresnek, amelyek a kívánt állapotokba vezetnek.
- Lépések:
 - A probléma pontos megfogalmazása
 - A probléma megoldását felépítő alkotóelemek megadása
 - Általános rendeltetésű keresési algoritmusok megadása
 - Nem informált algoritmusok: a probléma definícióján kívül más információval a problémáról nem rendelkeznek.

Problémamegoldó ágensek

- Célmegfogalmazás:
 - a pillanatnyi helyzeten és az ágens hasznosságmértékén alapul;
 - cél reprezentációja: a világ állapotainak azon halmaza, amelyben a cél teljesül (ezeket az állapotokat nevezzük célállapotoknak);
 - az ágens feladata a cselekvések egy olyan sorozatának a megkeresése, amely amely eljuttatja őt egy célállapotba;
- Probléma–megfogalmazás: adott cél esetén a figyelembe veendő állapotok meghatározása

Keresés

- Cselekvéssorozat előállítási folyamata:
 - A keresési algoritmus bemenete egy probléma, kimenete pedig egy cselekvéssorozat formájában előálló megoldás.
 - A megoldás megtalálálása után az abban foglalt cselekvéseket végre lehet hajtani: ez a végrehajtási fázis.
 - Ágenstervezési séma: "fogalmazd meg, keresd meg, hajtsd végre"

```

1: function EGYSZERU-PROBLEMAMEGOLDO-AGENS(erzekeles)
2:   inputs erzekeles                                ▷ egy érzékelés
3:   static sorozat                                ▷ egy cselekvéssorozat, kezdetben üres
4:   static allapot                                ▷ a világ pillanatnyi állapotának valamilyen leírása
5:   static cel                                    ▷ egy cél, kezdetben üres
6:   static problema                                ▷ egy probléma megfogalmazása
7:   allapot ← ALLAPOT-FRISSITES(allapot, erzekeles)
8:   if sorozat = ures then
9:     cel ← CEL-MF(allapot)
10:    problema ← PROBLEMA-MF(allapot, cel)
11:    sorozat ← KERESSES(problema)
12:  end if
13:  while sorozat ≠ ures do
14:    cselekves ← AJANLAS(sorozat)
15:    sorozat ← MARADEK(sorozat)
16:  return cselekves
17:  end while
18: end function

```

Az egyszerű ágens tulajdonságai

- a környezet statikus
- a kezdeti állapot ismert (ennek ismerete akkor a legkönnyebb, ha a környezet megfigyelhető)
- alternatív cselekvések számontartása: a környezet diszkrét
- a környezet determinisztikus
 - nyílt hurkú (open loop) rendszer: az érzékelések figyelmen kívül hagyása az ágens és környezete közötti hurkot felbontja.

Jól definiált problémák

- Egy problémát a következő komponensekkel lehet definiálni:
 - Kiinduló állapot (initial state)
 - Cselekvések (actions): egy adott állapotban az ágens számára lehetséges cselekvések leírása
 - A kezdeti állapot és az állapot-átmenet függvény implicit módon definiálja a probléma állapotterét (state space)
 - Az állapotter egy gráfot alkot: csomópontjai az állapotok, a csomópontok közötti élek a cselekvések.
 - Célteszt: meghatározza, hogy egy adott állapot célállapot-e.
 - állapotok explicit halmaza
 - absztrakt tulajdonság
 - Útköltség függvény: az ágens a saját hatékonysági mértékének megfelelő költségfüggvényt használja.

Cselekvések reprezentációja

- állapotátmenet-függvény (successor function): egy adott x állapot esetén az *ALLAPOTATMENET* – $FV(x)$ visszaadja a rendezett $\langle cselekves, utodallapot \rangle$ párok halmazát, ahol minden cselekvés egyike az x állapotban legális cselekvéseknek, és minden utódállapotot egy cselekvésnek az x állapotra való alkalmazásával nyerünk.
- más megfogalmazás: operátorok egy halmaza, amelyet egy állapotra alkalmazva lehet utódállapotokat generálni.

A problémák megfogalmazása

- Az absztrakció szükségessége
 - A nem releváns részlete kihagyása a probléma megfogalmazása során.
 - Lustasági kritérium: annyit és csak annyit emeljünk be a modellbe, amennyi a probléma megoldásához szükséges.
- Az állapotleírás során végzett absztrakció
- A cselekvések leírása során végzett absztrakció
- Az absztrakció érvényes, ha az absztrakt megoldást megoldássá fejthetjük ki egy részletesebb világban is.
- Az absztrakció hasznos, ha a megoldásbeli cselekvések végrehajtása az eredeti problémánál egyszerűbb.

Néhány példa

- Játékproblémák
 - 8-királynő probléma (inkrementális — totális)
- útkeresési problémák
- körutazási problémák
- utazó ügynök probléma
- automatikus összeszerelés
- interneten kereső szoftverrobotok

Keresési fák használata

- A keresési fa az állapottérből származtatható: a kezdeti állapot és az állapotátmenet függvény generálja.
- Általános esetben nem fáról, hanem gráfról beszélhetünk (pl. ha egy állapotot több úton is elérhetünk)
- A keresési fa gyökere: a kezdeti állapotnak megfeleltetett csomópont.
- Lépések
 - A kezdeti állapot célállapot-e.
 - Az állapotátmenet-függvénnyel az állapotok egy új halmazát generáljuk
 - A kifejtendő állapot kiválasztását a keresési stratégia határozza meg.

Állapottér — keresési fa

- Míg az állapottér véges, addig a keresési fa lehet végtelen is!
- A keresési fa csomópontjai: öt komponensből álló adatszerkezet:
 - 1 Állapot: az állapottérnek a csomópont-hoz tartozó állapota
 - 2 Szülő-csomópont: a keresési fa azon csomópontja, amely a kérdéses csomópontot generálta
 - 3 Cselekvés: a csomópont szülő csomópontjára alkalmazott cselekvés
 - 4 Út-költség: a kezdeti állapottól a kérdéses csomópontig vezető út költsége
 - 5 Mélység: a kezdeti állapottól vezető út mélysége
- Perem: kifejtendő csomópontok (ezeket is nyilván kell tartani), a fa levélelemei
 - Melyik a következő kifejtendő: várakozási sor.

Informális fakeselési algoritmus

```

1: function FA-KERESES(problema, strategia)
2:   a probléma kiinduló állapotából kiindulva inicializáld a keresési fát
3:   loop
4:     if nincs kifejtendő csomópont then
5:       return kudarc
6:     end if
7:     a stratégiának megfelelően válasz ki egy levélcsomópontot
8:     if a csomópont célállapotot tartalmaz then
9:       return a hozzá tartozó megoldás
10:    else
11:      fejtsd ki a csomópontot és az eredményül kapott
      csomópontokat add a keresési fához
12:    end if
13:  end loop
14: end function

```

Hatékonyság kérdései

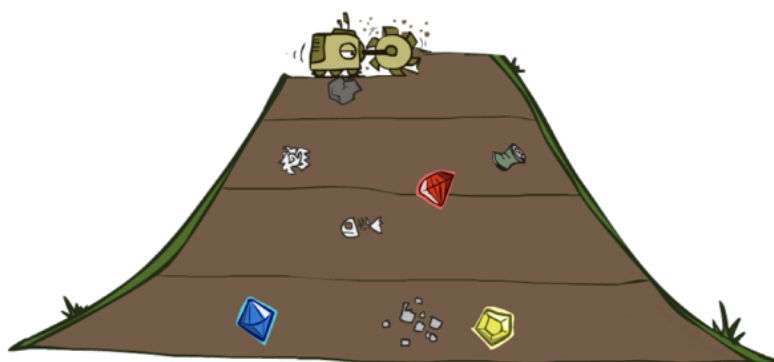
- Teljesség: az algoritmus garantáltan megtalál egy megoldást, amennyiben létezik.
 - Optimalitás: a stratégia megtalálja az optimális megoldást.
 - Időigény (time complexity): mennyi ideig tart a megoldás megtalálása.
 - Tárigény (space complexity): a keresés elvégzéséhez mennyi memóriára van szükség.
-
- A mesterséges intelligenciában a komplexitást kifejezői:
 - elágazási tényező (b)
 - a legsekélyebb célállapot mélysége (d)
 - az állapottérben található utak maximális hossza (m)
 - idő: a keresés közben generált csomópontok számával mérik
 - tár: a memóriában maximálisan tárolt csomópontok számával mérik
 - útköltség; keresési költség; összköltség

Nem informált (vak) keresés

- A stratégiáknak nincs semmilyen információjuk az állapotokról a probléma definíciójában megadott információkon kívül.
- Két dolgot tehetnek::
 - generálhatják a következő állapotokat;
 - meg tudják különböztetni a célállapotot a nem célállapottól

Szélességi keresés(breadth-first-search)

Breadth-First Search



Szélességi keresés

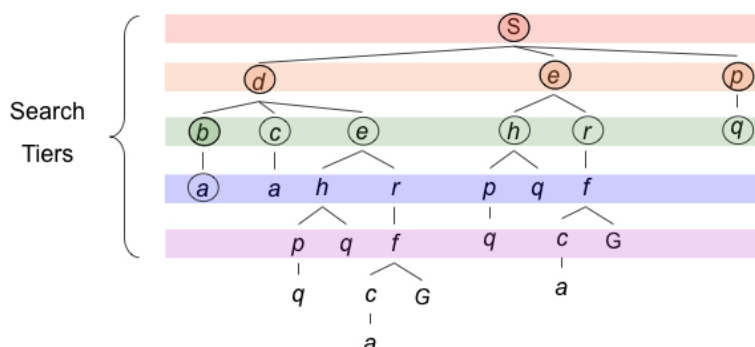
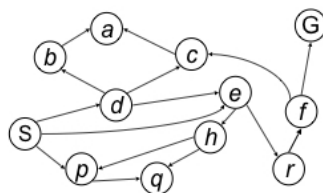
- Gyökércsomópont kifejtése
- Az összes gyökércsomópontból generált csomópont kifejtése, stb.
- A keresési stratégia minden adott mélységű csomópontot hamarabb fejt ki, mielőtt bármelyik egy szinttel lejjebbi csomópontot kifejtené.
- Megvalósítása:
 - FA-KERESES algoritmussal, egy olyan üres peremmel, amely először-be-először-ki (first-in-first-out, FIFO) sor
 - a korábban generált csomópontokat az algoritmus korábban fejt ki.
- a keresés teljes
- a legsekélyebb célcsomópont d mélységben fekszik, és a b elágazási tényező véges, akkor a szélességi keresés eljut hozzá (az összes nála sekélyebb csomópontot kifejtve)
- a legsekélyebb célcsomópont nem feltétlenül optimális
- A szélességi keresés optimális, ha az útköltség a csomópont mélységének nem csökkenő függvénye.

Szélességi keresés, egy példa

Breadth-First Search

Strategy: expand a shallowest node first

Implementation:
Fringe is a FIFO queue



A szélességi kereséssel problémái

- Ha a probléma megoldása d mélységben található, és minden csomópont b számú csomópontot generál, akkor a legrosszabb esetben a kifejtett csomópontok száma:

$$b + b^2 + \dots + b^d + (b^{d+1} - b) = \mathcal{O}(b^{d+1})$$
 - Ha f és g egyváltozós valós függvények, akkor $f(x) = \mathcal{O}(g(x))$ [kiolvasás: $f(x)$ egyenlő nagy ordó $g(x)$] akkor és csak akkor, ha léteznek olyan M és x_0 pozitív valós számok, hogy minden $x > x_0$ esetén $f(x) \leq Mg(x)$.
 - Intuitív jelentés: elég nagy x értékek esetén az f függvény nem nő gyorsabb a g függvényénél.
- $b = 10$, akkor $d = 10$ esetén a csomópontok maximális száma 10^{11} , időigény 129 nap, tárigény 101 Tbájt (10000 csomópont/perc; 1000 bájt/csomópont)
- $d = 12$, akkor az időigény 35 év!

Egyenletes költségű keresés (Uniform cost search)

Uniform Cost Search



Egyenletes költségű keresés/1

- A szélességi keresés a legkisebb költségű, azaz optimális megoldást adja vissza, ha minden lépés költsége azonos.
- Az egyenletes költségű keresés: tetszőleges lépésköltség esetén az optimális megoldást adja vissza.
- Mindig a legkisebb útköltségű csomópontot fejt ki először (nem pedig a legkisebb mélységű csomópontot).
- Ha a lépésköltségek azonosak, akkor a szélességi keresés is egyenletes költségű keresés.
- Az egyenletes költségű keresés nem foglalkozik az út hosszával, csak a költségével.
- Végtelen hurokba kerülhet: ha egy csomópont kifejtése zérus költségű cselekvéshez vezet, és az adott állapothoz való visszatérést eredményez.

Egyenletes költségű keresés/2

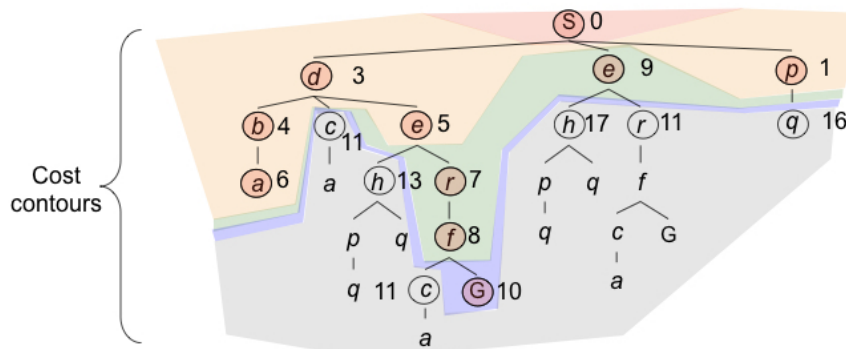
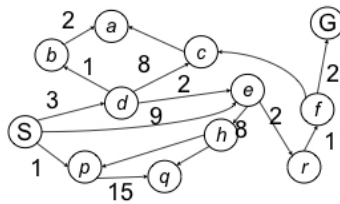
- Teljesség: ha minden lépés költsége $\geq \epsilon$ ahol $\epsilon > 0$.
- Optimalitás: ha teljes, akkor optimális.
- Az egyenletes költségű keresést nem a mélység, hanem az útköltség vezérli.

Egyszerű költségű keresés, egy példa

Uniform Cost Search

Strategy: expand a
cheapest node first:

Fringe is a priority queue
(priority: cumulative cost)



Mélységi keresés(depth-first search)

Depth-First Search



Mélységi keresés/1

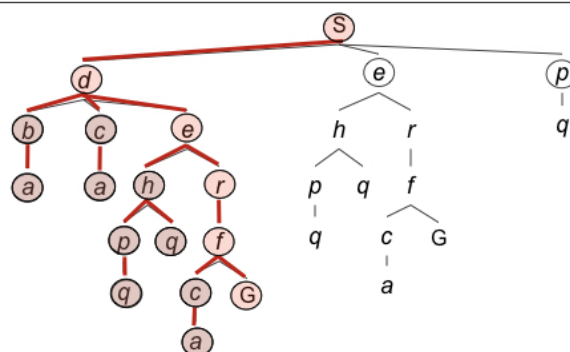
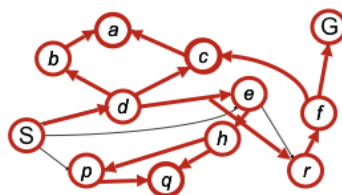
- Mindig a keresési fa aktuális peremében a legmélyebben fekvő csomópontot fejt ki.
- A keresés azonnal a fa legmélyebb pontjára jut el (a csomópontoknak már nincsenek követői).
- A legmélyebb csomópontok kifejtését követően a kikerülnek a peremből, és a keresés visszalép ahhoz következő legmélyebben fekvő csomópontához, amelynek még vannak ki nem fejtett követői.
- A stratégia implementálható egy olyan FA-KERESÉS függvénnyel, amelynek sorbaállító függvénye az utolsónak-be-elsőnek-ki (last-in-first-out, LIFO), ezt veremnek is nevezik.
- Nagyon szerény tárigényű: a gyökércsomóponttól egy levélcsomópontig terjedő utat kell tárolnia + az út minden egyes csomópontja melletti kifejtetlen csomópontokat.
- Ha egy kifejtett csomópont összes leszármazottja meg lett vizsgálva, akkor a csomópont törölhető a memóriából.

Mélységi keresés, egy példa

Depth-First Search

Strategy: expand a deepest node first

Implementation:
Fringe is a LIFO stack



Mélységi keresés/2

- Hátránya: egy rossz választással egy hosszú út mentén haladhat.
- Ha például a bal oldali részfa korlátlanul mély, és nem tartalmazza a megoldást, akkor a mélységi keresés soha nem állna meg: a mélységi keresés nem teljes.
- Előfordulhat, hogy a mélyebben fekvő megoldást adja találja meg először, azaz a mélységi keresés nem optimális.

Mélységkorlátozott keresés (depth-limited search)

- Kiküszöböli a végtelen keresési fák problémáját:
 - az utak maximális hosszára egy l korlátot ad;
 - az l mélységben levő csomópontokat úgy kezeli, mintha nem is lennének követőik.
- Megjelenik a nem-teljesség egy újabb forrása: ha $l < d$, azaz ha a legsekélyebb célcsomópont a mélységkorláton túl van.

Az ismételt állapotok elkerülése

- Ha az állapotátmenet függvények (az operátorok) reverzibilisek, akkor nem kerülhető el az ismételt állapotok megjelenése a keresési fában.
 - Pl.: útkeresési problémák
 - Ezekben az esetekben a keresési fák végtelenek.
- A megismételt állapotok egy részének kimetszésével, a keresési fát véges méretűvé vághatjuk.
- Az ismétlődő állapotokat detektálni kell: az új kifejtendő csomópontot a már kifejtett csomópontokkal hasonlítjuk össze.
 - Egyezés esetén az adott csomóponthoz az algoritmus két utat talált, valamelyiket eldobhatja.
 - Ehhez ismerni kell a történetet: az az algoritmus, amely elfelejti történetét, kénytelen azt megismételni.

Gráf keresés algoritmusai

- A FA-KERESÉS algoritmusnak egy sajátos módosítása.
 - Tartalmazzon egy zárt listának nevezett adatszerkezetet, amely minden kifejtett csomópontot tárol.
 - Ha az aktuális állapot egybeesik a zárt listán lévő állapotok egyikével, akkor eldobható (a kifejtésével nem kell foglalkozni).
 - Az egyenletes költségű és a konstans lépésköltségű szélességi keresés optimális fakeresési stratégia, és a belőle nyert gráfkeresési stratégia is optimális.

```

1: function GRAF-KERESES(problema, perem)
2:   zart lista  $\leftarrow$  egy üres halmaz
3:   perem  $\leftarrow$  BESZUR(CSOMOPONT-LETREHOZ
4:     (KIINDULO-ALLAPOT[problema]), perem)
5:   loop
6:     if URES?(perem) then
7:       return kudarc
8:     end if
9:     csomopont  $\leftarrow$  VEDD-AZ-ELSO-ELEMET(perem)
10:    if CEL-TESZT[problema](ALLAPOT[csomopont]) then
11:      return MEGOLDAS(csomopont)
12:    end if
13:    if ALLAPOT[csomopont] nem eleme a zárt listának then
14:      adjuk hozzá az ALLAPOT[csomopont]-ot a zárt listához
15:      perem  $\leftarrow$  BESZUR-MIND
16:        (KIFEJT(csomopont, problema), perem)
17:    end if
18:  end loop

```

Nem informált vs. informált keresés

- Nem informált keresési stratégiák:
 - szisztematikusan új állapotokat generálnak és összehasonlítják azokat a célállapottal;
 - sok esetben gyenge a hatékonyságuk.
- Informált keresési stratégia:
 - probléma-specifikus tudást alkalmaz

A legjobbat–először keresés (best-first search BFS)

- A FA-KERESÉS vagy a GRÁF-KERESÉS speciális esete:
 - egy csomópont kifejtésre való kiválasztása egy $f(n)$ kiértékelő függvényről függ;
 - a legkisebb értékű csomópontot választjuk kifejtésre, mert a kiértékelő függvény a céltól aló távolságot méri;
 - egy prioritási sor segítségével implementálható: olyan adatstruktúra, amely a peremet a növekvő f -értékek szerint rendez;
 - csak a kiértékelő függvény szerint legjobbnak tűnő csomópontot választja ki.
- BFS: egy keresési algoritmus család: elemeit az eltérő kiértékelő függvények különböztetik meg.
 - a kiértékelő függvény megadásában kulcsszerepet játszanak a heurisztikus függvények
 - Heurisztikus függvény:
 - ha n egy célállapot, akkor a heurisztikus függvény értéke 0;
 - $h(n)$: az n csomóponttól a célig vezető út költségének becslője

A mohó legjobbat–először keresés (greedy best-first search)

- azt a csomópontot fejt ki a következő lépésben, amelynek az állapotát a célállapothoz legközelebbinek ítéli;
- kiértékelő függvénye: $f(n) = h(n)$;
- problémák:
 - zsákutcákba jutva visszalép: ha nem ismeri fel az ismétlődő állapotokat, akkor soha nem találja meg a megoldást;
 - nem optimális;
 - nem teljes;

A^* keresés

- A legjobbat–először keresés egyik változata
- a csomópontokat úgy értékeli ki, hogy figyelembe veszi
 - az aktuális csomópontig megtett út költségét ($g(n)$)
 - az adott csomóponttól a célig vezető út becsült költségét ($h(n)$)
- kiértékelő függvénye: $f(n) = g(n) + h(n)$
- $f(n)$ jelentése: a legolcsóbb, az n csomóponton keresztül vezető megoldás becsült költsége;

Definíció

- *A csomópontokon értelmezett h heurisztikus függvény elfogadható heurisztika, ha a $h(n)$ érték soha nem becsüli felül az n csomópontból a cél eléréséhez szükséges költséget.*

Megjegyzés

- *A h függvény értéke csak a csomóponthoz tartozó állapottól függ.*
- *Az elfogadható heurisztikák optimisták.*
- *Mivel a $g(n)$ az n csomópont elérésének pontos költsége, ezért az f függvény ($f(n) = g(n) + h(n)$) soha nem becsüli túl az adott csomóponton át vezető legjobb megoldás valódi értékét.*
- *Pl.: útvonalkeresőnél a légvonalban mért távolság.*

Tétel

- Ha a h elfogadható heurisztika, akkor a FA-KERESÉS-t használó A^* algoritmus optimális.

Megjegyzés

- Még a h heurisztika esetén is előfordulhat, hogy a GRÁF-KERESÉS-t használó A^* algoritmus nem optimális.
 - Visszatérhet egy szuboptimális megoldással, mert elvetheti az ismétlődő optimális állapothoz vezető utat, ha az nem elsőnek került kiszámításra.
 - Megoldás 1.: a GRÁF-KERESÉS-t ki kell terjeszteni úgy, hogy az ugyanahhoz a csomóponthoz vezető két út közül a drágábbat vesse el. (Sokat kell adminisztrálni.)
 - Megoldás 2.: azt kell biztosítani, hogy bármelyik ismétlődő csomóponthoz vezető optimális utat elsőnek találja meg az algoritmus. (Az egyenletes költségű keresésnél az teljesült.)

Definíció

- A csomópontokon értelmezett h heurisztikus függvény konzisztens, ha minden n csomópontra és annak egy tetszőleges a cselekvéssel generált n' utódcsomópontjára teljesül a következő:

$$h(n) \leq c(n, a, n') + h(n')$$

ahol $c(n, a, n')$ az n állapotból n' állapotot eredményező a cselekvés lépésköltsége.

Megjegyzés

- A konzisztencia követelménye az általános háromszög egyenlőtlenség egy formája.
- Minden konzisztens heurisztika elfogadható heurisztika.

Tétel

- Ha h konzisztens heurisztika, akkor az f függvény bármely út mentén monoton növekvő (nem csökkenő).

Megjegyzés

- Ha n' az n utódja, akkor
- $g(n') = g(n) + c(n, a, n')$
- $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$

Tétel

- Ha a h konzisztens heurisztika, akkor a GRÁF-KERESÉS-t használó A^* algoritmus optimális.

- A gyökérből kiinduló utakat bővítő optimális algoritmusok közül az A^* keresési algoritmus bármely adott heurisztikus függvény mellett optimális hatékonyságú: egyetlen más optimális algoritmus sem fejt ki garantáltan kevesebb csomópontot, mint az A^* .
- Az A^* algoritmus
 - teljes
 - optimális
 - optimálisan hatékony
 - de: mégsem jó:
 - Az összes legenerált csomópontot a memóriában tárolja (ahogy ezt az összes GRÁF-KERESÉS algoritmus teszi), ezért az algoritmus nagyon hamar felemészti a rendelkezésre álló memóriát.

Példa: kirakójáték

- h_1 : a rossz helyen lévő lapkák száma
- h_2 : a lapkáknak a saját célhelyeiktől mért távolságaik összege: a Manhattan-távolság

Effektív elágazási tényező

- Ha az A^* algoritmus által kifejtett csomópontok száma egy adott problémára N , és megoldás mélysége d , akkor b^* annak a d mélységű kiegyensúlyozott fának az elágazási tényezőjével egyezik meg, amely $N + 1$ csomópontot tartalmazna:

$$N + 1 = 1 + b^* + (b^*)^2 + (b^*)^n$$

- Pl.: 5 mélység, 52 csomópont, $b^* = 1,92$
- h_2 jobb mint a h_1 ()és mindkettő jobb mint a nem informált keresés.
- Minden n csomópontban $h_2(n) \geq h_1(n)$, a h_2 domimálja a h_1 -et.
- Az A^* keresés egyenletes költségű keresés, ha $h(n) = 0$ minden n -re.

- A h_1 és h_2 alulról becsülik a fennmaradó út hosszát:
- Ha a játékot egyszerűsítjük, akkor a pontos úthossz értékét adják meg:
 - h_1 : egy lapka bárhová áthelyezhető, nemcsak a szomszédos mezőkre;
 - h_2 : egy lapka bármelyik szomszédos mezőre átmozgathat, még akkor is, ha a szomszédos mezőn már van lapka.
- Az operátorokra kevesebb megkötést adtunk mint az eredeti problémában: relaxált probléma.
- A relaxált probléma optimális megoldásának költsége egy elfogadható heurisztika az eredeti problémára.
- A relaxált problémákat automatikusan is elő lehet állítani.

- Ha a probléma megfogalmazása formális nyelven adott, akkor a relaxált problémákat automatikusan is elő lehet állítani.
- 8-as kirakójáték
 - Egy lapka az A mezőről a B mezőre mozgatható, ha az A és B szomszédosak, és a B mező üres.
- Relaxált problémák:
 - Egy lapka az A mezőről a B mezőre mozgatható, ha az A és B szomszédosak. (Manhattan-távolság vezethető le belőle.)
 - Egy lapka az A mezőről a B mezőre mozgatható, ha a B mező üres. (Gaschnig heurisztika)
 - Egy lapka az A mezőről a B mezőre mozgatható. (Levezethető heurisztika: nem a helyükön lévő lapkák száma.)
- Ha a relaxált problémákat nehéz megoldani, akkor a kapcsolatos heurisztikus értékek számítása nehéznek bizonyulhat. (Tökéletes heurisztika mindig megkapható egy teljes szélességi keresés lefuttatásával, de épp ezt akarjuk elkerülni!)

Melyik a jó heurisztikus függvény?

- Tegyük fel, hogy egy problémához adottak a $h_1, h_2, \dots, h_m$ elfogadható heurisztikus függvények és egyik sem dominálja a másikat!
- A lehető legjobb heurisztikus függvény:
$$h(n) = \max\{h_1(n), h_2(n), \dots, h_m(n)\}$$
- Ez mindig azt a függvényt használja, amely az adott csomópontra a legpontosabb.
- h elfogadható heurisztikus függvény.
- h konzisztens heurisztikus függvény.
- h dominálja a $h_1, h_2, \dots, h_m$ elfogadható heurisztikus függvényeket.

Részproblémákból származtatott heurisztikus függvények

- PL.: 1,2,3,4 lapkák helyükre mozgatása.
- A részprobléma optimális megoldásának költsége a teljes probléma megoldásának költségét alulról korlátozza.

Eddig megismert algoritmusok

- A keresési tereket szisztematikusan járják be.
- A szisztematikusság megvalósításához:
 - egy vagy több utat a memóriában tartanak;
 - minden pontban megjegyzik, hogy az út mentén melyik alternatívát vizsgálták már meg, és melyiket nem.
 - a célhoz vezető út egyben a probléma megoldása is.
- De:
 - számos probléma esetén az út érdektelen.

Lokális keresési algoritmusok

- Csak egy aktuális állapotot veszik figyelembe.
- Általában csak az aktuális állapot szomszédjaira lépnek tovább.
- A keresés során követett utat (tipikusan) nem tárolják el.

- Bár a lokális keresési algoritmusok nem szisztematikusak, két előnyük van:
 - igen kevés, általában konstans mennyiségű memóriát használnak
 - sokszor nagy (esetleg végtelen, folytonos) keresési térben elfogadható megoldást produkálnak (ezekben a terekben a szisztematikus algoritmusok hatékonyan nem alkalmazhatóak).
- Hasznosak a tisztán optimalizációs problémák megoldásában: a cél a legjobb állapot megtalálása egy célfüggvény értelmében.

A lokális keresés segédfogalmai

- Állapottér–felszín
 - van pontja: az állapot definiálja;
 - van magassága: a heurisztikus vagy a célfüggvény értéke határozza meg;
 - ha a magasság a költséggel arányos, akkor a cél a legalacsonyabban fekvő völgyet (globális minimum) megtalálása;
 - ha a magasság a célfüggvénynek felel meg, akkor a cél a legmagasabban fekvő csúcs (globális maximum) megtalálása;
- A teljes lokális keresés mindig talál megoldást, ha az egyáltalán létezik;
- Egy optimális algoritmus mindig megtalálja a globális minimumot vagy maximumot.

Hegymászó keresés (mohó lokális keresés)

- A keresés egy ciklus, ami mindig javuló értékek felé lép.
- Az algoritmus nem tart nyilván keresési fát:
 - A csomópontot leíró adatszerkezetnek csak az állapotot és a célfüggvény értékét kell nyilvántartania.
 - Csak az aktuális állapotot közvetlenül követő szomszédokat figyeli.
 - Gyakran igen gyorsan halad a megoldás felé.
 - De: gyakran megakad:
 - lokális maximum esetén;
 - hegygerinc setén (feljutunk a gerincre, de nem tudunk rajta haladni);
 - fennsík: nem tudja megtalálni a fennsíkról kivezető utat.

A hegymászó keresés algoritmus

```
1: function HEGYMASZAS(problema)
2:   inputs: problema, egy probléma
3:   local variable: aktualis, egy csomópont
4:   local variable: szomszed, egy csomópont
5:   aktualis  $\leftarrow$ 
      CSOMOPONT-LETREHOZ(KIINDULO-ALLAPOT[problema])
6:   loop
7:     szomszed  $\leftarrow$  az aktualis legnagyobb értékű követő csomópontja
8:     if ERTEK(szomszed)  $\leq$  ERTEK(aktualis) then
9:       return ALLAPOT(aktualis)
10:    end if
11:    aktualis  $\leftarrow$  szomszed
12:  end loop
13: end function
```

Állapottér-beli keresés esetén

- A keresési algoritmus számára mindegyik állapot egy olyan fekete doboz, amelynek a belső struktúrája nem ismert:
 - az állapotokat egy tetszőleges struktúra reprezentálja;
 - a struktúrához a problémára jellemző rutinokkal lehet hozzáférni:
 - az állapotátmenet függvényével;
 - a heurisztikát megadó függvényével;
 - a célállapottesztel.

Kényszerkielégítési problémák esetén

- az állapotok és a célteszt illeszkedik egy szabályos struktúrához;
- az állapotstruktúra segítségével keresési algoritmusokat lehet definiálni;
- nem problémaszpecifikus, hanem általános célú heurisztikák használatával nagy problémák megoldása is lehetővé válik;
- a célteszt szabályos reprezentációja feltárja magának a problémának a struktúráját;
- lehetővé válik a probléma dekompozíciója.

Kényszerkielégítési problémák

- Változók: $X_1, X_2, \dots, X_n$
- Kényszerek: $C_1, C_2, \dots, C_m$
- Változók értékeinek tartománya: $D_1, D_2, \dots, D_n$
 - $D_i \neq \emptyset$
- Minden egyes C_i kényszer a változók valamely részhalmazára vonatkozik.
 - Meghatározza a részhalmaz megengedett értékkombinációit.
- Problémaállapot: néhány (vagy mindegyik) változóhoz értéket rendelünk hozzá.
 - Egy hozzárendelés megengedett (konzisztens), ha egyetlen kényszer sem sért meg.
 - Teljes az a hozzárendelés, amelyben mindegyik változó szerepel.
 - Egy teljes hozzárendelés a kényszerkielégítési probléma megoldása.
 - Néha arra is szükség van, hogy a megoldás egy célfüggvényt maximalizáljon.

Probléma mint kényszerkielégítési probléma

- Előnyök:
 - Az állapotok reprezentációja miatt a kényszerkielégítési problémák egy sztenderd mintára illeszkednek.
 - A minta a hozzárendelt értékekkel rendelkező változók halmaza.
 - Az állapotátmenet függvényt és a célállapottesztet az összes kényszerkielégítési problémára általános módon meg lehet adni.
 - Létrehozhatók hatékony, általános heurisztikák minden tárgyszerület-specifikus tudás nélkül.
 - A kényszergráf struktúrájának segítségével lerövidíthető a megoldási folyamat (csökkentheti a probléma komplexitását).

Inkrementális megfogalmazás

- A kényszerkielégítési problémát szabályos keresési problémaként tekinti:
 - Kiinduló állapot: üres hozzárendelés, ahol egyetlen változónak sincs értéke.
 - Állapotátmenet-függvény: bármelyik hozzárendelés nélküli változó értéket kaphat, amennyiben az nem ütközik a korábbi értékadásokkal.
 - Célteszt: az aktuális hozzárendelés teljes.
 - Az út költsége: egy konstans költség mindegyik lépésre.
- Mindegyik megoldásnak egy teljes hozzárendelésnek kell lennie.
- n változó esetén az n -edik szinten jelenik meg.
- A keresési fa csak n mélységű.
- A mélységi algoritmusok a népszerűek.
- A megoldáshoz vezető út nem lényeges.
- Használható a teljes állapotleírás is: minden egyes állapot egy teljes változó-hozzárendelés: akár kielégíti a kényszereket, akár nem.
 - Ekkor a lokális keresési eljárások is használhatóak.

Legegyszerűbb eset

- A változók diszkrét és véges tartományúak.
- Pl.:
 - térképszínezési problémák
 - 8- királynő probléma (változók: $Q_1, \dots, Q_8$, tartomány: $\{1, 2, \dots, 8\}$)
- Ha n változónk van, és a változók tartozó tartomány legfeljebb d számosságú, akkor a lehetséges teljes hozzárendelések száma legfeljebb $\mathcal{O}(d^n)$ azaz a változók számának exponenciális függvénye.
- Boole kényszerkielégítési problémák: a változók értéke 0 vagy 1 (hamis vagy igaz).
- A diszkrét változók lehetnek véges tartományúak is:
 - Pl.: építkezési munka naptárának ütemezése.
 - Nem tudjuk a kényszereket a megengedett értékkombinációk felsorolásával leírni.
 - Ekkor kényszernyelvet kell használni.
 - Az ilyen kényszereket már nem lehet az összes lehetséges hozzárendelés felsorolásával megoldani.

Kényszerfajták

- Unáris kényszer: egy változó értékére tesz megkötést.
 - A kérdéses változó tartományának előfeldolgozásával az unáris kényszerek kiküszöbölhetőek.
- Bináris kényszer: két változót köt össze: ha csak bináris kényszer van, akkor a kényszerkielégítési problémát binárisnak nevezzük.
 - A bináris kényszerkielégítési problémákat kényszergráffal lehet reprezentálni: a gráf csomópontjai a változóknak, élei a kényszereknek felelnek meg.
- A magasabb rendű kényszerek legalább három változóra vonatkoznak.
 - Pl.: betűrejtvény
- A magasabb rendű kényszereket egy kényszer hipergráffal lehet ábrázolni:
 - A gráf csúcsai: változók + kényszerek
 - A gráf élei: a kényszer típusú csúcsok azokkal a változókkal vannak összekötve, amelyekre az adott kényszer vonatkozik.
 - De: Elegendő segédváltozó bevezetésével mindegyik magasabb rendű véges tartományú kényszer átírható bináris kényszerek halmazává.

Abszolút kényszer — preferenciakényszer

- Abszolút kényszer: a kényszer megszegése kizár egy megoldásjelöltet.
- Preferenciakényszer: jelzi, hogy mely megoldások preferáltak
 - A preferenciakényszereket gyakran az egyedei változó–hozzárendelések költségeként ábrázolják.

Problémák kommutativitása

- Egy probléma akkor kommutatív, ha a végeredmény szempontjából közömbös, hogy a cselekvések egy adott sorozatát milyen sorrendben alkalmazzuk.
- A kényszerkielégítési problémák kommutatívak.
- Mindegyik kényszerkielégítési problémamegoldó a következő állapot generálásakor a keresési fa minden csomópontjában csak egyetlen változó hozzárendeléseit veszi figyelembe.
 - Ezzel a megszorítással jelentősen csökken a keresési fa leveleinek a száma.

A visszalépéses keresés

- Olyan mélységi keresésekre használjuk, amelyek egyszerre csak egy változóhoz rendelnek értéket, és visszalépnek, ha már nincs megengedett hozzárendelési lehetőség.

- A kényszerkielégítési problémák hatékonyan megoldhatók tárgyterület-specifikus tudás nélkül is.
- Általános célú eljárások alakíthatóak ki.
- Kérdések:
 - A következő lépésben melyik változóhoz rendeljük értéket, és milyen sorrendben próbálkozzunk az értékekkel?
 - Milyen következményei vannak a jelenlegi változó-hozzárendeléseknek a még hozzárendeletlen változók számára?
 - Ha egy út sikertelennek bizonyul, a következő utak során el tudja-e kerülni a keresés ezt a hibát?

Változórendezés

- A legkevesebb fennmaradó érték heurisztika (MRV): azt a változót emeli ki, amelyik a legvalószínűbben fog hamarosan hibához vezetni.
 - Ha van egy változó, amelynek egyetlen megengedett értéke sincs, akkor az MRV heurisztika ki fogja választani ezt a változót, és azonnal kideríti a hibát, elkerülve a többi változó közötti haszontalan keresgélést.
- Fokszám heurisztika: azt a változót választja ki, amelyik legtöbbször szerepel a hozzárendeletlen változókra vonatkozó kényszerekben.

Értékrendezés

- Legkevesbé–korlátozó–érték heurisztika: előnyben részesíti azt az értéket, amely a legkevesebb választást zárja ki a kényszergráfban a szomszédos változóknál.
 - Ez a heurisztika a későbbi változó–hozzárendelések számára a lehető legnagyobb szabadságot meghagyni.
 - Ha az összes megoldást meg kell találni, akkor a sorrend közömbös.

Előrenéző ellenőrzés

- Minden egyes alkalommal, amikor egy X változó értéket kap, minden, az X -hez kényszerrel kötött, hozzárendeletlen Y -t megvizsgál, és Y tartományából törli az X számára választott értékkel inkonzisztens értékeket.
- Az MRV heurisztika az előrenéző ellenőrzés nyilvánvaló partnere.
- Még hatékonyabb megoldás: az MRV heurisztika munkájához szükséges információt inkrementálisan számítjuk.

A kényszerek terjesztése

- Az előrenéző ellenőrzés nem néz eléggé messze előre.
- Kényszerek terjesztése: ha az egyik változó kényszerének a többi változót érintő következményeit terjesztjük.
 - De: semmi értelme a keresés méretét csökkenteni, ha több időt töltünk a kényszerek terjesztésével, mint az egyszerű kereséssel.

Élkonzisztencia

- Az él a kényszergráf irányított éleit jelenti.
- az X -ből Y -ba mutató él akkor konzisztens, ha X minden x értékéhez található egy x -szel konzisztens y értéke Y -nak.
 - Egy él konzisztenssé tehető az olyan értékek törlésével, amelyhez nem létezik a végpontnak megengedett értéke.
 - Az élkonzisztencia ellenőrzés lehetővé teszi, hogy korábban észrevegyük az egyszerű előrenéző ellenőrzés által fel nem fedett inkonzisztenciát.
 - Alkalmazható előfeldolgozó lépésként a keresés megkezdése előtt.
 - A keresési folyamat minden egyes hozzárendelését követő terjesztési lépésként (az élkonzisztencia fenntartásának algoritmusa).
 - Mindkét előző esetben addig kell ismételve alkalmazni a folyamatot, amíg nem marad inkonzisztencia.
 - Ugyanis a törléssel a változóhoz mutató éleknél új inkonzisztencia jöhet létre.

Lokális keresés kényszerkielégítési problémáknál

- Hatékony: a kiinduló állapot minden változóhoz értéket rendel, és az állapotátmenet-függvény működése során általában egyszerre csak egy változó értékét módosítja.
- Min-konfliktusok heurisztika: azt az értéket választja ki, amelyik a legkevesebb konfliktust eredményezi más változókkal.
 - Megjegyzés: akár a millió–királynő problémát is megoldja átlagosan 50 lépésben.
 - Hubble: a megfigyelések ütemezéséhez szükséges három hetet tíz percre rövidítette le.
 - Alkalmazható online elrendezésben is, amikor a probléma változik (pl.: légitársaság heti ütemezése, időjárás-változás.)

Többágenses környezetek

- Minden ágensnek számolnia kell más ágensek cselekvéseivel:
 - kooperatív környezet;
 - verseny környezet: az ágensek céljai konfliktusban vannak.
- A verseny környezet: ellenségek melletti keresés:
 - gyakran kétszemélyes játékoknak nevezik.
- Matematikai játékelmélet: Neumann János
- Harsányi János a nem teljes információs játékok kutatója.
 - Közgazdaságtani Nobel Díjat kapott 1994-ben:
 - A nem kooperatív játékok elméletében az egyensúlyelemzés terén végzett úttörő munkásságért
- Az MI-ben a játékok specializáltak: determinisztikus, váltott lépésű, kétszemélyes, zérusösszegű teljes információjú játékok.
 - Két ágens helyezkedik el egy determinisztikus és teljesen megfigyelhető környezetben, a cselekvéseik váltják egymást, és a játék végén a hasznosságértékeik azonosak és ellentétes előjelűek.

- A kétszemélyes játékok azért érdekesek, mert nagyon nehéz őket megoldani.
 - Sakk: átlagos elágazási tényező: 35
 - Ha mindkét fél 50 lépést tesz meg, akkor a keresési fának 35^{100} , azaz 10^{154} csomópontja van.
- A játékok (a mindennapi élethez hasonlóan) azt a képességet igénylik, hogy valamilyen döntést hozzunk, akkor is, ha az optimális döntés kiszámítása kivitelezhetetlen.
- A játékok nagyon komolyan büntetik a rossz hatékonyságot: A játékelméleti kutatás számos olyan ötlethez vezetett, amely lehetővé tette a rendelkezésre álló idő minél jobb felhasználását.

Optimális döntések kétszemélyes játékokban

- Szereplők: MAX, MIN; (MAX lép először)
- A játékot egyfajta keresési problémaként lehet definiálni az alábbi komponensekkel:
 - Kiinduló állapot: táblaállás + ki lép
 - Állapotátmenet-függvény: (lépés, állapot) párok listájával tér vissza: megadja a legális lépéseket és a létrejövő állapotokat.
 - Végteszt (terminál teszt): megadja, hogy mikor van vége a játéknak.
 - Hasznosságfüggvény: a játék végeredményéhez egy számértéket rendel.
- Játékfa: a kezdeti állapot és mindkét fél legális lépései által generált fa.
- Pl.: 3×3 -as amőba

Logikai ágensek

Tudásalapú ágensek

- Feltételezi:
 - a tudás reprezentációját;
 - a tudás alkalmazását lehető tevő következtetési folyamatokat.
- A tudásbázisú ágens képes kihasználni a nagyon általános formában leírt tudást.
- A tudásbázisú ágens képes összekombinálni az általános tudást a pillanatnyi érzetekkel.
- A tudás reprezentálásának elsődleges eszköze: a logika.
- A logikai ágensek tudása mindig határozott: minden kijelentés vagy igaz vagy hamis az adott világban.
 - De: a bizonytalan tudás felhasználása problematikus.

A tudásbázisú ágens

- Központi eleme: a tudásbázis
 - A tudásbázis állításoknak egy halmaza.
 - Az állításokat egy nyelv segítségével fejezzük ki: ezt a nyelvet tudásrepresentációs nyelvnek nevezzük, és a világról szóló állításokat fogalmazzunk meg segítségével.
- Szükséges eljárások:
 - KIJELENT: állítások tudásbázishoz való hozzáadását valósítja meg;
 - KÉRDEZ: a tudás lekérdezését valósítja meg.
- Mindkét feladat (eljárás) tartalmazhat következtetést: új állítások levezetését a meglévőkből.

Általános tudásbázisú ágens

```

1: function TB-AGENS(eszleles)
2:   static: TB, egy tudásbázis
3:   static: t, egy számláló, kezdetben 0, mutatja az időt
4:   KIJELENT(TB, ESZLELES – MONDAT – KESZITES(eszleles, t))
5:   cselekves ←
     KERDEZ(TB, CSELEKVES – KERDEZES – KESZITES(t))
6:   KIJELENT(TB, CSELEKVES – MONDAT – KESZITES(t))
7:   t ← t + 1
8:   return cselekves
9: end function

```

- *ESZLELES – MONDAT – KESZITES* eljárás egy olyan állítást konstruál, amelyik megállapítja, hogy az ágens egy adott pillanatban észlelte az érzékelt dolgot.
- *CSELEKVES – KERDEZES – KESZITES* az időt felhasználva bemeneti adatként, visszatér egy mondattal, amely alkalmas arra, hogy megkérdezzük milyen cselekvés szükséges az adott pillanatban.
- *CSELEKVES – MONDAT – KESZITES* egy olyan állítást hoz létre, amely megállapítja, hogy a kiválasztott cselekvés végrehajtása megtörtént.
- A következtetési mechanizmus részletei a *KIJELENT* és a *KERDEZ* eljárások belsejében van elrejtve.

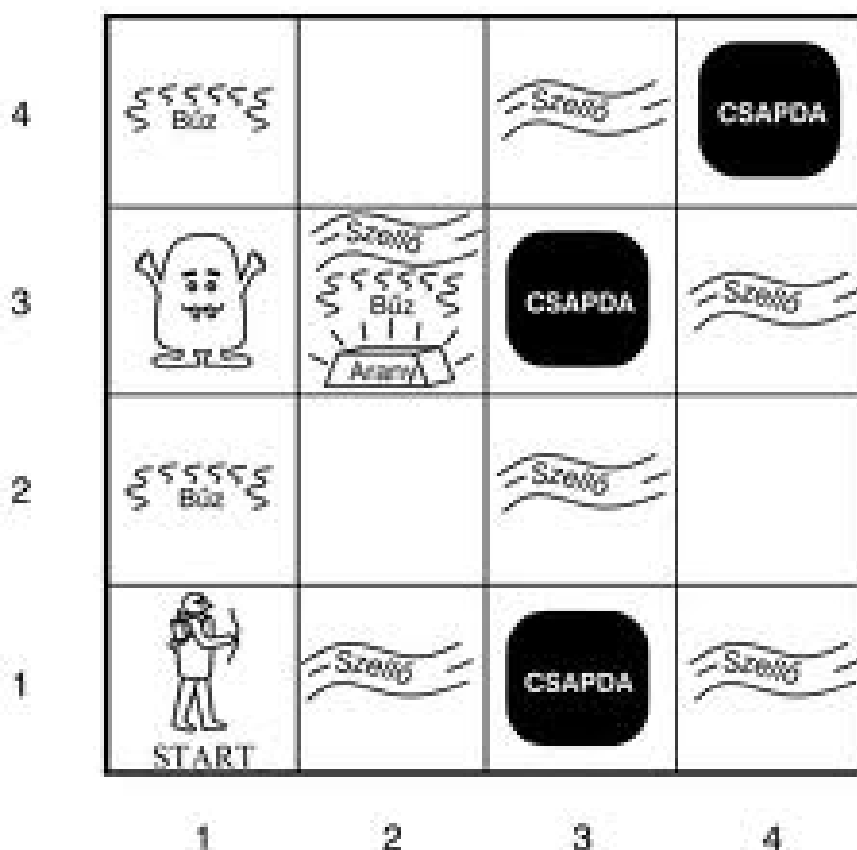
A wumpus világ

- Egy barlang, amely szobákból és átjárókból áll.
- A wumpus egy szörnyeteg, aki mindenkit megesz, ha a szobájába lép, a barlangban lapul valahol.
- Az ágens le tudja lőni a wumpust, de csak egyetlen nyila van ehhez.
- Néhány szoba feneketlen csapdát tartalmaz, amely mindenkit csapdába ejt, aki belép a szobába (kivéve a wumpust).
- A wumpus környezetében egy halom aranyat lehet találni.

A wumpus világ

- Teljesítményérték: +1000 az arany felvétele; −1000 a csapdába esés, vagy ha felfal a wumpus; −1 minden végrehajtott cselekvés; −10 a nyíl használata.
- Környezet: 4×4 -es háló; az ágens mindig az $\langle 1, 1 \rangle$ -ből indul (arccal jobbra nézve). Az arany és a wumpus elhelyezkedése véletlenszerű (egyenletes eloszlású). Bármely négyzet 0.2 valószínűséggel csapda.
- Cselekvés: az ágens előre mehet, jobbra, balra fordulhat; meghal, ha csapdába, vagy élő wumpust tartalmazó szobába lép; megragad; lő.
- Érzékelők: bűz; szellő; csillogás; ütés (ha falnak megy); sikoly, ha megölték a wumpust (bárhon hallható);
- Az érzeteket az ágens egy lista formájában kapja meg. Pl.: $\langle \text{Bűz, Szellő, Nincs, Nincs, Nincs} \rangle$

A wumpus világ



A wumpus világ

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 [A] OK	2,1 OK	3,1	4,1

[A] = Ágens
 Sz = Szellő
 R = Ragyogás, Arany
 OK = Biztonságos négyzet
 Cs = Csapda
 B = Bűz
 M = Megláthatott
 W = Wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 Cs?	3,2	4,2
OK			
1,1 M OK	2,1 [A] Sz OK	3,1 Cs?	4,1

(a)

(b)

A wumpus világ

1,4	2,4	3,4	4,4
1,3 W?	2,3	3,3	4,3
1,2 [A] B OK	2,2 OK	3,2	4,2
1,1 M OK	2,1 B M OK	3,1 Cs?	4,1

[A] = Ágens
 Sz = Szellő
 R = Ragyogás, Arany
 OK = Biztonságos négyzet
 Cs = Csapda
 B = Bűz
 M = Megláthatott
 W = Wumpus

1,4	2,4 Cs?	3,4	4,4
1,3 W?	2,3 [A] B R Sz	3,3 Cs?	4,3
1,2 B B OK	2,2 B OK	3,2	4,2
1,1 M OK	2,1 B M OK	3,1 Cs?	4,1

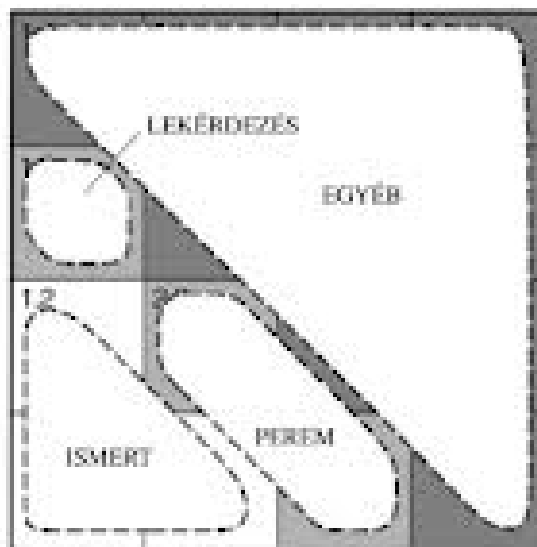
(a)

(b)

A wumpus világ

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 B OK	2,2	3,2	4,2
1,1 OK	2,1 B OK	3,1	4,1

(a)



(b)

- Formális (formalizált) nyelv
- Szemantika
- A modell fogalma: Hol igaz TB minden állítása?.
- A következmény fogalma: szemantikai, szintaktikai
- Helyesség és teljesség
- Mi a kapcsolata a TB-nek a világgal?

A wumpus tudásbázisa

- $p_{i,j}$: csapda van $\langle i, j \rangle$ -ben;
- $q_{i,j}$: szellő van $\langle i, j \rangle$ -ben.
- r_1 : nincs csapda $\langle 1, 1 \rangle$ -ben: $\neg p_{1,1}$
- Egy négyzet akkor és csak akkor szellős, ha csapda van a szomszédos négyzetben.
 - $s_{1,1} =_{\text{def}} q_{1,1} \equiv p_{1,2} \vee p_{2,1}$
 - $s_{4,1} =_{\text{def}} q_{4,1} \equiv p_{3,1} \vee p_{4,2}$
 - $s_{1,4} =_{\text{def}} q_{1,4} \equiv p_{1,3} \vee p_{2,4}$
 - $s_{4,4} =_{\text{def}} q_{4,4} \equiv p_{3,4} \vee p_{4,3}$
 - $s_{1,j} =_{\text{def}} q_{1,j} \equiv p_{1,j+1} \vee p_{2,j} \vee p_{1,j-1}$ ha $j = 2, 3$
 - $s_{4,j} =_{\text{def}} q_{4,j} \equiv p_{4,j+1} \vee p_{3,j} \vee p_{4,j-1}$ ha $j = 2, 3$
 - $s_{i,1} =_{\text{def}} q_{i,1} \equiv p_{i-1,1} \vee p_{i,2} \vee p_{i+1,1}$ ha $i = 2, 3$
 - $s_{i,4} =_{\text{def}} q_{i,4} \equiv p_{i-1,4} \vee p_{i,3} \vee p_{i+1,4}$ ha $i = 2, 3$
 - $s_{i,j} =_{\text{def}} q_{i,j} \equiv p_{i-1,j} \vee p_{i+1,j} \vee p_{i,j-1} \vee p_{i,j+1}$ ha $0 < i-1, j-1$ és $i+1, j+1 < 4$.
- Ezek a formulák minden wumpus világban igazak.

- Tekintsük most az első két meglátogatott négyzet utáni állapotot:
 - r_1 , azaz $\neg p_{1,1}$ teljesül;
 - tudjuk, hogy $\neg(q_{1,1} \equiv p_{1,2} \vee p_{2,1})$ azaz $\neg s_{1,1}$ teljesül;
 - tudjuk, hogy $q_{2,1} \equiv p_{1,1} \vee p_{2,2} \vee p_{3,1}$ azaz $s_{2,1}$ teljesül.
- 7 paraméter, összesen $2^7 = 128$ interpretáció.
- 3 interpretáció modellje TB-nek.
- TB modelljeiben $\neg p_{1,2}$ teljesül, azaz nincs csapda $\langle 1, 2 \rangle$ -ban.
- De: $p_{2,2}$ kétféleképpen igaz, egyben hamis, így még nem tudjuk megmondani, hogy van-e csapda $\langle 2, 2 \rangle$ -ben.

Logikai fogalmak

- Kielégíthetőség, kielégíthetetlenség
- Logikai ekvivalencia
- Érvényesség
- Dedukció tétel
- Következtetési minták az állításlogikában: a természetes levezetés rendszere.
- A monotonitás jelentősége.
- A kompaktság jelentősége.

- A következtetési szabályok nélkül a modellek felsorolása alkalmas a feltett kérdés megválaszolására.
- A következtetési szabályok összes lehetséges alkalmazásának generálására definiálhatunk egy új állapotátmenet-függvényt:
 - az eddigi kereső algoritmusok felhasználhatóak a feltett kérdés megválaszolására;
 - a keresési feladat haladhat előre: a kezdeti tudásbázisból kiindulva, alkalmazva a következtetési szabályokat a célmondat levezetéséig;
 - visszafelé a célmondattól, megpróbálva megtalálni a következtetési szabályoknak olyan láncolatát, amely a kiindulási tudásbázisra alkalmazható szabályokból indul ki.
- A legrosszabb esetben egyik módszer sem hatékonyabb a másiknál.
- A gyakorlatban: a következtetési szabályok használata sokkal hatékonyabb lehet, mert képes figyelmen kívül hagyni az irreleváns állításokat (függetlenül attól, hogy hány van belőlük).

Nulladrendű rezolúció

- A természetes levezetés szabályai nemcsak helyesek, hanem teljesek is.
- Nulladrendű rezolúció:
 - olyan következtetési szabály, amelynek alkalmazása, párosítva bármelyik keresési módszerrel, egy teljes következtetési algoritmust eredményez;
 - alapja az egyik automatikus tételbizonyítási módszernek, a rezolúciós kalkulusnak.
 - a logikai programozás alapnyelve, a Prolog, a rezolúció egy fajtájának az algoritmikus megvalósítása.
- Alapötlet: két formulához hozzárendelünk egy speciális formulát, a rezolvensüket, amely következik a kiinduló formulákból.
 - Legyenek A, B, C tetszőleges nulladrendű formulák.

$$(A \vee C) \wedge (B \vee \neg C) \models (A \vee B)$$

$$(A \vee C) \wedge (B \vee \neg C) \vdash (A \vee B)$$

- Cáfolási (megcáfolási) teljesség: képes annak az eldöntésére, hogy egy formula következménye-e egy (véges) formulahalmaznak:
 - el tudja dönteni, hogy az $A_1 \wedge A_2 \wedge \dots \wedge A_n \wedge \neg B$ formula kielégíthetetlen-e, azaz hogy teljesül-e az, hogy $\{A_1, A_2, \dots, A_n\} \models B$;
 - de: nem alkalmazható az összes következmény felsorolására, az összes olyan mondat megadására, amely egy adott tudásbázis esetén igaz.
 - Pl.: el tudja dönteni, hogy $(A \models A \vee B)$, de mint következtetési szabály A -ból nem tud eljutni az $(A \vee B)$ -ig.
- A rezolúció alapjául szolgál teljes következtetési algoritmusok egy családjának, azoknak, amelyekben azt kell tesztelni, hogy az adott tudásbázisból következik-e egy állítás.

A rezolúció fogalmi eszközei/1

- Literál: Ha $p \in Con$, akkor p -t és $\neg p$ -t p alapú literálnak nevezzük.
- A p literál kiegészítő literálja $\neg p$, a $\neg p$ literál kiegészítő literálja p .
- Egy literált vagy literálok diszjunkcióját klóznak (clause) nevezzük. Ha a klóz egy literál, akkor egységklóznak nevezzük.
- Megállapodás: az egyszerűség érdekében az egyetlen egy literált sem tartalmazó üres karaktersorozat üres klóznak nevezzük, jele: $\square$.
- Minden klózhoz egyértelműen hozzárendelhető literálok egy véges halmaza: Ha $A_1, A_2, \dots, A_n$ (nem feltétlenül különböző) literálok, akkor a $A_1 \vee A_2 \vee \dots \vee A_n$ klózhoz rendelt literálhalmaz $\{A_1, A_2, \dots, A_n\}$.
 - Egy n diszjunktív tagot tartalmazó klóz literálhalmaza lehet n -nél kisebb számosságú.
 - A diszjunkció asszociativitása, kommutativitása és idempotenciája miatt a klóz literálhalmaza használható a klóz helyett: a klóz literálhalmazának elemeiből képzett diszjunkció logikailag ekvivalens a klózzal.
 - Az üres klóz literálhalmaza az üres halmaz.

A rezolúció fogalmi eszközei/2

- Az üres klóz ($\square$) felfogható egy olyan formulának, amelynek az értéke mindig hamis, hisz ez egy tagok nélküli diszjunkció, és ahhoz, hogy egy diszjunkció igaz legyen legalább egy tagjának igaznak kell lennie.
- Egy elemi diszjunkció (egy literál, vagy különböző alapú literálok diszjunkciója) mindig klóz, de nem minden klóz elemi diszjunkció.
- Elemi diszjunkciók konjunkcióját konjunktív normálformának nevezzük.
- A továbbiakban használni fogjuk az alábbi állításlogikai tételt: Ha A nem érvényes formula, akkor van vele logikailag ekvivalens konjunktív normálformájú formula.
- Ha A egy formula, akkor van vele logikailag ekvivalens olyan formula, amely egy klóz vagy klózek konjunkciója.
- Megjegyzés: A mesterséges intelligencia irodalmában gyakran az utóbbi alakot is konjunktív normálformaként emlegetik.

A rezolúció fogalmi eszközei/3

• Az előállítás lépései:

- ① kiküszöböljük a materiális ekvivalenciát:
 $(A \equiv B) \Leftrightarrow (A \supset B) \wedge (B \supset A);$
- ② kiküszöböljük az implikációt: $(A \supset B) \Leftrightarrow (\neg A \vee B);$
- ③ elérjük, hogy a negáció csak állításparaméterekre vonatkozzék:
 $\neg\neg A \Leftrightarrow A, \neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B), \neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B);$
- ④ alkalmazzuk a disztributivitási szabályokat:
 $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C), A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

A rezolúció fogalmi eszközei/4

• Az egységrezolúció szabálya:

- Legyen A egy klóz, melynek literálhalmaza $L(A)$, B egy egységklóz, B' pedig a B literál kiegészítő literálja. Ekkor ha $B' \in L(A)$, akkor A és B rezolvense az $L(A) \setminus \{B'\}$ literálhalmazzal definiált klóz.

• Teljes rezolúciós szabály:

- Legyen A, B két klóz, melyeknek literálhalmaza $L(A), L(B)$. Ha van olyan $C \in L(B)$, hogy $C' \in L(A)$, ahol C' C kiegészítő literálja, akkor az A és B klózok rezolvense az $L(A) \cup L(B) \setminus \{C, C'\}$ literálhalmazzal definiált klóz.

Példa/1

- Legyen $A = l_1 \vee \dots \vee l_k$, B pedig l_i kiegészítő literálja és tegyük fel hogy $l_1, \dots, l_k$ különböző literálok. Ekkor az egységrezolúciós szabály a következő alakot ölti:

$$\frac{l_1 \vee \dots \vee l_k, \quad B}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k}$$

Példa/2

- Legyen $A = l_1 \vee \dots \vee l_k$, $B = m_1 \vee \dots \vee m_n$ és tegyük fel hogy l_i és m_j kiegészítő literálok. Tegyük fel továbbá, hogy $L(A) = \{l_1, \dots, l_k\}$ és $L(B) = \{m_1, \dots, m_n\}$, azaz mind A mind B literáljai különbözőek. Ekkor a teljes rezolúciós szabály a következő alakot ölti:

$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

Példa/3

- Ha csak kettő hosszúságú klózokkal foglalkozunk, azaz $A = l_1 \vee l_2$, $B = \neg l_3 \vee l_3$, akkor a teljes rezolúciós szabály a következő alakot ölti:

$$\frac{l_1 \vee l_2, \quad \neg l_2 \vee l_3}{l_1 \vee l_3}$$

- Üres klózhoz akkor jutunk, ha kiegészítő literálokat rezolválunk:

$$\frac{p, \quad \neg p}{\square}$$

A rezolúció algoritmusa

- Az ellentmondásokra vezető bizonyítások elvén működnek:
 - azt kell belátni, hogy egy formula kielégíthetetlen:
 - Legyen TB a tudásbázist leíró formulák konjunkciója, A pedig az a formula, amelynek teljesülésére kíváncsiak vagyunk. A akkor fog bizonyosan teljesülni a tudásbázis mellett, ha $TB \models A$ teljesül. Ez utóbbi pedig akkor teljesül, ha a $TB \wedge \neg A$ formula kielégíthetetlen.
- Elsőször a $TB \wedge \neg A$ formulát konvertáljuk olyan alakra, amely klózok konjunkciójából áll (gyenge konjunktív normálforma).
- A konjunkció tényezőiként szereplő klózokból álló klózalmazra alkalmazzuk a rezolúciós szabályt.
 - Minden egyes párt, amely kiegészítő literálokat tartalmaz rezolválunk, a létrejött új klózt hozzáadjuk a klózalmazhoz (ennek akkor van hatása, ha az új klóz még nem eleme a klózalmaznak).
 - A folyamat addig folytatódik, ameddig a következő két eset valamelyike nem következik be:
 - nincs több új klóz amit hozzá lehet adni: ebben az esetben $TB \models A$;
 - a rezolúció alkalmazása egy üres klózra vezet: ekkor $TB \models A$.

Előre- és hátrafelé láncolás

- A rezolúciót a teljesség tulajdonsága fontos következtetési módszerre teszi.
 - De: számos esetben a rezolúció teljes erejére nincs szükség.
 - Ekkor a TB a klózoknak csak egy speciális fajtáját tartalmazza, a Horn-klózokat.

Definíció

A Horn-klóz literálok olyan diszjunkciója, amelyek közül legfeljebb egy pozitív (nem tartalmaz negációt).

- Minden Horn-klóz felírható egy implikációként.
- A Horn-klózokat határozott klózoknak nevezik:
 - a pozitív literál: a klóz feje;
 - a negatív literálok alkotják a klóz testét.
- A negatív literálok nélküli klózt ténynek nevezzük.

- A határozott klózok alkotják a logikai programozás alapját.
- Egy pozitív literál nélküli Horn-klóz felírható olyan implikációként, amelynek utótagja a falsum ($\perp$):
 - Az ilyen formulákat az adatbázis-kezelés területén integritás kényszernek nevezik (adatbázishibát jelez).
- Horn-formájú tudásbázisok:
 - határozott klózokat tartalmaz;
 - nincsenek benne integritáskényszerek.
- A továbbiakban az egyszerűség kedvéért csak Horn-formájú tudásbázisokkal foglalkozunk.
- A Horn-klózokon végzett következtetés két tipikus algoritmus:
 - előre felé láncolás;
 - hátrafelé láncolás.
- A következtetés helyességének eldöntéséhez szükséges idő lineárisan függ a tudásbázis méretétől.

Előrefelé láncolás

- Az algoritmus a tudásbázisban található ismert tényekből (pozitív literálok) indul ki.
- Kérdése: egy állításparaméter következménye-e a Horn-klózat tartalmazó tudásbázisnak?
- Ha egy implikáció minden előtagja ismert, akkor az utótagját hozzáadjuk az ismert tények halmazához.
- A folyamat megáll:
 - ha a kérdésben szereplő állításparamétert hozzá tudjuk adni az ismert tények halmazához;
 - ha már nem tudunk további következtetést végrehajtani.
- Az előrefelé láncolás az általános adatvezérelt következtetési elvnek egy példája:
 - olyan következtetési elv, amelyben a figyelem fókusza kezdetben az ismert adatokon van;
 - a mindennapi életben gyakori, de kontroll alatt kell tartani.

Hátrafelé láncolás

- A lekérdezésből kiindulva működik.
 - Az algoritmus megtalál minden olyan implikációt a tudásbázisban, amelynek következménye a lekérdezésben szereplő állításparaméter.
 - Ha valamelyik ilyen implikációnak az összes előtagját be lehet bizonyítani, akkor a lekérdezésben szereplő állításparaméter igaz.
- A hátrafelé láncolás a a célorientált következtetés egy formája.
- Egy ágensnek meg kell osztania a munkát az előrefelé és a hátrafelé láncolás között:
 - korlátozni kell az előrefelé láncolást a releváns tényekre;
 - a releváns tények hátrafelé láncolás útján derülhetnek ki.

Példa: előre felé — hátrafelé láncolás

- Tudásbázis:
 - $p \supset q$
 - $r_3 \wedge r_4 \supset p$
 - $r_2 \wedge r_3 \supset r_4$
 - $r_1 \wedge p \supset r_3$
 - $r_1 \wedge r_2 \supset r_3$
 - r_1
 - r_2
- Kérdés: Igaz-e q ?

DPLL algoritmus

- Három szinten fejleszti tovább az IK–VONZAT? algoritmust:
 - 1 korai leállítás;
 - 2 tiszta szimbólum heurisztika;
 - 3 egységklóz heurisztika.

Korai leállás

- Az algoritmus észreveszi, hogy egy biztosan igaz vagy hamis még részben elkészült modell alapján is:
 - egy klóz igaz, ha bármelyik literál igaz;
 - ha bármelyik klóz hamis (azaz minden literálja hamis), akkor a formula hamis.
- A korai leállás a keresési tér egész részfáinak átvizsgálását kerüli el.

Tiszta szimbólum heurisztika

- Egy tiszta szimbólum egy olyan szimbólum, amely mindig ugyanolyan "előjellel" szerepel.
 - $p \vee \neg q, \neg q \vee \neg r, p \vee r$: p, q tiszta, r nem tiszta.
- Ha egy formulának van modellje, akkor akkor létezik olyan tiszta szimbólumokat tartalmazó modellje is, amelyben a tiszta igazak.
- Fontos: a szimbólum tisztaságának meghatározásakor az algoritmus figyelmen kívül hagyhatja azokat a klózokat, amelyekről már tudjuk, hogy igazak a modell eddig konstruálása alapján.

Egységklóz heurisztika

- Az egységklózokra vonatkozó hozzárendelést végiggördíti a klózokon mielőtt elágazna a maradékon.

- Az állításlogika nyelve:
 - reprezentatív: képes a tudás reprezentálására;
 - deklaratív természetű: a tudás és a következtetés különálló fogalmak, a következtetés helyessége csak a logikai sajátosságoktól függ;
 - kompozicionális;
 - nem kontextusfüggő
 - de: a kifejező ereje nagyon korlátozott: sok objektumot tartalmazó környezet tömör leírását nem teszi lehetővé.
- A természetes nyelvek lehetővé teszik a környezet tömör és összefogott leírását.
- A természetes nyelvet deklaratív tudásreprezentációs nyelvnek (is) tekintik.
 - kontextusfüggő,
 - többértelmű.
- Amire szükség van: egy deklaratív, reprezentatív, kompozicionális, kontextusfüggetlen, nagy kifejezőerővel rendelkező nyelv.
- Az elsőrendű logika nyelve

Az elsőrendű logika nyelve

- $L^{(1)} = \langle LC, Var, Con, Term, Form \rangle$
 - Nevek — objektumok: $Term$ — U elemei
 - Műveletek kifejezői — műveletek reprezentálása:
 - függvényjelek — $f : U^{(n)} \rightarrow U$
 - Tulajdonságok kifejezői — tulajdonságok reprezentálása:
 - egyargumentumó predikátumparaméterek — U részhalmazai
 - Relációk kifejezői — relációk reprezentálása:
 - n -argumentumó predikátumparaméterek ($n \geq 2$) — $U^{(n)}$ részhalmazai
 - Kvantorok: univerzális ($\forall$), egzisztenciális ($\exists$)
 - Azonosság (egyenlőség): $=$

Az elsőrendű logika szemantikája

- Interpretáció: $\langle U, \varrho \rangle$
- Értékelés: $v : Var \rightarrow U$
- Modell: $\langle U, \varrho, v \rangle$

Állítások és lekérdezések az elsőrendű logikában

- Az állításokat a *KIJELENT* segítségével adjuk hozzá a tudásbázishoz.
 - $KIJELENT(TB, Kiraly(Janos))$
 - $KIJELENT(TB, \forall x(Kiraly(x) \supset Szemely(x)))$
- Lekérdezés: *KERDEZ*:
 - $KERDEZ(TB, Szemely(Janos))$: válasz 0 vagy 1 (igaz vagy hamis)
 - $KERDEZ(TB, \exists x Szemely(x))$: válasz: helyettesítési lista: változó/terminus párok halmaza (pl.: $\{x/Janos, x/Istvan\}$)

Példa/1: rokonsági kapcsolatok

- Szándékolt objektumok: személyek
- Egyargumentumú predikátumok: *Ferfi*, *No*
- Kétargumentumú predikátumok: *Szuloje*, *Testvere*, *Fivere*, *Novere*, *Gyereke*, *Lanya*, *Fia*, *Hazastarsa*, *Felesege*, *Ferje*, *Nagyszuloje*, *Unokatestvere*, *Nagynenje*, *Nagybattyja* stb.
- Függvények: *Anyja*, *Apja*
- Alapvető tények (axiómák) a tárgyterület felépítésre szolgálnak:
 - $\forall x \forall y (Anyja(y) = x \equiv No(x) \wedge Szuloje(x, y))$
 - $\forall x \forall y (Ferje(y, x) \equiv Ferfi(y) \wedge Hazastarsa(y, x))$
 - $\forall x (Ferfi(x) \equiv \neg No(x))$
 - $\forall x \forall y (Szuloje(y, x) \equiv Gyereke(x, y))$
 - $\forall x \forall y (Nagyszuloje(x, y) \equiv \exists z (Szuloje(x, z) \wedge Szuloje(z, y)))$
 - $\forall x \forall y (Testvere(x, y) \equiv (\neg(x = y) \wedge \exists z (Szuloje(z, x) \wedge Szuloje(z, y))))$
- Tétel-e a következő:

$$KERDEZ(TB, \forall x \forall y (Testvere(x, y) \equiv Testvere(y, x)))$$

Példa/2: természetes számok (a teljes indukció nélkül)

- Egyargumentumú predikátum: *TermSzam*
- Függvény: *S* (a rákövetkezés művelete)
- Axiómák:
 - $TermSzam(0)$
 - $\forall x (TermSzam(x) \supset TermSzam(S(x)))$
 - $\forall x \neg(0 = S(x))$
 - $\forall x \forall y (\neg(x = y) \supset \neg(S(x) = S(y)))$
- Az összeadás definiálása:
 - $\forall x (TermSzam(x) \supset (+ (0, x) = x))$
 - $\forall x \forall y (TermSzam(x) \wedge TermSzam(y) \supset (+ (S(x), y) = S(+ (x, y))))$
- Asszociatív-e a művelet?
 $KERDEZ(TB, \forall x \forall y \forall z (+ (x, + (y, z)) = (+ (+ (x, y), z))))$

Tudástervezés az elsőrendű logikában

- Tudástervezés: a tudásbázis felépítése
- A tudástervezés folyamata:
 - A feladat beazonosítása: Analóg a TKBÉ-folyamat során ismertetett lépésekkel.
 - A releváns tudás összegyűjtése (ezen a szinten tudást formálisan nem reprezentáljuk).
 - Meg kell határozni a predikátumparaméterek, függvények és névparaméterek szótárát. Az eredmény egy szótár: ezt a szótárt a tárgysterület ontológiájának nevezzük.
 - A tárgysterületről szóló általános tudás kódolása.
 - Az adott problémapéldány leírásának kódolása.
 - Lekérdezéseket fogalmazunk meg a következtetési folyamat számára és válaszokat vezetünk le.
 - Kiszűrjük a hibákat a tudásbázisból (pl.: pótoljuk a hiányzó axiómákat).

Kvantorokra vonatkozó következtetési szabályok/1

- Például:
 - $\forall x(Kiraly(x) \wedge Moho(x) \supset Gonosz(x))$
 - $Kiraly(Janos) \wedge Moho(Janos) \supset Gonosz(Janos)$
 - $Kiraly(Richard) \wedge Moho(Richard) \supset Gonosz(Richard)$
- Univerzális példányosítás:
 - $\forall x A \models A_x^t$, ha az x változó helyettesíthető a t terminussal az A formulában.
 - Jelölés: $HELYETTESIT(\theta, A) = A_x^t$, ahol $\theta = \{x/t\}$ (az x változó minden szabad előfordulását a t terminussal helyettesítjük az A formulában).
 - $\forall x A \models HELYETTESIT(\{x/t\}, A)$
 - Az univerzális példányosítás bármely olyan terminussal elvégezhető, amelyre teljesül, hogy x behelyettesíthető t -vel A -ban (változókat nem tartalmazó úgynevezett alapterminusok esetén ez mindig teljesül).

Kvantorokra vonatkozó következtetési szabályok/2

- János van anyja. Nevezzük el Mariskának. Ekkor Mariska anyja Jánosnak, de csak olyan nevet választhattunk, amely nem fordul elő máshol a tudásbázisban.
- Egzisztenciális példányosítás:
 - $\exists x A \models A_x^c$, ha a c névparaméter nem fordul elő a tudásbázisban, azaz egy eddig nem használt névparamétert helyettesítünk be. (A névparaméterek esetében mindig teljesül, hogy az x változó helyettesíthető egy névparaméterrel az A formulában.)
 - A felhasznált c névparamétert szokták Skolem–konstansnak nevezni.
 - Az egzisztenciális példányosítás csak egyszer végezhető el.

Redukálás állításlogikára

- Az egzisztenciális kvantorral kezdődő formulát felcserélhettünk a formula egy speciális példányával.
- Egy univerzális kvantorral kezdődő formulát felcserélhetünk egy formulahalmazzal, amely a formula összes lehetséges alapterminussal való példányosítását tartalmazza:
 - TB
 - $\forall x(Kiraly(x) \wedge Moho(x) \supset Gonosz(x))$
 - $Kiraly(Janos)$
 - $Moho(Janos)$
 - $Fiver(Richard, Janos)$
 - Példányosítás: $\{x/Janos\}, \{x/Richard\}$
 - $Kiraly(Janos) \wedge Moho(Janos) \supset Gonosz(Janos)$
 - $Kiraly(Richard) \wedge Moho(Richard) \supset Gonosz(Richard)$
- Ez az állításlogikára való visszavezetés technikája.

- Az állításlogikára való visszavezetés technikája:
 - Minden elsőrendű tudásbázis és lekérdezés átalakítható állításlogikai formulákra úgy, hogy a tudásbázis következményei nem változnak.
- De: ha a tudásbázis tartalmaz függényszimbólumot, a lehetséges alapterminusok, a lehetséges alapterminusok helyettesítéseinek halmaza végtelen. Pl.: *Apja*:
 $Apja(Janos), Apja(Apja(Janos)), Apja(Apja(Apja(Janos)))$ stb.
- Herbrand tétele: Ha egy formula következik az eredeti elsőrendű tudásbázisból, akkor létezik olyan bizonyítás, amely csak egy véges részhalmazt használ fel az állításlogikára átalakított tudásbázisból.
- Ezt a részhalmazt meg lehet találni úgy, hogy először generáljuk az összes példányt
 - az állásparaméterekhez: megnézzük teljesül-e a következmény;
 - ha nem, akkor hozzáadjuk a tudásbázishoz az összes I-es mélységű terminust: megnézzük teljesül-e a következmény;
 - ha nem, akkor hozzáadjuk a tudásbázishoz az összes II-es mélységű terminust: megnézzük teljesül-e a következmény;
 - stb. mindaddig, amíg nem teljesül a következmény.

- A felvázolt eljárás teljes: bármely következményt bizonyítani tudunk.
- De: Mi történik, ha a lekérdezésben szereplő formula nem következmény?
- Erre a kérdésre a válasz elsőrendű logikában nemleges: végtelen ciklusba kerülhetünk.
- Ez hasonló a Turing gépek leállási problémájához.
- Turing, Church tétel: Az elsőrendű logikában a következmény kérdése félig eldönthető: létezik olyan algoritmus, amely egy következményről bebizonyítja, hogy az adott formula valóban következmény, de nem létezik olyan algoritmus, amely egy tetszőleges formuláról el tudja dönteni, hogy következmény-e (nemleges választ nem tud adni).
- Szükség van-e az összes lehetséges helyettesítés elvégzésére?

- Általánosított modus ponens: Ha $A_1, \dots, A_n, A'_1, \dots, A'_n, B$ olyan elsőrendű atomi formulák, amelyekre θ egy olyan helyettesítés, hogy $HELYETTESIT(\theta, A_i) = HELYETTESIT(\theta, A'_i)$ minden i -re, akkor

$$\frac{A'_1, \dots, A'_n, \quad A_1 \wedge \dots \wedge A_n \supset B}{HELYETTESIT(\theta, B)}$$
 - Tehát elegendő csak azokat a példányosításokat vizsgálni, amelyek a konklúzió szempontjából relevánsak.
- Az általánosított modus ponens kiemelt modus ponensnek is nevezik: átemeli az állításlogikából az elsőrendű logikába a modus ponens következtetési sémát.
 - Ennek alapján az előre felé és a hátra felé láncolás kialakítható az elsőrendű logikára is.
- Kifejleszthető a rezolúció algoritmus az elsőrendű logikára is.
- A kiemelt következtetési szabályok előnye az állításlogikára való visszavezetéssel szemben az, hogy csak azokat helyettesítéseket hajtják végre, amelyek bizonyos következtetések végrehajtását teszik lehetővé.

Egyesítés

- A kiemelt következtetések végrehajtásához olyan helyettesítéseket kell találni, amelyek a különböző formulákat látszólag azonossá teszik:
 - a folyamatot egyesítési lépésnek nevezzük.
 - Az *EGYESIT* algoritmus vesz két formulát, és visszaad egy rájuk vonatkozó egyesítést, ha létezik ilyen.
 - $EGYESIT(A, B) = \theta$, ahol
 $HELYETTESIT(\theta, A) = HELYETTESIT(\theta, B)$
- Például:
 - $EGYESIT(Ismer(Janos, x), Ismer(y, Lajos)) = \{x/Lajos, y/Janos\}$
 - De: $EGYESIT(Ismer(Janos, x), Ismer(x, Erzsebet)) = sikertelen$
 - Ilyenkor az egyesítendő formulákban át kell neveznünk a változókat.
- A legáltalánosabb (a változók értékeire legkevesebb korlátozást megadó) helyettesítésre kell törekedni.
 - $EGYESIT(Ismer(Janos, x), Ismer(y, z)): \theta = \{y/Janos, x/z\}$

- Előrefelé láncolás az elsőrendű logikában.
- Hátrafelé láncolás az elsőrendű logikában.