HÁZI FELADAT SEGÉDLET

Mesterséges Intelligencia

Tudásreprezentáció és Tervkészítés

Készítette:
Kovács Dániel László
(dkovacs@mit.bme.hu)

Budapest Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

2009.

Tartalomjegyzék

I. BEVEZETÉS **3**

II. TUDÁSREPREZENTÁCIÓ ÉS TERVKÉSZÍTÉS **4**

II.1. ELMÉLETI ALAPFOGALMAK 4
II.2. GYAKORLATI ALAPFOGALMAK 7
II.2.1. Sussman anomália 7
II.2.2. PDDL és LPG 8
II.2.3. Sussman anomália megoldása 9
II.2.4. Típusok bevezetése 23

III. ÖSSZEFOGLALÁS **28**

IV. FÜGGELÉK **29**

IV.1. EGY BONYOLULTABB SUSSMAN ANOMÁLIA ÉS MEGOLDÁSA 29
IV.2. SUSSMAN ANOMÁLIA MÁSFAJTA MEGOLDÁSA 31
IV.3. NUMERIKUS VÁLTOZÓK BEVEZETÉSE 32

V. IRODALOMJEGYZÉK **37**

I. Bevezetés

Jelen segédlet a Budapest Műszaki és Gazdaságtudományi Egyetem Méréstechnika és Információs Rendszerek tanszéke által szervezett „Mesterséges Intelligencia” c. tárgy „Tudásreprezentáció és Tervkészítés” c. házi feladata kapcsán került kidolgozásra. Célja a hallgatók elméleti és gyakorlati felkészítése, továbbá a tervkészítés témakörének általános bemutatása.

A ***tervkészítés*** nagyjából az 1960-as évek óta tekinthető aktív kutatási területnek. A kutatás célja olyan tervkészítő rendszerek kidolgozása, amelyek a valós adatok, tapasztalatok fényében képesek hatékony és megbízható cselekvéstervek előállítására.

Példának okáért közismert, hogy 1991-ben az amerikai hadsereg tervkészítő rendszereket vetett be az Öböl-háború során harci egységei vezérlésére, ellátásuk ütemezésére, stb. Ezzel a technológiával igen nagy előnyre tettek szert, hiszen a tervkészítő rendszerek temérdek olyan lehetőséget, eshetőséget is képesek voltak mérlegelni, amit a másik fél hadi-vezetői, stratégái már nem tudtak számításba venni. Ezen felül – a beérkező információk alapján – szinte azonnali döntést, stratégia-módosítást, vagy éppen új stratégiát tudtak javasolni, miközben az ellenfélnél ugyanez emberi döntéshozást, tervezést igényelt, ami jóval tovább tart. Tehát a tervkészítési eszközök használatának köszönhetően az amerikai haderők (már a tényleges harc megkezdése előtt) nem csak minőség, hanem gyorsaság tekintetében is előnyre tettek szert (az egyéb technológiai előnyökről nem is beszélve). A modern haditechnika alkalmazása így válhatott teljessé.

Természetesen a tervkészítési módszereknek számos békésebb alkalmazása is van, ámde ezek ismertetése túlmutat e segédlet keretein. A segédletben a házi feladat elkészítéséhez szükséges főbb elméleti és gyakorlati alapfogalmakat vezetjük be.

A segédlet elkészítéséhez nyújtott segítségéért kiemelt köszönet illeti Dr. Kovács Andrást (akovacs@sztaki.hu), aki számos hasznos javaslattal, tapasztalattal, oktatási segédanyaggal, és gyakorlati eszközzel támogatta e segédlet létrejöttét.

Kovács Dániel László
BME-VIK-MIT

II. Tudásreprezentáció és Tervkészítés

Ebben a fejezetben bevezetjük a tervkészítés és tudásreprezentáció alapvető elméleti és gyakorlati alapfogalmait, továbbá egy mintapéldát is megoldunk.

II.1. Elméleti alapfogalmak

A tervkészítést végző autonóm rendszert tekintsük tervkészítési környezetbe ágyazott ***ágens**nek*. A tervkészítő ágensnek nem mindig elegendő pusztán csak az aktuális megfigyelést, vagy éppen saját aktuális benső állapotát figyelembe vennie ahhoz, hogy a megfelelő döntést meghozhassa (lásd. reaktív ágensek). A megfelelő cselekvés kiválasztása legtöbbször bizonyos szintű „előrelátást" kíván, aminek során az ágens különböző cselekvés-sorozatokat mérlegel annak érdekében, hogy kiválaszthassa közülük a céljainak legmegfelelőbbet. Ezt a folyamatot nevezik ***tervkészítés**nek*.

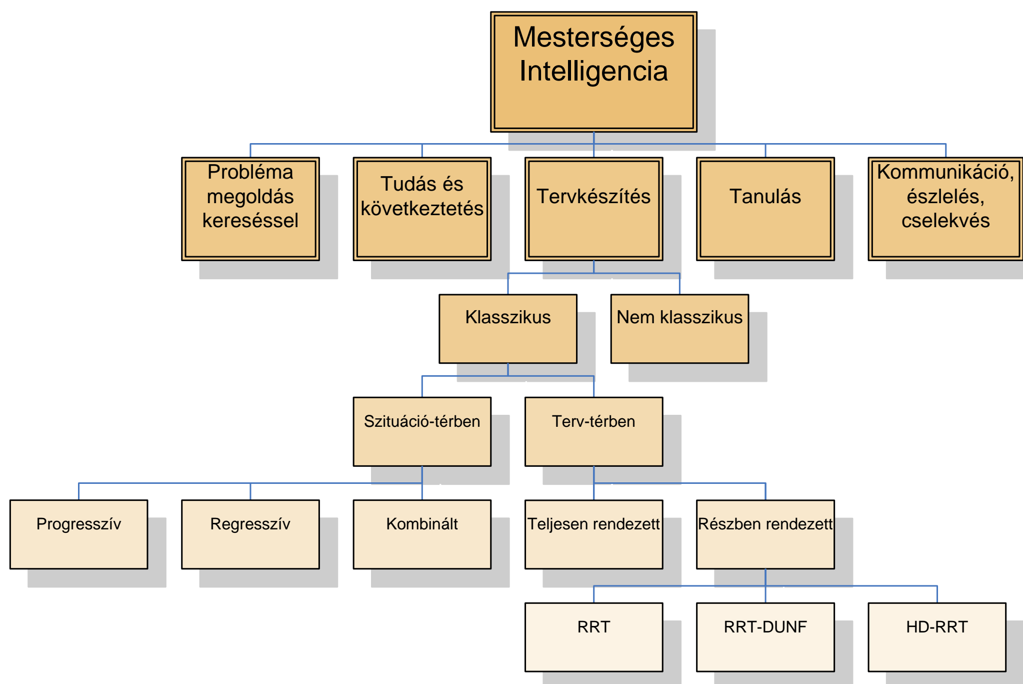
Problémá**nak* nevezik az ágens-környezet *kiinduló állapotának*, *cél-állapotainak* és lehetséges *cselekvéseinek* együttesét. Ekkor a ***probléma megoldása a cselekvések egy olyan sorozata, melyek végrehajtása a kiinduló állapotból a cél-állapotok valamelyikébe vezet. A cél-állapot megadása lehet explicit, vagy implicit. Utóbbi esetben többnyire valamiféle *cél-függvény* segítségével dönthetjük el, hogy az adott állapot része-e a cél-állapotok halmazának. Cél-állapotnak tekinthetjük például ekkor mindazon állapotokat, melyekhez a cél-függvény – a hozzájuk vezető cselekvés-sorozat ismeretében – egy-egy adott értéket rendel (lehet ez akár a függvény maximuma, minimuma, vagy bármely más értéke). A problémákat a következőképp osztályozhatjuk:

- *Egyállapotú problémák* azok, amelyek olyan teljesen hozzáférhető (kvázi determinisztikus) környezetet írnak le, ahol az egyes cselekvések kimenetele az ágens számára teljes egészében ismert.
- *Többállapotú problémák* azok, amelyek olyan, nem teljesen hozzáférhető (kvázi nem-determinisztikus) környezetet írnak le, ahol az egyes cselekvések lehetséges kimenetelei az ágens számára teljes egészében ismertek.
- *Eshetőségi problémák* azok, amelyek olyan, nem teljesen hozzáférhető (kvázi nem-determinisztikus) környezetet írnak le, ahol az egyes cselekvések lehetséges kimenetelei az ágens számára csak részben ismertek.
- *Felderíthetőségi problémák* azok, amelyek olyan, nem teljesen hozzáférhető (kvázi nem-determinisztikus) környezetet írnak le, ahol az egyes cselekvések lehetséges kimenetelei az ágens számára (kezdetben) egyáltalán nem ismertek.

A problémák definíciója jól láthatóan az ágens szemszögéből, a környezet viszonylatában történt. Ez nem jelent megkötést a problémák körére vonatkozólag, mivel az ágens, illetve a környezet kellő általánosítása esetén az ágens-környezet által reprezentált probléma ekvivalens lehet bármely általános értelemben vett problémával, amellyel egy autonóm probléma-megoldó rendszer szembesülhet. E problémák megoldására szolgálnak a tervek.

Terv*nek* nevezzük *lépések* egy halmazát és a rajtuk értelmezett *kényszerek* és *relációk* összességét. A tervnek ez a – már-már megfoghatatlanul általános – definíciója nem véletlen, ugyanis a különböző tervkészítő módszerek más-más módon definiálják és reprezentálják mind a lépéseket, mind a kényszereket, mind pedig a relációkat, miközben gyakorlatilag mind a fenti, általános definíció speciális esetei.

Az előbb bevezetett fogalmak lényegében mind-mind a ***klasszikus*** (avagy más néven determinisztikus) ***tervkészítés*** témaköréből erednek. Ennek felépítése és elhelyezkedése a Mesterséges Intelligencia (MI) tárgyterületén belül a következő:



II-1. ábra: a Klasszikus tervkészítés, mint az MI egy részterülete

A II-1. ábra szerint az MI öt részre tagolható, amin belül a „Tervkészítés" egy egészen különálló fejezetet képez. Ezen belül beszélhetünk ún. „Klasszikus", és „Nem klasszikus" tervkészítésről. Az előbbi csak azokkal az esetekkel foglalkozik, ahol a tervkészítő ágens környezete statikus (azaz csak az ágens idézhet elő változást), determinisztikus, és teljesen hozzáférhető, míg az utóbbi foglalkozik minden egyéb esettel (pl. nem teljesen hozzáférhető, dinamikus, sztochasztikus környezetekkel). Mivel a segédlethez kapcsolódó házi feladat során kizárólag klasszikus tervkészítéssel kívánunk foglalkozni, ezért most ennek a résznek a felosztását vizsgáljuk tovább.

A klasszikus tervkészítés két további részre osztható: ***szituáció-térben***, és ***terv-térben történő keresés***re. Az előbbi lényegében a „Probléma megoldás kereséssel” részben tárgyalt klasszikus kereső algoritmusokat fedi, ahol a keresési tér állapotai az ágens-környezet állapotainak felelnek meg, míg az operátorok az ágens cselekvései. Attól függően, hogy a kezdőállapotból a célállapot felé, vagy fordítva, esetleg mindkét irányban egyszerre haladunk, beszélhetünk ***progresszív***, ***regresszív***, és ***kombinált*** tervkészítésről (a szituáció-térben).

A terv-térben történő tervkészítés során a keresési tér állapotai tervek, míg az operátorok e terveket finomítják, módosítják. Ennek a szemléletváltásnak előnye, hogy jelentős mértékben csökkenti a tervkészítés komplexitását, mivel amíg a szituáció-térben az operátorok (azaz az állapot-változásért felelős ágens-cselekvések) száma tetszőlegesen nagy lehet, s így keresési tér elágazási tényezője is tetszőlegesen nagy, addig a terv-térben csak konstans számú operátor áll rendelkezésünkre, s így az elágazási tényező is konstans korlátos. Ez jelentős hatékonyságnövekedést eredményez általában.

A terv-térben kereshetünk ***teljesen***, és ***részben rendezett*** tervek közt. Az előbbi esetben a lépések sorrendje fix, míg az utóbbi esetben előfordulhatnak olyan lépések, melyek sorrendje nem kötött (és végül majd csak a terv végrehajtásakor dől el). Ennek több előnye is van (pl. a párhuzamos végrehajtás). A fentiekből kifolyólag a jelen segédlethez kapcsolódó házi során főként részben rendezett tervkészítéssel foglalkozunk.

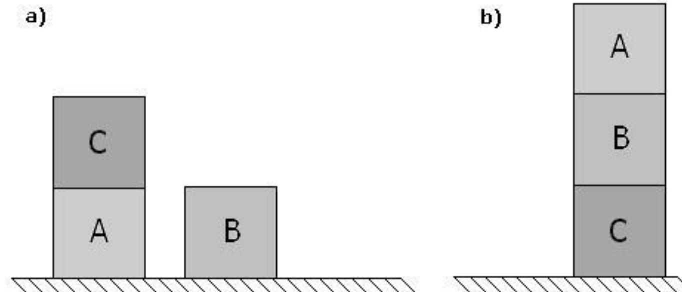
A részben rendezett tervkészítés három fő részre (algoritmusra, módszerre) tagolható: ***RRT*** (R*észben* R*endezett* T*ervkészítés*), ***RRT-DUNF*** (R*észben* R*endezett* T*ervkészítés* D*iszjunkcióval*, U*niverzális kvantorokkal*, N*egálással és* F*eltételes következményekkel*), és ***HD-RRT*** (H*ierarchikus* D*ekompozíciós* R*észben* R*endezett* T*ervkészítés*). Mindhárom algoritmus részletes leírása megtalálható a függelékben.

II.2. Gyakorlati alapfogalmak

Az elméleti bevezetőben láthattuk, hogy a jelen segédlethez kapcsolódó házi feladat során klasszikus, azon belül is kizárólag részben rendezett tervkészítéssel foglalkozunk. Ebben a fejezetben tehát ennek gyakorlati megvalósítása kerül terítékre. Megismerkedünk a ***PDDL*** (P*lanning* D*omain* D*efinition* L*anguage*) nyelvvel (McDermott és társai, 1998), amely a de facto szabvány a klasszikus tervkészítési problémák leírására. E mellett megismerkedünk egy könnyen kezelhető, RRT-alapú, ***LPG*** (L*ocal search for* P*lanning* G*raphs*) nevezetű tervkészítő alkalmazással is (Gerevini és Serina, 2002)[1], amit – amolyan fekete dobozként – PDDL-ben leírt problémák megoldására fogunk használni. A PDDL lényegét leginkább egy példán keresztül érthetjük meg (lásd. II.2.1).

II.2.1. Sussman anomália

A probléma a következő: *adott egy asztal, és rajta kockák. Célunk, hogy a II-2. ábra (a) részén látható kiindulási állapotból az ábra (b) részén látható cél-állapotba jussunk pusztán csak a kockák rakosgatásával. A mozgatott kockán más kocka nem foglalhat helyet, továbbá – az asztalt leszámítva – csak olyan kockára tehetjük, amelyiken még nincsen kocka.*



II-2. ábra: Sussman anomália: (a) kezdőállapot, (b) célállapot

A probléma ***Sussman anomália*** (Sussman, 1973) néven vált ismertté, mivel egy időben (mikor a tervkészítés még igencsak gyerekcipőben járt) az akkori tervkészítő módszerek nemigen tudtak rá megoldást adni. Ennek oka, hogy a rész-célok („A van B-n” és „B van C-n”) összefüggnek egymással[2]. Természetesen a mai tervkészítő alkalmazások, melyek alapja a részben rendezett tervkészítés (RRT), már könnyedén megoldják az ilyen, és ehhez hasonló, akár jóval összetettebb problémákat. Próbáljuk meg tehát mi is – egy alkalmasan választott tervkészítő eszköz segítségével – megoldani e problémát!

---

[1] Az alkalmazás ingyenesen letölthető a következő URL-ről: http://zeus.ing.unibs.it/lpg

[2] Amellett, hogy az „*A-t B-re rakom*”, és a „*B-t C-re rakom*” cselekvések nem hajthatók végre egymástól függetlenül (pl. egyszerre), ráadásul még az is nehezíti a helyzetet, hogy C rajta van A-n, miközben A nincsen B-n, aminek C-n kellene lennie… Ez utóbbi „körkörösség” C asztalra helyezésével oldható fel.

II.2.2. PDDL és LPG

1. Első lépésben válasszunk egy **alkalmas tervkészítő** szoftvert, amely képes PDDL nyelven leírt problémák megoldására.
2. **Reprezentáljuk a** fenti **problémát** PDDL nyelven, és oldjuk meg a választott szoftver segítségével.

Mindez igen egyszerűnek tűnik, azonban a látszat csal. Ha van is alkalmas tervkészítő eszköz a birtokunkban, a probléma leírása még akkor is igen sok megfontolást és körültekintést igényel. Ráadásul a tervkészítő szoftver kiválasztása sem egyszerű feladat, hiszen ahány szoftver (és verzió), annyi különböző képesség és jellemző.

A házi során tehát az LPG nevezetű szoftvert fogjuk használni, mint olyat, amely a leginkább igazodik a feladat célkitűzéseihez. Természetesen sok egyéb tervkészítő alkalmazás is rendelkezésre áll, számunkra azonban elegendő a PDDL 2.1 kompatibilitás. Ezt az LPG-TD 1.0 biztosítja.

A PDDL nyelv verziószáma tehát igen fontos tényező, mivel a PDDL is folyamatosan fejlődik, módosul, bővül (a tervkészítő közösség igényeinek megfelelően). Hat főbb verziója ismeretes: 1.0, 1.7, 2.1, 2.2, 3.0, és 3.1. Mindegyik verziónak megvan a maga története, de főként arról ismertek, hogy rendre a 2 évente megrendezésre kerülő ***Nemzetközi Tervkészítési Verseny (International Planning Competition, IPC)*** hivatalos nyelvét képezik. Magyarán, a versenyen a nemzetközi tervkészítő közösség mindig az éppen aktuális verziószámú PDDL-ben leírt problémákon mérheti össze az erejét.

1998-ban az 1.0-ás (McDermott és társai, 1998), 2000-ben az 1.7-es (McDermott, 2000), 2002-ben a 2.1-es (Fox és Long, 2003), 2004-ben a 2.2-es (Edelkamp és Hoffmann, 2003), 2006-ban a 3.0-ás (Gerevini és Long, 2005), 2008-ban pedig a 3.1-es PDDL (Helmert, 2008) volt a verseny hivatalos nyelve. A 3.0-ás és PDDL temérdek új elemet tartalmaz elődeihez képest (aminek csak kis kiegészítése a 3.1-es). Ezek bemutatása azonban sajnos messze túlmutat e segédlet keretein. A házival kapcsolatban számunkra elég, ha csak a 2.1-es változat fontosabb elemeivel foglalkozunk. A PDDL leírásnak (a 2.1-es változat óta) három szintje ismert:

I. **STRIPS-ES**
II. **NUMERIKUS**
III. **TEMPORÁLIS**

Az ***I. szint*** a problémák elsőrendű logikai leírását biztosítja (STRIPS-hez hasonló módon); a ***II. szint*** már nem csak logikai, hanem numerikus változókat és kényszereket is megenged, továbbá jósági kritériumok megfogalmazására is lehetőséget ad; a ***III. szint*** pedig már időzítéseket is tartalmaz (ezzel most nem foglalkozunk).

A PDDL 2.1 első egy-két szintje bőven elég számunkra ahhoz, hogy a házi feladatot megoldjuk. A konkrétumok talajára lépve térjünk is rá a Sussman anomália megoldására!

II.2.3. Sussman anomália megoldása

Most, miután a II.2.2-es szakaszban foglaltaknak megfelelően kiválasztottuk, hogy a PDDL melyik verzióját, illetve ahhoz mérten mely tervkészítő eszközt fogjuk használni, rátérhetünk a II.2.1-es szakaszban informálisan vázolt problémára.

Ahhoz, hogy a problémával bármit is kezdeni lehessen, először is le kell „fordítani" a tervkészítő eszköz, nevezetesen az LPG számára. Magyarán meg kell adnunk a probléma informális leírásának **formális reprezentációját** PDDL-ben. Ehhez először is azt kell tudnunk, hogy a PDDL **nem** egy **domain-specifikus** leírónyelv. Ez azt jelenti, hogy különböző ágens-környezetek különböző problémáinak leírására alkalmas – általános.

Ebből kifolyólag különválik benne a tervkészítési környezet (az ágenst is beleértve), illetve a megoldandó konkrét probléma (környezeti állapot) leírása. Ennek a modularitásnak köszönhetően adott tervkészítési környezethez különálló módon konstruálhatunk különböző problémákat anélkül, hogy a környezet általános leírásán bármit is változtatni kellene.

Első lépésben tehát dekomponálnunk kell a II.2.1-es szakaszban vázolt problémát egy általános, probléma-független részre, és egy speciális, éppen aktuális esetet leíró részre. Az előbbit ***domain leírásnak***, míg az utóbbit ***probléma leírásnak*** nevezik.

Domain-leírás elkészítése

Mi a Sussman anomália probléma-független része? – merül fel a kérdés. A válasz nem egyértelmű – értelmezés kérdése. A probléma formalizálása nem egy mechanikus lépés. Nem adható rá általános algoritmus (egyelőre), csak néhány irányelv, melyek segítségével szisztematikusan, az intuíciónkra hagyatkozva közelíthetünk a megoldáshoz.

Először is célszerű kiemelnünk a főbb fogalmakat (többnyire főneveket, igéket) az informális leírásból[3]. Miről is van szó? Adottak kockák, és adott egy asztal. A kockák lehetnek egymáson, és lehetnek az asztalon is, de valamelyiken biztosan. A kockákat – adott feltételek mellett – mozgatni lehet. Egyelőre ennyi.

Se a kockáknak, se az asztalnak nincs különösebb ismérve. Tehát, mint objektumoknak, nincsenek saját, különálló, egyedi jellemzőik, attribútumaik. Viszont adott közöttük egy reláció: a „*rajta*". Ez is tekinthető tulajdonságnak. Ezt kell tehát első nekifutásra megragadnunk.

Hívjuk segítségül az elsőrendű logikát, avagy más néven predikatív kalkulust! Vezessünk be egy alkalmas **predikátum**ot arra, hogy reprezentálni tudjuk a „rajta" tulajdonságot/relációt! Legyen ez mondjuk az „`on/2`", ahol a 2-es szám a predikátum aritására, avagy argumentumainak számára vonatkozik (hiszen 2 objektum kapcsolatát kell leírnunk általa).

[3] Hasonlóan az UML (Unified Modelling Language) fél-formális módszertanához.

Tehát egyetlen pedikátumunk „on(?A,?B)" alakban adható meg általánosságban, ahol „?A" és „?B" logikai változókat jelöl, melyek az (nyilván logikai értelemben vett) Univerzum tetszőleges atomjával, avagy objektumával egyesíthetők. Egyelőre. A későbbiekben ugyanis kiköthetjük az említett változók típusát (hogy csak az adott típusnak megfelelően vehessék fel értéküket). Egyelőre azonban nincs szükségünk ilyesfajta megkötésekre, ilyesfajta **típusosság**ra.

Objektumainknak, változóinknak tehát egyelőre nincs típusa, mivel – szerencsére – nincs rá szükség ahhoz, hogy reprezentálni tudjuk a problémát. Minden objektum (az asztalt is beleértve) egyetlen osztályhoz tartozik, amit egyetlen tulajdonság jellemez csupán: az „on/2". E tulajdonság adja meg tehát, hogy egy teszőleges objektum épp melyik másik objektumon van rajta.

Az áttekinthetőség érdekében törekednünk kell az egyszerűségre. Törekednünk kell arra, hogy a probléma minél egyszerűbb leírását tudjuk megadni (nem csak az „Ockham borotvája" elvből kifolyólag, hanem praktikus okokból is, például a későbbi hibajavítás, módosíthatóság, bővíthetőség, skálázhatóság, és áttekinthetőség végett[4], stb).

Természetesen nem egyszerűsíthetjük a leírást minden határon túl, mivel az előbb-utóbb információveszteséggel járna. Ilyen értelemben tehát célszerű kompromisszumra törekednünk a reprezentált információ mennyisége, pontosabban a leírás kifejezőereje és egyszerűsége közt.

Eddit tehát egy predikátumunk van, amivel a tudást megfelelő logikai tény-kijelentések formájában tudjuk reprezentálni. PDDL-ben ennek deklarációja a következőképp néz ki:

```
(define     (domain sussman)
(:requirements :strips :equality)
(:constants       table)
(:predicates      (on ?x ?y)))
```

Tehát PDDL-ben a domain-leírás egy különálló szövegfájl adott szintaxissal (azaz nyelvtannal) és szemantikával (azaz annak megadásával, hogy melyik „mondat" mikor „igaz"). Jelen segédletben azonban ennek részleteire nem kívánunk kitérni, mivel egyrészt túlmutat a segédlet keretein, másrészt az irodalomban úgyis megtalálható.

Nézzük inkább a fenti PDDL domain-leírást! Ami elsőre szemet szúrhat, az a zárójelek tetemes száma. Ennek oka, hogy a PDDL nyelvet a LISP (LIS*t* P*rocessor*) mintájára dolgozták ki (McCarthy, 1960). Mivel a LISP egy funkcionális nyelv, ezért például a…

```
(+ a b)
```

…deklaráció jelentése „a" és „b" összege, ahol a „+" egy függvényt (funkciót, operátort) jelöl, aminek 2 bemenő paramétere/argumentuma van: „a" és „b". Természetesen a PDDL nem funkcionális, hanem **deklaratív**, nincsenek benne klasszikus értelemben vett

[4] Ahogyan az áram (pl. hallgatói áram) is mindig a gyengébb ellenállás irányába folyik… ☺

függvények, csak predikátumok. Viszont a LISP-hez hasonlóan ***prefix***, azaz mindenfajta predikátum a zárójelekkel körbezárt (és szóközökkel elválasztott) *listák* első eleme.

A fenti PDDL domain-leírás ilyen értelemben tehát egy lista, melynek első eleme egy „`define`" string. Ezt követi egy 2-elemű lista:

```
(domain sussman)
```

Mivel domain-leírásról van szó, ezért ennek első eleme a „`domain`" string, második eleme viszont tetszőleges – a domain nevét jelöli. Esetünkben ez legyen „`sussman`".

A következő elem újfent egy lista:

```
(:requirements :strips :equality)
```

A lista első eleme a „`:requirements`" string. Ez jelzi, hogy itt most a domain-leírás tervkészítőkkel szemben támasztott követelményeinek felsorolása következik. A lista többi eleme pedig az úgynevezett ***követelmény flag-ek*** halmaza.

Szerencsére esetünkben egyelőre elég a „`:strips`" és „`:equality`" követelmények megadása. Ezzel tehát azt deklaráljuk, hogy a domain összes problémája leírható egyszerű STRIPS-es formalizmus, és egyenlőségvizsgálat által. Tehát annak a tervkészítőnek, amely e domain problémáit kívánja megoldani, fel kell készülnie erre, azaz meg kell felelnie ezeknek a követelményeknek, tudnia kell olyan STRIPS-es problémákat megoldani, melyekben egyenlőségvizsgálat is előfordul.

A „`:requirements`" tehát igen fontos része a PDDL domain-leírásnak, hiszen jelzi a tervkészítők számára, hogy mire számítsanak. Amennyiben egy tervkészítő nincs felkészítve valamely követelményre, úgy hibát/figyelmeztetést dob, és legtöbbször nem ad megoldást a problémára. Szerencsére azonban esetünkben ez nemigen várható…

A domain-leírás következő eleme egy lista:

```
(:constants       table)
```

A konstansok deklarációja nem kötelező, csak akkor szükséges, ha létezik olyan objektum (ezt nevezik ***konstans***nak), amely a domain összes létező problémájában előfordul. Esetünkben van ilyen: az asztal. A domain-leírás következő sora:

```
(:predicates      (on ?x ?y))
```

A lista első eleme a „`:predicates`" string. Ez jelöli, hogy itt most a különböző predikátumok megadása következik. A lista többi eleme pedig egy-egy lista, ami rendre a különböző predikátumokat deklarálja. Esetünkben csak egy ilyen lista van, mivel csak egyetlen predikátumot kívánunk bevezetni:

```
(on ?x ?y)
```

Ez tehát az ominózus „`on`” predikátum PDDL-definíciója, ahol „`?x`” és „`?y`” a predikátum két argumentumát jelöli – két változót. A változók megnevezése tehát mindig „`?`”-jellel indul. A predikátum „olvasata” (angolul) a következő: *X on Y.* Magyarán azt jelöli/jelenti, hogy egy tetszőleges X egy tetszőleges Y-on van. A predikátumok definíciójában az argumentumok (amennyiben vannak) mindig és kötelezően változók, nem pedig konkrét objektumok.

Ezzel tehát eljutottunk a fenti domain-leírás végére, mindazonáltal még nem vagyunk kész. Valami hiányzik – valami fontos! Az informális leírásban ugyanis szerepel még a következő rész is (lásd. II.2.1): „*…pusztán csak a kockák rakosgatásával.*”

Ezek szerint tehát van egy ***cselekvés*** is, amit egyelőre még nem formalizáltunk. A PDDL természetesen erre is lehetőséget ad, hiszen mit érne a tervkészítés cselekvések nélkül?… Mielőtt azonban nekifognánk a teljes PDDL domain-leírás elkészítésének, meg kell tudnunk fogalmazni, hogy esetünkben, a Sussman anomália kapcsán, milyen cselekvések jöhetnek szóba, és mi jellemzi őket. Sőt, abban is meg kell állapodnunk, hogy általában mi jellemez egy-egy ilyen cselekvést!

A leírás szerint esetünkben csupán egyetlen cselekvésről beszélhetünk: „*áthelyezés*”. A következetesség jegyében jelöljük ezt az angol „`move`” szóval. A „`move`”-nak három ***paramétere*** legyen: MIT, HONNAN, HOVA. Azaz cselekvésünk olyan, mint valamiféle függvény – paraméterezett. Ahhoz, hogy végre lehessen hajtani, meg kell adnunk, hogy konkrétan MIT kívánunk áthelyezni, HONNAN, és végül HOVA.[5]

Ezen felül azt is definiálnunk kell, hogy mik a cselekvés ***logikai elő- és utó-feltétel****ei*. Ésszerűen végiggondolva az előfeltételek (amik tehát a cselekvés végrehajtása előtt teljesülnek[6], amiknek igaznak kell lennie ahhoz, hogy a cselekvés „működjön”, azaz végrehajtásra kerüljön, és ne maradjon hatástalan) a következők:

- A kocka, amit át akarunk helyezni, ott van, ahonnan elvesszük.
- Nincs rajta más kocka.
- A kockát nem ugyanoda kívánjuk átrakni, mint ahonnan elvesszük.
- A kockát vagy az asztalra tesszük, vagy olyan kockára, amin nincs más kocka.

Tehát, amennyiben az előfeltételek teljesülnek, a cselekvés végrehajtható. Végrehajtását követően a következő állítások kellene, hogy igazak legyenek:

- A kocka ott van, ahová raktuk.
- A kocka nincs ott, ahonnan elvettük.

[5] Természetesen elképzelhetők bemenő-paraméter nélküli cselekvések is.

[6] …itt a lényeg, hogy próbáljunk meg „deklaratívan gondolkozni”, ahogyan Szeredi Péter Tanár úr is sugallja „Deklaratív Programozás” c. tárgya során. Ne függvényekben gondolkozzunk, amiknek be- és kimenete van, ne folyamatokban, hanem kijelentésekben, igazságokban! Tehát például egy cselekvés előfeltételeinek meghatározásakor egyszerűen csak jelentsük ki, hogy mi igaz olyankor.

Körvonalaztuk tehát a „`move`” cselekvést. Megadtuk paramétereit, elő- és utófeltételeit. Bővítsük tehát ennek megfelelően eddigi domain-leírásunkat – reprezentáljuk a cselekvés(sémá)t[7] PDDL nyelven:

```
(define     (domain sussman)
(:requirements :strips :equality)
(:constants       table)
(:predicates      (on ?x ?y))

(:action move
:parameters       (?cube ?from ?to)
:precondition     (and  (on ?cube ?from)
                        (not  (exists (?x) (on ?x ?cube)))
                        (not  (= ?cube table))
                        (not  (= ?cube ?from))
                        (not  (= ?cube ?to))
                        (not  (= ?from ?to))
                        (or   (= ?to table)
                              (not  (exists (?y) (on ?y ?to)))))

:effect           (and  (on ?cube ?to)
                        (not (on ?cube ?from)))))
```

Látható tehát, hogy a „`define`”-nal kezdődő lista egy újabb elemmel bővült, ami lényegében egy újabb, viszonylag összetett lista. Ez szolgál a cselekvés(séma) leírására:

```
(:action move
:parameters       (?cube ?from ?to)
:precondition     (and  (on ?cube ?from)
                        (not  (exists (?x) (on ?x ?cube)))
                        (not  (= ?cube table))
                        (not  (= ?cube ?from))
                        (not  (= ?cube ?to))
                        (not  (= ?from ?to))
                        (or   (= ?to table)
                              (not  (exists (?y) (on ?y ?to)))))

:effect           (and  (on ?cube ?to)
                  (not  (on ?cube ?from))))
```

Tetszőleges számú ilyen lista-elem adható a domain-leíráshoz attól függően, hogy hány cselekvést kívánunk bevezetni. Esetünkben csupán csak egyetlen cselekvés jön szóba, ezért egyetlen ilyen lista-elem lesz.

A cselekvés leírás tehát egy lista, aminek első eleme az „`:action`” string. Második eleme a cselekvés megnevezése – esetünkben „`move`”. Harmadik eleme a „`:parameters`” string, amit egy – akár üres – lista követ, amely felsorolja a nevezett cselekvés bemenő paramétereit változók formájában. Esetünkben ezek a következők:

[7] Valójában csak ***cselekvés-sémáról*** van szó, hiszen kizárólag a végrehajtás során, egy adott terv részeként, a változók adott behelyettesítése mellett beszélhetünk ***konkrét*** *cselekvésről*.

```
(?cube ?from ?to)
```

A „?cube" bemenő paraméter azonosítja a mozgatni kívánt kockát, a „?from" és a „?to" pedig rendre a kiindulási, és célpozíciót. A cselekvés-leírás következő eleme a „:precondition" string, amit egy lista követ, amely a cselekvés előfeltételének megfelelő (prefix jelölésű) elsőrendű logikai állítás. Lássuk ezt részletesen:

```
(and  (on ?cube ?from)
      (not  (exists (?x) (on ?x ?cube)))
      (not  (= ?cube table))
      (not  (= ?cube ?from))
      (not  (= ?cube ?to))
      (not  (= ?from ?to))
      (or   (= ?to table)
            (not  (exists (?y) (on ?y ?to)))))
```

Ez tehát egy elsőrendű logikai állítás – éppen 7 állítás ÉS-kapcsolata. Ennek első tagja:

```
(on ?cube ?from)
```

Mivel a cselekvés végrehajtásakor a bemenő paraméterek mindenképp teljesen behelyettesítve érkeznek, ezért a „?cube" és a „?from" változók be lesznek helyettesítve – konkrét objektumok lesznek a terv végrehajtása során. Épp azok, amik a bemeneten érkeztek. Ügyeljünk tehát erre a „láncolásra"!! Az ÉS-kapcsolat következő tagja:

```
(not (exists (?x) (on ?x ?cube)))
```

Ez tehát egy *egzisztenciális*an kvantifikált logikai állítás negáltja, amely kimondja, hogy nem létezik olyan „?x", hogy (on ?x ?cube) teljesüljön. Ugyanez *univerzális* kvantorral a következő:

```
(forall (?x) (not (on ?x ?cube)))
```

Ami újdonságot jelenthet itt, az a ***kvantifikált változók*** használata. Ez ugyanis az egyetlen módja annak, hogy a bemenő változókon túl, általában az aktuális világállapotra hivatkozzunk. Itt tehát nem olyan egyszerű a helyzet, mint mondjuk Prolog-ban (Colmerauer, 1975), ahol mintaillesztés útján dől el, hogy mi a változók értéke. **PDDL-ben** vagy – a cselekvések paraméter-részében lévő – **bemenő változókat,** vagy **konkrét objektumokat,** vagy **kvantifikált logikai változókat használunk arra, hogy** a cselekvések elő- és utófeltételeiben **az aktuális világ-állapotra hivatkozzunk.**

Az ÉS-kapcsolat következő 4 tagja:

```
(not  (= ?cube table))
(not  (= ?cube ?from))
(not  (= ?cube ?to))
(not  (= ?from ?to))
```

Ezekben már jól láthatóan egyenlőség-vizsgálat is szerepel[8]. A 4 feltétel valamivel többet mond, mint amit eredetileg specifikáltunk[9] (hogy a kiindulási, és a célhely nem lehet azonos). Az előbbi 4 feltételből ugyanis az első 3 (amit ezidáig halványabban jelöltünk) csupán csak a robusztusságot szolgálja. Nincs semmi más funkciójuk. Csak azért van rájuk szükség, hogy – típusok hiányában – még véletlenül (pl. helytelenül megadott kiindulási tudásbázis esetén) se fordulhasson elő a „`move`" cselekvés értelmetlen bemenő-paraméterekkel történő meghívása/végrehajtása.

Itt jegyzem meg, hogy a jelenlegi LPG implementáció kényes az egyenlőség-vizsgálat argumentumainak sorrendjére. Példának okáért adott fentebb a következő:

```
(not  (= ?cube table))
```

Habár az egyenlőség-reláció elvben szimmetrikus, mégis amennyiben konstans objektumot (`table`) szerepltetnék a 0. helyen, és változót (`?cube`) az 1-diken, úgy az LPG hibát jelez. Erre legyünk tehát tekintettel a megoldás során…

Egyébként az egyenlőség-vizsgálat használatára hívja fel a figyelmet a domain-leírás követelmény részében az „`:equality`" flag. Igazság szerint ez a feltétel elhagyható volna, ahogyan a többi követelményt sem feltétlen szükséges megadni. Alkalmazás-függő, hogy melyik-mennyire veszi komolyan. Az LPG példának okáért a követelményektől kvázi függetlenül parse-olja fel a domain- és probléma-leírást, és ennek alapján dönti el, hogy valójában milyen követelményeknek kell megfelelnie a problémamegoldás során. Mindazonáltal a `:requirements` rész – redundáns jellege ellenére – irányadó lehet mind a tervkészítő számára, mind a forrást böngésző Olvasó számára, így helyes használatát az elkövetkezőkben (is) megköveteljük.

Az egyenlőség-vizsgálat (`=`), ÉS-kapcsolat (`and`), és negálás (`not`) mellett természetesen még más egyéb logikai operátorok is rendelkezésünkre állnak ahhoz, hogy az előbbieknél összetettebb logikai állításokat is meg tudjunk fogalmazni. Ilyen például a VAGY-kapcsolat (`or`), és az implikáció (`imply`). Amíg az előbbi legalább 2-argumentumú, addig az utóbbi már csakis csak 2-argumentumú logikai művelet.

Az ÉS-kapcsolat utolsó, 7-dik eleme a következő:

```
(or   (= ?to table)
      (not  (exists (?y) (on ?y ?to))))
```

[8] Látható, hogy ez nem numerikus egyenlőséget vizsgál, hanem logikai értelemben vett *egyesíthetőséget*.

[9] Ennek oka, hogy a specifikáció nem veszi figyelembe az implementáció sajátosságait. A gyakorlatban ez igen gyakori, és legtöbbször elkerülhetetlen. Oka, hogy a *specifikáció* per definitio magasabb absztrakciós szintű, általánosabb, mint megvalósítása, az *implementáció*. Viszont a specifikációnak ennek ellenére törekednie kell a konzisztenciára, a teljességre, és arra, hogy a benne foglalt elvek változatlanul, esetleg némi szükséges kiegészítéssel kerülhessenek megvalósításra. Ennek ellenére legtöbbször igen nehéz, sőt reménytelen verifikálni a kettő közötti megfelelést, garantálni a minőséget, stb. Főleg szabványosan! Például a Windows-os LPG egy CygWin nevezetű Linux-emulációs környezet alatt fut (lásd. http://cygwin.com/), és amennyiben a CygWin nem kezeli le valamely LPG által dobott kivételt, úgy az LPG, pontosabban a CygWin elszáll (pl. egy STATUS_ACCESS_VIOLATION hibaüzenet kíséretében).

Ez egy újabb lista – két állítás VAGY-kapcsolata. Ezek szerint VAGY az igaz, hogy a célpozíció az asztal, VAGY pedig nincs olyan „`?y`”, amely a célpozíción rajta van. Ezt akár így is írhatnánk:

```
(imply      (not (= ?to table))
            (not  (exists (?y) (on ?y ?to))))
```

Implikációként felírva valamelyest „olvashatóbb”. Ezek szerint tehát, ha nem az asztalra kívánjuk rakni a kockát, akkor azon a valamin, ahová rakjuk (ami már biztosan kocka), nem lehet semmi. Az asztalon pedig nyilván több kocka is lehet egyszerre.

Visszatérve a ***bemenő paraméterek***re: úgy érdemes elképzelni őket, mint amik mindig teljesen behelyettesítve érkeznek. Az előfeltételeket úgy, mint amik illeszkednek a világ aktuális állapotára. Az utófeltételekre a következők vonatkoznak:

Az utófeltételeket egy „`:effect`” string előzi meg a cselekvés leírásában. Ezt követi a cselekvés következményét leíró elsőrendű logikai állítás. Esetünkben, a „`move`” cselekvés kapcsán ez a következő:

```
(and (on ?cube ?to)
     (not (on ?cube ?from)))
```

FIGYELEM: a cselekvések következmény-részében kizárólag **konjunkció (ÉS-kapcsolat)** foglalhat helyet, a diszjunkciónak nincs értelme. Ennek megfelelően tehát a „`move`” cselekvés következmény-része két állítás konjunkciója. Ebből az első:

```
(on ?cube ?to)
```

A cselekvés végrehajtása után (kizárólag akkor, ha az előfeltételek teljesültek) igaz lesz tehát, hogy az átmozgatni kívánt kocka a célhelyen van. Másrészt igaz lesz az is, hogy:

```
(not (on ?cube ?from))
```

Azaz nem lesz igaz, hogy az átmozgatni kívánt kocka a kiindulási pozícióban van. Ezekre az utófeltételekre tehát úgy kell tekintenünk, mint amik módosítják a világ állapotát. Alapértelmezésben ugyanis a ***zárt-világ feltételezés***sel élünk, amely kimondja, hogy amiről nem tudjuk, hogy igaz-e, azt hamisnak tekintjük. Ezek szerint a negált következmények eltávolítják a tudásbázisból a tényt (retract), míg a ponált következmények hozzáadják (assert).

Probléma-leírás elkészítése

Az előbbiekben, a Sussman anomália kapcsán áttekintettük a PDDL probléma-független, ún. domain-leírási mechanizmusának alapjait. Megadtuk a Sussman anomália probléma-független, PDDL leírását. A megfelelő szövegfájlt, amely ezt a leírást tartalmazza, nevezzük a következőképp: `hf02_pelda_output_domain.pddl`

A következőkben a Sussman anomália probléma-függő részét, az úgynevezett probléma-leírást fogjuk elkészíteni. Ez a leírás tehát az előbbi domain-leíráshoz fog igazodni, és a következő fájlnév alatt mentjük el[10]: `hf02_pelda_output_problem.pddl`

A probléma-leírás tehát, mint már említettük, igazodik a kapcsolódó domain-leíráshoz. Legegyszerűbb esetben 3 fő részből tevődik össze (a fejlécen kívül):

1. Világban lehetséges véges sok objektum felsorolása
2. Világ kiinduló-állapotában igaz tények felsorolása
3. Célállapot meghatározása

Mielőtt azonban rátérünk a PDDL-szintaxis részleteire, gondoljuk végig, hogy az előbbi, probléma-független leírásnak megfelelően mik lesznek esetünkben a probléma-függő elemek! Tegyünk – egyelőre informálisan – eleget az előbbi három kitételnek!

A II-2. ábra szerint négy objektumunk van: 3 kocka, és 1 asztal. Nevezzük ezeket rendre „`a`"-nak, „`b`"-nek, „`c`"-nek, és „`table`"-nek. Ebből a „`table`" konstans – mindig, minden probléma esetén szerepel, ezért már definiáltuk a domain-leírásban. A probléma-leírásban felesleges újra szerepltetni. Lássuk tehát, hogy mi igaz a kiinduló-állapotban!

A II-2. ábra szerint „`c`" az „`a`"-n van, „`a`" viszont az asztalon, azaz a „`table`"-en. Emellett még az is igaz, hogy „`b`" a „`table`"-en van. Összességében tehát három tény lesz igaz a Sussman anomália kiinduló-állapotában:

```
(on c a)
(on a table)
(on b table)
```

FIGYELEM: a STRIPS kapcsán bevezetett zárt világ feltételezéssel élve csupán csak az igaz tények felsorolása szükséges – ami nincs megadva, azt hamisnak tekintjük. Az LPG ennek megfelelően hibát jelez, ha negált tényállítást talál a kezdetben igaz tények közt.

Végül hasonlóképp határozzuk meg azt, hogy minek kell teljesülnie a célállapotban. Itt nyilván már nem csak tények konjunkciója szerepelhet, hanem tetszőleges elsőrendű (akár kvantifikált[11]) logikai állítás. Esetünkben ez viszonylag egyszerű: „`a`" van „`b`"-n, „`b`" van „`c`"-n, „`c`" pedig az asztalon, azaz a „`table`"-en. Ez a cél. Ennek kell teljesülnie végül. Ez PDDL-ben a következő állításnak felel meg:

```
(and  (on a b)
      (on b c)
      (on c table))
```

[10] Érdemes valamiféle elnevezési konvenciót/szisztémát követnünk ahhoz, hogy ne kuszálódjanak össze a különböző PDDL leírások.
[11] Azaz olyan, amelyben egzisztenciális és/vagy univerzális kvantor szerepel. Mindazonáltal az LPG jelen verziója sajnos nem támogatja ezt egynél több kvantifikálható objektum esetén, így használata felesleges.

Összefoglalva, a Sussman anomália PDDL-alapú probléma-leírása a következő:

```
(define (problem sussman1)
(:domain sussman)
(:objects a b c)
(:init      (on c a)
            (on a table)
            (on b table))
(:goal      (and  (on a b)
                  (on b c)
                  (on c table))))
```

A PDDL-alapú probléma-leírás tehát szintaxisában hasonló a domain-leíráshoz. Egyetlen egy LISP-es listából áll, aminek első eleme egy „`define`” string. Második eleme egy 2-elemű lista, aminek első eleme a „`problem`” string, második eleme pedig a probléma tetszőleges megnevezése. Esetünkben ez „`sussman1`”. Ezt követi a domain-hivatkozás:

```
(:domain sussman)
```

Vegyük észre, hogy itt nem fájl-szinten, hanem azonosító-szinten történik a hivatkozás (hiszen a domain-t fentebb „`sussman`”-nek neveztünk el). Ezután következik a probléma-specifikus objektumok felsorolása:

```
(:objects a b c)
```

Itt nyilván azért nem szerepel a „`table`”, mivel már a domain-leírásban definiáltuk, mint olyan konstans objektumot, amely minden Sussman domain-hez tartozó probléma esetén megjelenik. Az objektumok megadását a kezdetben igaz – **teljesen behelyettesített** (!!!) – tények felsorolása követi:

```
(:init      (on c a)
            (on a table)
            (on b table))
```

Végül pedig a célállapot meghatározása:

```
(:goal      (and  (on a b)
                  (on b c)
                  (on c table)))
```

A célállapotot egyébként nyilván nem kell teljes egészében specifikálni. Lehet ugyanis több célállapot, több olyan tény-elrendezés, több olyan világ-állapot is, amelyben teljesül a megoldástól elvárt feltétel. Elég tehát egy olyan logikai állítás megadása, amely csak és kizárólag a célállapotokban teljesül. Ezzel – félig-meddig implicite – a kívánatos világ-állapotok (ún. szituációk) halmazát definiáljuk.

A probléma-leírás végére értünk, következhet a probléma megoldása!

Probléma megoldása

Az előbbiekben megkonstruáltuk a Sussman anomália PDDL nyelvű domain- és probléma-leírását. Vizsgáljuk meg, miképpen futtathatjuk ezen választott tervkészítő alkalmazásunkat, az LPG-t![12]

Az LPG futtatására több mód is van, mi azonban ezek közül egyelőre csak azt az esetet vizsgáljuk, amikor Windows Command Prompt, azaz parancssor alól történik a futtatás. Először is lépjünk be az LPG könyvtárába (pl. `c:\lpg`). Bizonyosodjunk meg arról, hogy az előbb létrehozott `hf02_pelda_output_domain.pddl`, és `hf02_pelda_output_problem.pddl` fájlok rendelkezésre állnak (pl. magában a `c:\lpg` könyvtárban). Az LPG futtatásához ekkor a következőt írjuk be:

```
lpg-td-1.0 -o hf02_pelda_output_domain.pddl -f hf02_pelda_output_problem.pddl -quality
```

Amennyiben mindent helyesen csináltunk, úgy a futás eredményeképpen a megoldás (különböző egyéb információkkal egyetemben) kiíródik mind a képernyőre, mind egy külön fájlba. A kigenerált fájl neve: `plan_hf02_pelda_output_problem.pddl_1.SOL`

Látható, hogy a kigenerált fájl nevét az LPG automatikusan határozza meg a probléma-leírást tartalmazó PDDL-fájl neve alapján. Vegyük szemügyre a fentebb szereplő parancssori utasítást, amivel az LPG-t futásra bírtuk.

Jól láthatóan 3 opciót adtunk meg: a „`-o`” arra való, hogy megadjuk a domain-leírást tartalmazó PDDL-fájl elérését (az úgynevezett operátor-fájlt, amely tehát az operátorokat, avagy operátor-sémákat tartalmazza). A második opció a „`-f`”. Ez arra való, hogy megadjuk a probléma-leírást tartalmazó PDDL-fájl elérését (az úgynevezett tényeket (facts) tartalmazó fájlt). Végül a harmadik opció a „`-quality`”. Ennek az opciónak nincs paramétere – arra való, hogy az LPG modalitását szabályozza.

Az LPG jelen verziójának kétféle modalitása van ugyanis: „`-quality`”, és „`-speed`”. Az előbbi hatására jobb minőségű, rövidebb tervek adódnak valamivel hosszabb idő alatt, mint az utóbbi esetben, mikor is a futási idő, nem pedig a terv minősége az elsődleges szempont.[13]

A domain-leírás, probléma-leírás, és modalitás megadása szükséges ahhoz, hogy a LPG fusson. A modalitás helyett esetleg még a „`-n`” opcióval az is megadható, hogy a problémának legfeljebb hány megoldására vagyunk kíváncsiak.

Térjünk most azonban még egy kicsit vissza az előbbi futás eredményéhez. A képernyőre nagyjából a következő kerül:

[12] Csak a Windows-os esetet vizsgáljuk.
[13] Érdemes megfigyelni, hogy amíg „`-quality`” opcióval többszörös meghívásra is mindig ugyanazt a tervet adja a tervkészítő, addig „`-speed`” opcióval meghívva hívásonként más-más lehet az eredmény.

```
Parsing domain file:  domain 'SUSSMAN' defined ... done.
Parsing problem file:  problem 'SUSSMAN1' defined ... done.



Modality: Quality Planner

Number of actions             :      64
Number of conditional actions :       0
Number of facts               :      32


Analyzing Planning Problem:
      Temporal Planning Problem: NO
      Numeric Planning Problem: NO
      Problem with Timed Initial Litearals: NO
      Problem with Derived Predicates: NO

Evaluation function weights:
     Action duration 0.00; Action cost 1.00

Computing mutex... done

Preprocessing total time: 0.06 seconds

Searching ('.' = every 50 search steps):
 Restart.
.. search limit exceeded. Restart.
.. search limit exceeded. Restart.
..

Plan computed:
   Time: (ACTION) [action Duration; action Cost]
 0.0003: (MOVE C A TABLE) [D:1.0000; C:1.0000]
 1.0005: (MOVE B TABLE C) [D:1.0000; C:1.0000]
 2.0008: (MOVE A TABLE B) [D:1.0000; C:1.0000]

Solution found:
Total time:      0.78
Search time:     0.03
Actions:         3
Execution cost:  3.00
Duration:        3.000
Plan quality:    3.000
     Plan file:       plan_hf02_pelda_output_problem.pddl.SOL



Do not use option -quality for better solutions.
```

Itt először is az látszik, ahogyan az LPG beolvassa (be-parse-olja) a domain- és probléma-leírást tartalmazó fájlokat. Ezt követi a modalitás jelzése. Jelen esetben ugyebár „-quality” opcióval hívtuk meg a tervkészítőt, így a modalitás „Quality Planner”. A következő három sor a cselekvések, és a tények számát adja meg.

Elsőre meglepő lehet, hogy miért éppen 64 cselekvést említ az LPG, mikor mi csak egyetlen egyet adtunk meg. Ennek oka egyszerű: nem egy cselekvést, hanem egy *cselekvés-sémát* adtunk meg. A cselekvés-séma 3 bemenő paramétert tüntet fel. A problém-leírás szerint pedig összesen 4 különböző objektum van a világban. Ezek szerint összesen 4^3=64 különböző bemenettel hívható meg a „`move`” cselekvés.

A következő sor azt jelzi, hogy egyetlen feltételes cselekvésünk (`conditional action`) sincs. **Feltételes cselekvések** azok, melyek következmény-része feltételes. A PDDL már a kezdetek óta alkalmas ilyesfajta cselekvések leírására, mi azonban a kapcsolódó házi feladat alkalmával **nem foglalkozunk velük**, mivel nem egyeztethető össze azzal a célkitűzésünkkel, amely szerint kizárólag csak klasszikus tervkészítéssel foglalkozunk.[14]

A következő sor szerint 32 tény van. Ez nyilván nem a kiindulási tudásbázisban (vagy munkamemóriában) szereplő tények száma, ami – mint fentebb látszik – pusztán csak 3. Nem, ez itt az összes elképzelhető tény számát adja meg, amit úgy kapunk, hogy vesszük egyetlen 2-argumentumú predikátumunkat, és minden lehetséges módon behelyettesítjük: így adódik 4^2=16 darab különböző lehetséges ponált tény[15], amelynek elvben (pl. a zárt-világ feltételezés elhagyásával) még másik 16 negált párja lehet. Így összesen 16+16=32 különböző lehetséges tényről beszélhetünk.

A következő pár sor a probléma analízisét mutatja. Mivel esetünkben (egyelőre) semmi különöset sem alkalmaztunk a domain és/vagy a probléma leírásában, ezért minden szempontnál a „`NO`” szerepel.

Az LPG a keresés során használ egy úgynevezett kiértékelő függvényt (evaluation function), amely súlyozza a cselekvéseket végrehajtási idejük, és áruk alapján. Jelen esetben – alapértelmezésben – ezek a súlyozási faktorok 0 és 1 értékűek.

A következő sor szerint sikeresen kiszámításra kerültek az úgynevezett „***mutex***”-ek. A mutex-szócska az angol „mutually exclusive” szavak összetételéből adódik, és olyan cselekvésekre vonatkozik, melyek kölcsönösen kizárják egymást.

A mutex-ek alatt az előfeldolgozás ideje, a tervkészítés/keresés folyamata, és maga a megoldás, azon belül is a terv lépései láthatók. Minden lépés mellett ott szerepel, hogy melyik időpillanatban indult meg végrehajtása, mennyi ideig tartott, és mennyibe került. Alapértelmezésben (a nem temporális, STRIPS-es tervkészítésnél) minden időtartam és költség egységnyi.[16]

[14] Továbbá sajnos az LPG jelen verziója sem támogatja…

[15] Amiknek jó része nyilván abszurdum, a tervkészítő azonban „buta” módon ezt mégsem veszi észre. Hasonlóan, a cselekvések esetében is látható volt, hogy nem történt szűrés arra vonatkozólag, hogy mely paraméter-kombinációkkal nem lesz sohasem (értsd. semmilyen világállapot mellett sem) végrehajtható a cselekvés. Mi azonban tudjuk, hogy például az összes olyan bemeneti paraméter-kombináció elhagyható volna, amelyben valamely bemeneti elem legalább kétszer szerepel. Mindennek a „least commitment”, avagy a ***legkisebb megkötés elve*** az oka. Ez az elv allegorikusan itatja át a részben rendezett tervkészítést.

[16] Úgy látszik az LPG ehhez az időtartamhoz a biztonság kedvéért még hozzáad 0.2-0.3 microsec.-ot.

Végül a tervkészítés ideje, és a megoldás(ok) paraméterei látható(k). Jelen esetben egy 3 lépés hosszú tervet sikerült találni, amely 3 időegység alatt kivitelezhető, illetve költsége és minősége egyaránt 3.[17]

A futás során kigenerált fájl (esetünkben: `plan_hf02_pelda_output_problem.pddl_1.SOL`) is lényegében ugyanezeket az információkat tartalmazza, csak valamivel felületesebb formában. A lényeg maga a megoldás. Nézzük most ezt:

```
0:   (MOVE C A TABLE) [1]
1:   (MOVE B TABLE C) [1]
2:   (MOVE A TABLE B) [1]
```

Ezek szerint a 0. időpillanatban a C kockát az A-ról az ASZTAL-ra helyezzük, majd közvetlen ez után, az 1. időpillanatban a B kockát az ASZTAL-ról a C-re tesszük, és végül, a 2. időpillanatban az A kockát az ASZTAL-ról a B kockára rakjuk át. Ezzel tehát megoldottuk a Sussman anomáliát.

Nyilván a Sussman anomália esetében „ránézésre" is látszik a megoldás. Látszik tehát, hogy az LPG által visszaadott terv optimális: nincs ennél gyorsabb, hatékonyabb megoldás. Viszont ez az egyszerűség ne tévesszen meg minket – „gazdag egyszerűség"-ről van szó ugyanis! Gondoljunk csak bele: 3 kocka esetén még nyilván könnyen átlátható és megoldható a probléma. De *mi van akkor, ha több kocka van? Ha 5, 7, 9, vagy esetleg 11, sőt, még sokkal-sokkal több kocka van az asztalon?*

Szerencsére a problémák skálázása (mint már említettük) nem jár a domain-leírás megváltoztatásával. A leírás ugyanolyan egyszerű marad, mint volt. Egyedül a probléma-leírás gazdagodik újabb objektumokkal, tényekkel, és esetleg célra vonatkozó feltétellel. Ha valami okból mindez mégis komplikáltnak tűnne, ne rettenjünk vissza! Csak az első néhány alkalommal van ez így. Valójában nagyon egyszerű, intuitív, és kézenfekvő. Gyakorlatlanságunk folytán kezdetben még elveszhetünk az apró részletekben, amiket később aztán rutinszerűen elsajátítunk. „Szoknia kell a szemnek"…

Minden esetre remélhető, hogy a fentiek elolvasása után nem maradt bennünk számottevő kérdés a Sussman anomália PDDL-ben történő reprezentációjával, és annak LPG általi megoldásával kapcsolatban.

[17] A megoldás minősége és költsége az LPG sajátja – keresési heurisztikáihoz szükséges faktorok.

II.2.4. Típusok bevezetése

Ebben a szakaszban az előzőekben bevezetett példát folytatjuk, és egészítjük ki típusokkal. Az előbbiekben esetleg zavaró lehetett, hogy „egy lapon” említettünk minden objektumot. Valóban, számunkra ez még csak zavaró, de a tervkészítő számára még akár megtévesztő is lehet, ugyanis ilyenkor temérdek olyan ténnyel és cselekvéssel kell számolnia, ami esetleg értelmetlen, sohasem fog előfordulni. A tervkészítő mégis számol(hat) vele, hiszen az objektumok között semmi különbséget nem tud tenni.

Az ilyen, és ehhez hasonló esetek számának csökkentése, illetve egyéb problémák elkerülése érdekében szükséges lehet specifikusabbá tenni a PDDL leírást. Ezt eddig vagy előfeltételekbe iktatott explicit egyenlőség-vizsgálatokkal, vagy a kezdeti világállapot megfelelő megadásával oldottuk meg. A két módszer kiegészítette egymást, ám még így együtt sem voltak igazán megfelelők.

Egyrészt a cselekvések előfeltételeinek bővítése *átláthatatlanná teheti* a cselekvések leírását, hiszen a bemenő paraméterek számával négyzetesen arányos számú egyenlőség-vizsgálatra lehet szükség (nem is beszélve a különböző konstansokkal történő esetleges összehasonlításokról).

Másrészt a kiindulási tudásbázis helyes megadása *túl implicit* megkötés. Esetünkben szerencsére viszonylag kézenfekvő volt, hogy minek kell az adott probléma kapcsán kezdetben igaznak lennie, és a tervkészítő szerencsére nem is ment tévútra. Viszont egy összetettebb probléma esetén már ilyesmi is előfordulhat!

A megoldást, mint ahogyan a szakasz címe is utal rá, a típusok bevezetése jelenti. Gyorsabb, egyszerűbb, és megbízhatóbb. Mostantól tehát minden objektumnak és változónak lesz típusa. A típusra gondolhatunk úgy, mint valamiféle tulajdonságra, ami minden objektumhoz rendel egy típus-azonosítót, de úgy is, mint valamiféle kategóriára, osztályra, amelyhez az objektum tartozik. Személy szerint én inkább ezt a szemléletet tartom célravezetőbbnek. Ekkor ugyanis úgy tekintünk a (logikai értelemben vett) univerzum atomjaira, a lehetséges objektumokra, mint egy halmaz elemeire. A halmaznak pedig vannak részhalmazai: ezek a „leszármazottak” – az egyes „osztályok”.

Az osztályokat, mint kategóriákat, nyilván úgy határozzuk meg, hogy olyan elemeket tartalmazzon, amelyek azonos tulajdonságokkal rendelkeznek (még ha értékük el is tér egymástól). Ezeket az elemeket tehát képzeletünkben „körbekarikázzuk”, és osztálynak nevezzük. Az említett elemek, az objektumok ekkor az *osztály példányai*. Az osztályra pedig úgy is tekinthetünk, mint ***típus***ra.

A típusok, ahogyan az osztályok is, hierarchiát alkotnak. Adottak szülő-típusok, és gyermek-típusok. Az utóbbiak öröklik az előbbiek tulajdonságait, jellemzőit. Pontosabban (mivel PDDL-ben nem tulajdonságok által implicite, hanem explicite adjuk meg a típusokat) a gyermek-típusú objektumok szülő-típusúak is egyben.

Az ős-típus PDDL-ben az „`object`”, ami nem csak fizikai objektumokra vonatkozhat, hanem – az *Ontology Web Language*-ben (OWL, lásd. McGuinness és van Hermelen, 2004) ismeretes „Thing” ős-osztályhoz hasonlóan – bármire.[18]

Domain-leírás

Egészítsük tehát ki az előbbi domain- és probléma-leírást típusokkal! Kezdjük a domain-leírás kiegészítésével:

```
(define   (domain sussman)
(:requirements :strips :equality :typing)
(:types cube table)
(:predicates       (on ?x - cube ?y - object))

        (:action move_to_table
        :parameters   (?cube - cube ?from - object ?to - table)
        :precondition (and    (on ?cube ?from)
                              (not      (exists (?x - cube) (on ?x ?cube)))
                              (not      (= ?cube ?from))
                              (not      (= ?cube ?to))
                              (not      (= ?from ?to)))

        :effect        (and    (on ?cube ?to)
                              (not (on ?cube ?from)))
        )

        (:action move_to_cube
        :parameters   (?cube - cube ?from - object ?to - cube)
        :precondition (and    (on ?cube ?from)
                              (not      (exists (?x - cube) (on ?x ?cube)))
                              (not      (= ?cube ?from))
                              (not      (= ?cube ?to))
                              (not      (= ?from ?to))
                              (not      (exists (?y - cube) (on ?y ?to))))

        :effect        (and    (on ?cube ?to)
                              (not (on ?cube ?from)))
        )
)
```

Az újdonságokat pirossal jelöltük. Az első, ami szembetűnik, hogy a `:requirements` közt már a `:typing` flag is szerepel, jelezve a tervkészítő számára, hogy ebben a domain-ben az objektumok domain-specifikus típussal is rendelkezhetnek. Eddig mindvégig, kimondva-kimondatlanul az „`object`” őstípust tekintettük az objektumok típusának.

A másik különbség, ami az előző szakaszban bemutatott domain-leíráshoz képest feltűnik, hogy közvetlenül a követelmények deklarációja után következik a lehetséges típusok felsorolása.[19] Ezek az „`object`”-en felül definiált, egyedi típusaink:

```
(:types cube table)
```

[18] Mindazonáltal szerencsésebbnek tartom az OWL-es „Thing” megnevezést, mivel intuitívebb. Gondoljuk csak meg, mi van akkor, ha PDDL-ben reprezentálni szeretnénk pl. az iskolai végzettséget. Legyen mondjuk „általános”, „középszintű”, „felsőfokú”, és „doktori”. Ezeket – akár konstans – objektumokként kell deklarálnunk. Típusuk az „iskolai végzettség”. Viszont ez a típus alapértelmezésben leszármazottja az „`object`”-nek, s így az előbb felsorolt végzettségek is rendre mind-mind objektumok. Zavaró lehet azonban, mikor pl. a felsőfokú végzettségre, mint fogalomra, objektumként tekintünk, nemde? – Természetesen nem fizikai, hanem *logikai objektum*ot kell értenünk alatta…

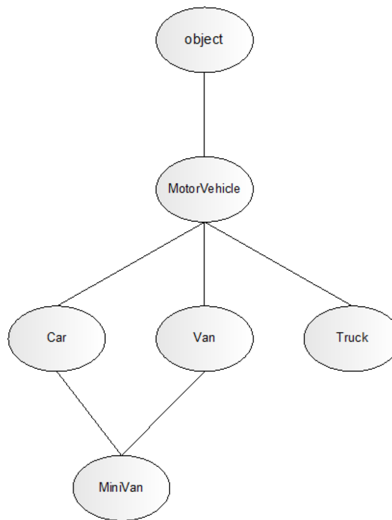
[19] Vegyük észre, hogy a fenti domain-leírásból kimaradtak a konstansok, mivel **az asztal immár egy típus**.

Az eddigiekhez mérten nyilván nem meglepő, hogy éppen erről a 2 típusról van szó. Jelentésük nyilvánvaló. Mindnyájan az „`object`” típus (és önmaguk[20]) leszármazottai. A továbbiakban különbséget csak a konstansok, a predikátumok, a cselekvések bemenő-paramétereinek, és esetleg a kvantifikált változók megadásában látunk. Itt ugyanis, ahol eddig pusztán csak egyetlen tényező (objektum, vagy változó) szerepelt, immár a típus megadása is jelen lesz.

Amennyiben egymás után több azonos típusú tényezőt is fel szeretnénk sorolni (pl. egy cselekvés-séma bemenő paramétereinél több azonos típusú változót), úgy a PDDL kényelmi szolgáltatásként lehetővé teszi, hogy ne kelljen mindannyiszor kiírnunk a típust, elég a következő formában megadnunk:

```
(... ?x1 ?x2 ... ?xN - typex ?y1 ?y2 ... ?yM - typey ...)
```

Jelenlegi domain-leírásunkban nincs típus-hierarchia, mivel eddig még nem volt rá szükség. Típusaink alapértelmezésben mind az „`object`” őstípus leszármazottai – azonos szinten helyezkednek el az egyszintes típus-hierarchiában, és mind diszjunktak.[21] Mindazonáltal egy röpke pillanatra, a példa kedvéért alakítsunk ki egy ennél összetettebb típus-hierarchiát, amin bemutathatjuk az alapelveket és szintaxist.



II-3. ábra: objektum-típusok egy hierarchiája

A II-3. ábrán jól látható a példa gyanánt felhozott típus-hierarchia. Az „`object`” őstípus mellett 5 egyénileg definiált típust vezetünk be: a „`MotorVehicle`” típus az „`object`” őstípus leszármazottja, és motoros gépjárműveket takar. A motoros gépjárműveket a példa kedvéért 3 különálló részre osztjuk: „`Car`”, amely a személygépkocsi kategóriájú járműveket takarja; „`Van`”, amely furgonokat takar; és „`Truck`”, amely a teherautókat takarja. Végül adott a „`MiniVan`” típus, amely a személyszállításra alkalmas, kisméretű furgonokat takarja, s így mind a „`Car`”, mind a „`Van`” típusnak leszármazottja.

[20] …mivel a típus-leszármazás reláció *reflexív*.
[21] PDDL-ben egy típus különböző altípusai alapértelmezésben diszjunktak, metszetük üres.

Az imént bemutatott típus-hierarchia PDDL-ben a következő:

```
(:types Car Van Truck - MotorVehicle MiniVan - (either Car Van))
```

Látható tehát, hogy ahogyan eddig a változók és objektumok típusát, úgy adjuk most meg a típusok szülő-típusát is. Látható továbbá, hogy például a „`MotorVehicle`” típusnak nincs megadva a szűlő-típusa, ami ezért alapértelmezésben „`object`”.

Az említett típusok (`object, MotorVehicle, Car, Van, Truck, MiniVan`) mind úgynevezett ***elemi típusok***. Viszont a típus-deklarációban valami más is szerepel:

```
(either Car Van)
```

Ez egy úgynevezett összetett típus. Az ***összetett típusok*** elemi típusok uniójaként adódnak, és nincs saját egyedi azonosítójuk. Egy lista jelöli őket, aminek első eleme az „`either`” string, többi – legalább egy – eleme pedig rendre az unióban szereplő elemi típusok azonosítója.

Egy összetett típust akkor tekintünk egy másik – akár elemi, akár összetett – típus *leszármazott*jának, ha a benne szereplő összes elemi típus leszármazottja a másik típusnak. Egy összetett típust akkor tekintünk egy másik – akár elemi, akár összetett – típus *felmenő*jének, ha a másik típus az összetett típusban szereplő valamely elemi típus(ok) leszármazottja.

FIGYELEM: ha a domain-leírásban egy konstansnak, egy bemenő paraméter-változónak, vagy egy kvantifikált változónak nincs külön feltüntetve a típusa, úgy alapértelmezésben „`object`”.

Ezzel tehát megadtuk a típusok szintaktikáját és szemantikáját, viszont nem indokoltuk meg a hierarchia szükségességét. *Mi értelme van a típus-hierarchia bevezetésének, ha így megbonyolítja a szintaxist?* – merülhet fel a kérdés.

A válasz elég összetett, ezért egyelőre legyen számunkra elég annyi, hogy típus-hierarchia nélkül bizonyos esetekben kénytelenek lennénk a predikátumok, illetve cselekvések számát jelentősen megnövelni. Tegyük fel ugyanis, hogy adott valahány típusunk, amik nincsenek hierarchikus viszonyban egymással. Minden objektumnak van egy ilyen típusa. Ekkor ahhoz, hogy egy predikátum, vagy egy cselekvés több különböző típusú objektumra is „működjön”, kénytelenek lennénk típusok szerint többszörözni…

Probléma-leírás

Az előzőekben bemutattuk a típusok fogalmát, és azt, hogy miként vezethetjük be őket a domain-leírásba. A probléma-leírásban egyedül az objektumok[22], és a `:goal` részben szereplő, esetleges kvantifikált változók kapcsán szükséges bevezetni őket.

[22] Sajnos az LPG jelen verziója nem támogatja az objektumok összetett típusának megadását. Ez azonban nem nagy akadály, hiszen az összetett típus megfelelő altípusának bevezetésével mindez kiváltható.

```
(define (problem sussman1)
(:domain sussman)
(:objects a b c - cube t - table)
(:init
         (on c a)
         (on a t)
         (on b t))
(:goal   (and    (on a b)
                 (on b c)
                 (on c t)))
)
```

Ezek után a tervkészítő a tervkészítés során már nem fog olyan objektumokkal kísérletezni pl. a cselekvések bemenő paramétereiben, melyeknek nem megfelelő a típusa. Tehát végső soron a típusok bevezetése egyfajta „gyorsító hatású” kényszerűség.

Probléma-megoldás

A típusok bevezetése lényegében azonban nem változtat a probléma megoldásán:[23]

```
0:   (MOVE_TO_TABLE C A) [1]
1:   (MOVE_TO_CUBE B T C) [1]
2:   (MOVE_TO_CUBE A T B) [1]
```

[23] Ha mindent alaposan végiggondoltunk, akkor ennek elvben így kell lennie. Ha a típusok bevezetése mégis szűkíti a megoldások körét, úgy vagy a típusok bevezetése helytelen, vagy a predikátumok, cselekvések értelmezése/szemantikája nem stimmel.

III. Összefoglalás

Jelen anyagban elsősorban a BME Villamosmérnöki és Informatikai Karának (VIK) Méréstechnika és Információs rendszerek Tanszékén (MIT) oktatott „Mesterséges Intelligencia” c. tárgy „Tudásreprezentáció és Tervkészítés” c. házi feladatához kívántunk segédletet nyújtani.

Ennek során megismerkedtünk a tervkészítés elméleti és gyakorlati alapfogalmaival. A **PDDL** (**P**lanning **D**omain **D**efinition **L**anguage) nyelv lényegi elemeivel (a 2.1-es verzióig bezárólag), és filozófiájával, továbbá az LPG (Local search for Planning Graphs) tervkészítő alkalmazással (egész pontosan a jelenleg legfrissebb, „1.0-td” verzióval).

A megismert fogalmak és eszközök segítségével megoldottunk egy egyszerű mintapéldát, aminek során mélyebb betekintést nyerhettünk a tervkészítés elméletébe és gyakorlatába, annak előnyeivel és buktatóival együtt. További elméleti és történeti útmutatót az átfogó irodalomjegyzék biztosít.

Mindvégig kizárólag determinisztikus, részben rendezett tervkészítéssel foglalkozunk, habár a megoldott mintapélda nem támaszkodik túlságosan a tervkészítő algoritmus mikéntjére. A tervkészítő alkalmazást amolyan „fekete dobozként” használjuk megfelelően reprezentált tervkészítési problémák megoldására. Végső soron tehát a **problémák reprezentációján**, nem pedig megoldási módján van a hangsúly.

Az anyag elsajátításával tehát viszonylag aktuális ismereteket szerezhetünk az MI tervkészítés témaköréből. Némi gyakorlatra is szert tehetünk a példák implementációja során. Mindazonáltal az élvonal, a jelen kutatás, és a gyakorlat ennél is jóval előrébb tart. Példának okáért, érdemben nem is esik szó a nem klasszikus tervkészítésről, ahol a környezet nem feltétlen determinált (Younes és Littman, 2004), illetve a multi-ágens rendszerekben történő tervkészítésről (ahol az egyes ágensek nem az egyedüli tervkészítők, s így egymás lehetőségeit és céljait is célszerű mérlegelniük a hatékony működéshez). Ilyen értelemben tehát a jelen anyag csupán kiindulópontját képezheti az „MI tervkészítés” témakör teljesebb megismerésének és elsajátításának, esetleges további kutatásának.

IV. Függelék

A függelékben foglaltak ismerete nem előfeltétele a házi feladat teljesítésének, sokkal inkább elméleti és gyakorlati adalék az érdeklődőbb, szorgosabb hallgatók számára.

IV.1. Egy bonyolultabb Sussman anomália és megoldása

Ebben a szakaszban a szemléletesség kedvéért a II.2.1. szakaszban bevezetett Sussman anomália PDDL-alapú (első körös) domain-leírásához mellékelünk egy valamivel bonyolultabb probléma-leírást, illetve annak megoldását.

Bonyolultabb probléma-leírás

```
(define (problem sussman1_2)
(:domain sussman)
(:objects a b c d e f g h i j k)
(:init  (on a table)
        (on b table)
        (on d table)
        (on i table)
        (on j a)
        (on c b)
        (on h d)
        (on g i)
        (on e j)
        (on k c)
        (on f k))
(:goal  (and    (on a b)
                (on b c)
                (on c d)
                (on d e)
                (on e f)
                (on f g)
                (on g h)
                (on h i)
                (on i j)
                (on j k)
                (on k table))))
```

Bonyolultabb probléma megoldása

```
0:   (MOVE E J G) [1]
0:   (MOVE F K TABLE) [1]
1:   (MOVE K C TABLE) [1]
1:   (MOVE E G TABLE) [1]
2:   (MOVE G I E) [1]
2:   (MOVE J A K) [1]
2:   (MOVE C B TABLE) [1]
3:   (MOVE I TABLE J) [1]
3:   (MOVE F TABLE B) [1]
4:   (MOVE H D I) [1]
5:   (MOVE G E H) [1]
6:   (MOVE F B G) [1]
7:   (MOVE E TABLE F) [1]
8:   (MOVE D TABLE E) [1]
9:   (MOVE C TABLE D) [1]
10:   (MOVE B TABLE C) [1]
11:   (MOVE A TABLE B) [1]
```

Érdemes felfigyelni arra, hogy bizonyos időpillanatokban több cselekvés végrehajtása is megkezdhető. Ennek oka, hogy a részben rendezett tervkészítő számára nem írtuk elő, hogy egyszerre csak és kizárólag egyetlen mozgat cselekvést szabad végrehajtani (pl. egy `handfree/0` predikátum bevezetésével). Így tehát azon cselekvések, melyek sorrendje nem kötött egymáshoz képest, egyszerre kerül(het)nek végrehajtásra. …ha nem hiszi, hogy jó a terv, ellenőrizze manuálisan!

IV.2. Sussman anomália másfajta megoldása

Alább a Sussman anomália II.2.3-as szakaszban bemutatott PDDL-alapú reprezentációjától eltérő, „egyszerűbb" (második körös) domain- és probléma-leírását mellékeljük, amiből bár kimaradnak a kvantorok, viszont több a cselekvés és predikátum.

Másfajta domain-leírás

```
(define  (domain sussman2)
(:requirements :strips :equality)
(:constants      table)
(:predicates     (on ?x ?y)
                 (free ?x))

(:action move_to_cube
:parameters      (?cube ?from ?to)
:precondition    (and     (on ?cube ?from)
                          (free ?cube)
                          (not     (= ?cube table))
                          (not     (= ?cube ?from))
                          (not     (= ?cube ?to))
                          (not     (= ?from ?to))
                          (not     (= ?to table))
                          (free ?to))
:effect          (and     (on ?cube ?to)
                          (free ?from)
                          (not (on ?cube ?from))
                          (not (free ?to)))
)

(:action move_to_table
:parameters      (?cube ?from)
:precondition    (and     (on ?cube ?from)
                          (free ?cube)
                          (not     (= ?cube table))
                          (not     (= ?cube ?from))
                          (not     (= ?from table)))
:effect          (and     (on ?cube table)
                          (free ?from)
                          (not (on ?cube ?from)))
)
)
```

Másfajta probléma-leírás

```
(define (problem sussman2_1)
(:domain sussman2)
(:objects a b c)
(:init
         (on c a)
         (on a table)
         (on b table)
         (free b)
         (free c))
(:goal (and (on a b)
                 (on b c)
                 (on c table)))
)
```

A probléma megoldása pedig nyilván nem változik, hiszen annak ellenére, hogy *más módon* reprezentáltuk, végső soron még *ugyanazt* reprezentáltuk ekvivalens módon.[24]

[24] …bár ennek egzakt formális bizonyítására most itt nem vállalkoznék – elég, ha a józan ész látja, miről van szó (és nem mellesleg valóban ugyanazok a megoldások jönnek ki (amiket ugyanúgy interpretálhatunk)).

IV.3. Numerikus változók bevezetése

A Sussman anomália eddigiekben látott PDDL reprezentációja színtisztán logikai leírásra szorítkozott. Mindazonáltal a valóságban egy probléma ennél nyilvánvalóan bonyolultabb is lehet. Egy kockáknak súlya van, mozgatásuk energia fogyasztással jár, stb. Ennek megfelelően egészítsük most ki a II.2.4-es szakaszban látott, típusos PDDL leírásunkat – tegyük még valósághűbbé! Ehhez először is a probléma informális specifikációját egészítsük ki a következőkkel.

„A kockáknak súlya van. A kockák mozgatása a kockák súlyával egyenesen arányos mennyiségű energiát fogyaszt. A kockákat mozgató robot energiája korlátos.”

Ezek szerint nem csak új fogalmak bevezetésére lesz szükség (pl. energia, súly), hanem numerikus korlátok (pl. csak olyan mozgatás hajtható végre, ami nem igényel több energiát annál, mint ami rendelkezésünkre áll) betartása is.

Domain-leírás

Lássuk először is egy olyan PDDL domain-leírást, amely már a fentebb felsorolt szempontokat is magába foglalja:

```
(define (domain sussman)
        (:requirements :strips :equality :typing :fluents)
        (:types cube table)
        (:predicates
                (on ?x - cube ?y - object))
        (:functions
                (weight ?x - cube)
                (energy-left))

        (:action move_to_table
        :parameters     (?cube - cube ?from - object ?to - table)
        :precondition   (and    (on ?cube ?from)
                                (not    (exists (?x - cube) (on ?x ?cube)))
                                (not    (= ?cube ?from))
                                (not    (= ?cube ?to))
                                (not    (= ?from ?to))
                                (>= (energy-left) (weight ?cube)))
        :effect         (and    (on ?cube ?to)
                                (not (on ?cube ?from))
                                (decrease (energy-left) (weight ?cube)))
        )

        (:action move_to_cube
        :parameters     (?cube - cube ?from - object ?to - cube)
        :precondition   (and    (on ?cube ?from)
                                (not    (exists (?x - cube) (on ?x ?cube)))
                                (not    (= ?cube ?from))
                                (not    (= ?cube ?to))
                                (not    (= ?from ?to))
                                (not    (exists (?y - cube) (on ?y ?to)))
                                (>= (energy-left) (weight ?cube)))
        :effect         (and    (on ?cube ?to)
                                (not (on ?cube ?from))
                                (decrease (energy-left) (weight ?cube)))
        )

)
```

A bemutatott domain-leírásban pirossal külön kiemeltük, hogy az előző leíráshoz (lásd. II.2.4) képest mik a fő különbségek. Vegyük most ezeket sorra!

Először is látható, hogy a leírás elején, a követelményeket felsoroló `:requirements` részben immár egy újabb, `:fluents` flag-et is szerepeltetünk azért, hogy jelezzük a tervkészítő alkalmazások számára, hogy a probléma megoldásához numerikus változók kezelésére is szükség lehet.

A második, sokkal szembetűnőbb különbség a következő:

```
(:functions
        (weight ?x - cube)
        (energy-left))
```

Ez a rész a numerikus változók deklarációja. Azért szerepel az elején `:functions`, mert a PDDL-t specifikáló szakemberek döntése az volt, hogy numerikus értékű *függvények* formájában definiálják a numerikus változókat. Tehát minden egyes függvény egy-egy numerikus változónak felel meg. Pontosabban egy bizonyos „fajta” numerikus változónak.

Vegyük például a `(weight ?x - cube)` függvényt. Ez adott „`?x`” `cube`-típusú objektum-bemenetre valamilyen numerikus értéket ad eredményül (egy valós számértéket integer, vagy double, azaz `a.b` formában). Ideális esetben minden „`cube`” típusú objektum felett definiálva, azaz értelmezve van az értéke. Ebben az esetben tehát a `weight` függvény a bemenetén kapott tetszőleges konkrét kocka-objektumhoz hozzárendeli annak súlyát. ***Ezt* a függvényt tekintjük *numerikus változó*nak**.

A definícióban nem csak 1-argumentumú, hanem akár több-argumentumú függvény is szerepelhetne. De vannak olyan függvények is (pl. `energy-left`), melyeknek egy argumentuma sincs.

Kicsit „fordított gondolkodásra” vall az egész: arról van szó ugyanis, hogy mi magunk, a tervkészítés előtt, a kezdeti állapot meghatározásakor adjuk meg, hogy adott bemenetre mi legyen a függvény értéke. Tehát a fenti függvényeknek nincs „képlete”, ami alapján a bementből adódik a kimenet. Nem implicite határozzuk meg a kimenetet, hanem explicite mi adjuk meg, hogy egy-egy bemenethez egy-egy függvény milyen numerikus kimenő értéket társítson. Mintha csak numerikus változóknak adnánk értéket… – Erről van szó.

Így tehát pl. a `b` kocka-objektum súlyának értékét tartalmazó numerikus változóra `(weight b)` formában hivatkozhatunk (majd a probléma-leírásban).

A következő különbség, ami szembetűnik a domain-leírásban, a cselekvések előfeltételeiben és következményeiben látható. Haladjunk sorra: a „`move_to_table`” cselekvés előfeltételei kiegészültek a következő feltétellel:

```
(>= (energy-left) (weight ?cube))
```

Ezek szerint ahhoz, hogy a kocka mozgatását véghez lehessen vinni, legalább annyi energiára van szükség, mint amennyi a „?cube" kocka súlya. Ez a kényszer hozza tehát összefüggésbe az energiafogyasztást a kockák súlyával. Ezek szerint a két mennyiség egymással egyenesen arányos, ráadásul az egyszerűség kedvéért 1:1 arányban.

A „move_to_table" cselekvésnek azonban nem csak az előfeltételei, hanem a következményei is kiegészültek:

```
(decrease (energy-left) (weight ?cube))
```

Ezek szerint tehát a „move_to_table" cselekvés sikeres végrehajtása után a megmaradt energiamennyiség, azaz (energy-left) éppen (weight ?cube) értékével csökken (ami az előbbi érvelés nyomán teljesen érthető).

Az „>=", „=", és „<=" beépített eljárások függvények esetén érték-összehasonlításra szolgálnak. Érték-adásra/módosításra ekkor a „decrease", „assign", „increase", „scale-up", és „scale-down" beépített eljárások szolgálnak. Ekkor tehát meg kell adni, hogy minek mi legyen az értéke, vagy mi mennyivel nőjön/csökkenjen, szorzódjon/osztódjon. Mind az öt operátornak 2-2 argumentuma van.

Probléma-leírás

Látható, hogy az előbbi domain-leírásnak megfelelő konkrét problémáknak általánosságban igen sok megoldása lehet. Ezért célszerű valamiféle irányadó ***jósági mérce*** bevezetése, amely leszűkíti a jó megoldások körét, ami alapján mérhető egy-egy megoldási terv jósága. A megoldásnak tehát már nem csak a numerikus korlátokat kell betartania, hanem lehetőség szerint még ezt a jósági mércét is célszerű minél inkább optimalizálnia (értsd. minimalizálnia, vagy maximalizálnia).

A következőkben megadjuk a II.2.4-es szakaszban tárgyalt probléma megfelelő kiegészítését (a megfelelő függvények kezdeti értékével, és a jósági mércével).

```
(define (problem sussman1)
        (:domain sussman)
        (:objects a b c - cube t - table)
        (:init
                (on c a)
                (on a t)
                (on b t)
                (= (weight a) 10)
                (= (weight b) 20)
                (= (weight c) 30)
                (= (energy-left) 60))

        (:goal  (and    (on a b)
                        (on b c)
                        (on c t)))

        (:metric maximize (energy-left))

)
```

A leírásban pirossal kiemelve tüntettük fel azokat a részeket, amik nem szerepeltek a II.2.4-es szakasz probléma-leírásában. Ezek egy része tehát a kiindulási világállapotban igaz tények halmazát bővíti az új fogalmaknak, pontosabban a függvényeknek megfelelően. A másik újdonság pedig a jósági mérce.

Kezdjük az előbbi résszel – az új tényekkel! Sorra haladva láthatjuk, hogy először a kockák súlya szerepel, majd végül a kezdetben rendelkezésre álló összes energia mennyisége.[25]

FONTOS (!!!): Vegyük észre, hogy amíg a domain-leírásban (a cselekvések következmény-részében) `increase`, `assign`, … beépített eljárások által tudtunk csak értéket adni a numerikus változóknak (függvényeknek), addig a probléma-leírásban (lásd. fentebb) az ”=„ beépített eljárás (egyesítés) segítségével. Erre tehát ügyeljünk amellett, hogy – ahogy azt már egyszer kifejtettük – az ”=„ beépített eljárás sem mindig érték-adó (logikai változóknál), néhol érték-összehasonlító (numerikus változóknál).

A kezdeti tények felsorolását a jósági mérce megadása követi. Ez jelenleg a következő:

```
(:metric maximize (energy-left))
```

Jósági mércénk (ami – vegyük észre! – a konkrét problémához tartozik) tehát igen egyszerű: azt írja elő, hogy a leírt probléma megoldásai minimalizálják az összesen felhasznált energia mennyiségét. Ez egy egyszerű, ésszerű előírás, ami arra buzdítja a tervkészítőket, hogy jelen probléma megoldásakor optimalizálják az összesített energia-fogyasztást.

Természetesen nem csak maximalizálni (`maximize`) lehet, hanem minimalizálni (`minimize`) is. Nyilván ez utóbbi jelen esetben (az energia-fogyasztásra vonatkozóan) elég ésszerűtlen volna, viszont más esetekben, más jósági mérce megadása mellett még ésszerű lehet (pl. ha nem `energy-left`, hanem `energy-used` numerikus változót használtunk volna, aminek kapcsán akkor viszont kellett volna még egy numerikus változó, amely megadja a maximumát).

Az optimalizálandó jósági mércének koránt sem kell feltétlen ilyen egyszerűnek lennie, mint most – tetszőlegesen összetett numerikus formula lehet. Viszont, ne felejtsük el, numerikus változók esetén a **jósági mérce kötelező** jelleggel meg kell, hogy jelenjen **a probléma-leírásban**.

[25] A 0-argumentumú függvényeket nem muszáj bezárójelezni.

Probléma-megoldás

Lássuk tehát, hogy milyen megoldást ad az LPG a fentebb definiált problémára:

```
0:   (MOVE_TO_TABLE C A T) [1]
1:   (MOVE_TO_CUBE B T C) [1]
2:   (MOVE_TO_CUBE A T B) [1]
```

A megoldás nem meglepő módon, szerencsére változatlan. Ha viszont pl. akár csak egy hajszálnyival is kisebbre vettük volna a kezdetben rendelkezésünkre álló energia mennyiségét (`energy-left`), akkor az LPG hosszas keresés után arra jutott volna, hogy a probléma nem oldható meg. Viszont több kezdeti energia esetén is ugyanez a megoldás adódik (`-quality` opcióval). Ennek oka, hogy a jelenlegi, triviálisan egyszerű probléma esetén a legrövidebb megoldási terv egyben a jósági mércét is maximalizálja. Az LPG nem tud válogatni (a redundáns esetektől eltekintve) több különböző jóságú megoldás között. Az igazán izgalmas esetek komplexebb, nem triviális problémák esetén adódnak.

A numerikus változókkal kapcsolatban további részleteket Fox és Long 2003-as cikkéből tudhatunk meg. Viszont, újfent hangsúlyozni szeretnénk, **a numerikus változók használata a házi feladatban nem elvárás**. Csak akkor folyamodjunk hozzájuk, ha mindenképp szükséges, indokolt!

V. Irodalomjegyzék

Bacchus, F., Ady, M. (2001). *Planning with Resources and Concurrency: A Forward Chaining Approach*, In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2001), pp. 417-424

Baral, C., Kreinovich, V., Trejo, R. (1999). *Computational complexity of planning and approximate planning in presence of incompleteness.* In The International Joint Conferences on Artificial Intelligence (IJCAI).

Blum, A., Furst, M.L. (1995). *Fast planning through planning graph analysis*. Proc. IJCAI-95, Montreal, Canada.

Bylander, T. (1992). *Complexity results for serial decomposability*. In Proceedings of the Tenth National Conference of Artificial Intelligence (AAAI92), pp. 729-734. San Jose, California. AAAI Press.

Chapman, D. (1987). *Planning for conjunctive goals*. Artificial Intelligence, 32, pp. 333-377

Colmerauer, A. (1975) *Les grammaires de metamorphoses*. GIA, Univ. Marseille-Luminy, France, Tech. Rep.

Currie, K.W., Tate, A. (1991). *O-Plan: the Open Planning Architecture*. Artificial Intelligence, 52 (1), pp. 49-86.

Draper, D., Hanks, S., Weld, D.S. (1994). *Probabilistic Planning with Information Gathering and Contingent Execution*. In Kristian J. Hammond (ed.) Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94), pp. 31-36. Morgan Kaufmann.

Edelkamp, S., Helmer, M. (2000). *On the implementation of MIPS*. Paper presented at the Fifth International Conference on Artificial Intelligence Planning and Scheduling Workshop on Model-Theoretic Approaches to Planning, Breckenridge, Colorado.

Edelkamp S., Hoffmann J. (2003). *PDDL2.2: The Language for the Classical Part of the 4th International planning Competition*, Technical Report No. 195, Institut für Informatik.

Erol, K., Nau, D.S., Handler, J. (1994). *UMCP: A Sound and Complete Planning Procedure for Hierarchical Task-Network Planning*, In AIPS-94, Chicago.

Erol, K., Nau, D.S., Subrahmanian, V.S. (1995). *Complexity, decidability and undecidability results for domain-independent planning*, Artificial Intelligence, Vol. 76, pp. 75-88.

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., Williamson, M. (1992). *An approach to planning with incomplete information*. In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning.

Fikes, R.E., Nilsson, N.J. (1971). *STRIPS: a new approach to the application of theorem proving to problem solving*, Artificial Intelligence, 2 (3-4), pp. 189-208

Fox M., Long D. (2003). *PDDL2.1: An Extension to pddl for Expressing Temporal Planning Domains*, Journal of Artificial Intelligence Research 20: 61-124.

Gelernter, H. (1959). *Realization of geometry proving machine*. In Proceedings of an International Conference of Information Processing, pp. 273-282 Paris. UNESCO House.

Gerevini, A., Serina, I. (2002). *LPG: a Planner based on Local Search for Planning Graphs*, in Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02), Toulouse, France.

Gerevini, A., Long D. (2005). *Plan constraints and preferences in PDDL3 - the language of the fifth international planning competition*, University of Brescia, Italy, Tech. Rep.

Green, C. (1969). *Theorem-proving by resolution as a basis for question-answering systems.* In Meltzer, B., Michie, D., and Swann, M. (eds.), Machine Intelligence 4, pp. 183-205. Edinburgh University Press, Edinburgh, Scotland.

Helmert, M. (2008). *Changes in PDDL 3.1*. http://ipc.informatik.uni-freiburg.de/PddlExtension

Hoffmann, J., Nebel, B. (2001). *The FF Planning System: Fast Plan Generation Through Heuristic Search*, In Journal of Artificial Intelligence Research, Vol. 14, pp. 253-302

Kautz, H., Selman, B. (1996). *Pushing the Envelope: Planning, Propositional Logic and Stochastic Search*, In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), pp. 1194-1201. MIT Press.

Kautz, H., Selman, B. (1998). *BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving*. Workshop Planning as Combinatorial Search, AIPS-98, Pittsburgh, PA.

Littman, M.L., Goldsmith, J., Mundhenk, M. (1998). *The computational complexity of probabilistic planning*. Journal of Artificial Intelligence Research, volume 9, pp. 1-36

McAllester, D., Rosenblitt, D. (1991). *Systematic nonlinear planning*. In Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), Vol. 2, pp. 634-639. Anaheim, California. AAAI Press.

McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine. Communications of the ACM, Vol. 3, Issue 4, pp. 184-195.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M.; Weld, D., Wilkins, D. (1998). *PDDL---The Planning Domain Definition Language*. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

McDermott, D. (2000). The 1998 AI planning systems competition. AI Magazine 21, no. 2: pp. 35--55.

McGuinness, L. D., van Harmelen, F. (2004). *OWL: Web Ontology Language overview*. W3C Recommendation, http://www.w3.org/TR/owl-features/

Nau, D.S., Cao, Y., Lotem, A., Munoz-Avilla, H. (1999). *SHOP: Simple Hierarchical Ordered Planner*. In Proceedings of IJCAI-99.

Nau, D.S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F. (2003). *SHOP2: An HTN Planning System*. Journal of Artificial Intelligence Research. *(being published)*

Newell, A., Simon, H.A. (1956). *The logic theory machine*. IRE Transactions on Information Theory, IT-2(3), pp. 61-79

Newell, A., Simon, H.A. (1961). *GPS, a program that simulates human thought*. In Billing H. (ed.), Lernende Automaten, pp. 109-124. R. Oldenbourg, Munich, Germany. Reprinted in Feigenbaum and Feldman (1963).

Papadimitriou, C.H. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA.

Penberthy, J.S., Weld, D.S. (1992). *UCPOP: A sound, complete, partial order planner for ADL*. In Proceedings of KR-92, pp. 103-114
Peot, M., Smith, D. (1992). *Conditional nonlinear planning*. In Hendler J. (ed.), Proceedings of the First International Conference on AI Planning Systems, pp. 189-197. College Park, Maryland. Morgan Kaufmann.

Robinson, J.A. (1965). *A machine-oriented logic based on the resolution principle*. Journal of the Association for Computing Machinery, 12, 23-41.

Rónyai, L., Ivanyos, G., Szabó, R. (1998). *Algoritmusok*. Typotex, Budapest.

Russell, S.J., Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall.
Sacerdoti, E.D. (1974). *Planning in a hierarchy of abstraction spaces*. Artificial Intelligence, 5 (2), 1, pp. 15-135

Sacerdoti, E.D. (1975). *The nonlinear nature of plans*. In Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI75), pp. 206-214. Tbilisi, Georgia. IJCAII.

Stefik, M.J. (1981). *Planning and meta-planning*. Artificial Intelligence, 16, pp. 141-169.
Schwefel, H.P. (1995). *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series. John Wiley & Sons, Inc., New York.

Sussman, G.J. (1973). *A computational model of skill acquisition*. Massachusetts Institute of Technology, Technical Report No. AI TR-297.

Tate, A. (1975). *Interacting goals and their use*. In Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75), pp. 215-218, Tbilisi, Georgia. IJCAII.

Tate, A. (1977). *Generating project networks*. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI77), pp. 888-893. Cambridge, Massachusetts. IJCAII.

Warren, D.H.D. (1974). *WARPLAN: a system for generating plans*. Department of computational logic memo 76, University of Edinburgh, Edinburgh, Scotland.

Warren, D.H.D. (1976). *Generating conditional plans and programs*. In Proceedings of the AISB Summer Conference, pp. 344-354

Wilkins, D.E. (1988). *Practical Planning: Extending the AI Planning Paradigm*. Morgan Kaufmann, San Mateo, California.

Younes, H.L.S., Simmons, R.G. (2002). *On the role of ground actions in refinement planning*. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso (eds.), Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling Systems, pp. 54-61, Toulouse, France. AAAI Press.

Younes, H.L.S., Littman, M.L. (2004). *PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects*. Technical report, CMU-CS-04-167, Carnegie Mellon University.