1. Bevezetés

Kognitív tudományhoz több diszciplína is hozzájárul: pszichológia, lingvisztika, antropológia, mesterséges intelligencia, …

Kognitív tudomány célja: a tudás integrálásása kognitív jelenséges magyarázatára (probléma megoldás, döntéshozatal, nyelv, memória, tanulás, …)

A különböző diszciplínák különböző típusú kérdéseket tesznek fel és különböző típusú válaszokat adnak → EGYSZERRE ELŐNY ÉS HÁTRÁNY (Allen Newell)

Előny:

- Szakértelem
  - szabályszerűségek a viselkedésben
  - elméletek a szabályszerűségek magyarázatára
  - példák: garden path problem, Fitts' law, Tulving learning order

Hátrány: az elméletekben rejlik (Newell: *mikroteóriák*): a részeredmények megalkotásakor nem léteztek megkötések, amelyek garantálták volna, hogy az összes többi részeredménynek nem mondanak ellent. Igaz: nem minden struktúra és mechanizmus, amely a „mikro teóriák" alapját képezi, kell hogy alapját képezze a többi részelméletnek is, DE fontos hogy kompatibilisek legyenek (elvégre egy rendszer mutatja az összes vizsgált viselkedést)

Egyetlen megoldás: megpróbálni a teljes képet összerakni -> Unified Theory of Cognition (UTC - Newell). Igy jött létre a SOAR.

2. Architektúra

Nem új ötlet, pl. hardver architektúrákról sok szó esik. A szoftverre vonatkozó különböző feltételezések különböző architektúrákat tesznek előnyössé. („M architektúra hatékonyan futtatja-e A és B programokat?", ahelyett hogy „M architektúra hatékony-e?")

Nemcsak HW, hanem SW esetében is megkérdezhetjük, hogy bizonyos magasabb rendű műveleteket mennyire hatékonyan hajt végre egy SW -> SW architektúra

1. → Minden komplex rendszernél megkülönböztethető:
   - fix mechanizmusok, architektúra struktúrái
   - tartalom, amit a mechanizmusok a struktúrákon keresztül feldolgoz

   VISELKEDÉS = ARCHITEKTÚRA + TARTALOM

2. → Minden architektúrából kiviláglanak a fejlesztő feltételezései a feldolgozandó tartalom jellegéről

Tehát az architektúra fogalma hasznos, mert a viselkedések széles köréből lehetővé teszi a közös aspektusok kiszűrését.

Kognitív architektúra: fix mechanizmusok és struktúrák elmélete, amely leírja az emberi kogníció tulajdonságait.

3. SOAR mint kognitív architektúra

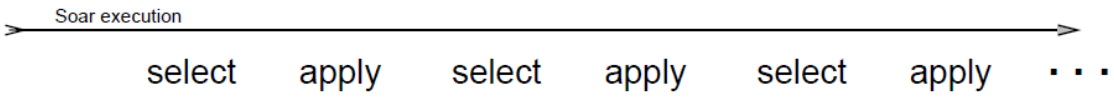
Kérdés: milyen viselkedést szeretnénk a SOAR-ral modellezni? Példák kognitív viselkedésre: egyenletek megoldása, vacsora főzés, autó vezetés, viccmesélés, stb. SOAR feltételezi, hogy a kognitív architektúra legalább a következő tulajdonságokat hordozza:

- Cél-orientált (goal oriented)
- Komplex, részletekben gazdag környezetben játszódik le a viselkedés (példa: vezetés ismeretlen helyre)
- SOK tudást igényel (gondoljunk bele, mennyi tudás kell egy egyenlet megoldásához - mi legyen, ha eltörik a ceruza, kiég a villany, számtani tudás, stb.)
- Szimbólumok és absztrakciók használata szükséges (konkrét csirke <->csirke fogalma)
- Rugalmas, és a környezet függvénye (NEM fix terv, pl. dugóban más irányba vezetek, stb.)
- Szükséges hozzá a környezetből való tanulás és tapasztalat

Vannak mások is, de a SOAR ezekből indul ki. Tehát a SOAR olyan mechanizmusai és struktúrái eleve lehetővé fogják tenni hogy a viselkedés ezeknek a követelményeknek eleget tegyen.

3.1 SOAR – áttekintés

Hipotézis: minden cél-orientált viselkedés leírható úgy, mint operátorok alkalmazásának sorozata adott állapotban
Állapot: jelenlegi probléma-megoldó helyzet leírása
Operátor: transzformálja az állapotot (annak belső reprezentációját módosítja)
Cél: kívánt kimenetele az operátorok alkalmazásának
Futáskor a SOAR folyamatosan alkalmazza az *aktuális operátort*, majd újabbat választ végrehajtásra, egészen addig amíg a cél nincs elérve. Egy állapotban egyszerre csak egy aktuális operátor van.



Külön tároló létezik a SOAR-ban a jelenlegi állapot, és a hosszú távú tanulás során felhalmozott tudás tárolására.
A *munka memória* (working memory) objektumok gyűjteménye, amelyek együtt leírják a jelenlegi állapotot (állapotot, szenzorinformációt, inferencia – következtetés - köztes állapotait, aktív célokat,

aktív operátorokat). Az objektumok attribútumokat tartalmaznak, és az attribútumok mutathatnak más objektumokra → hierarchikus felépítés (de lehet körkörös szerkezetű is)
Az LTM leírja, hogy milyen munka memória tartalomhoz milyen viselkedést érdemes produkálni. Tartalma tekinthető a SOAR architektúra programjának.

3.1.1 Problémamegoldó funkciók

A rendszer működése operátor választás és alkalmazás köré összpontosul, ezért a következő funckiók támogatottak:

- Tudás operátor kiválasztásához
  - operátor javaslat (az operátor alkalmas a jelenlegi szituációban)
  - operátor összehasonlítás (rendezés az operátorok hatékonysága között)
  - operátor választás (az optimális operátor kiválasztása)
- Tudás operátor alkalmazásához
  - Hogyan módosít adott állapotot egy adott operátor?

És ráadásként mindkét feladatkörhöz tartozik az alábbi tudás:

- Állapoton elvégezhető lehetséges monoton következtetések (állapot elaboráció, állapot bővítése)

Ez utóbbi tudás az adott állapot más szemszögből történő leírását teszi lehetővé, megkönnyítve további operátorok kiválasztását.

SOAR-ban a tudás reprezentálása ún. produkciós szabályokkal történik (if-then jellegű szabályokkal, kondíció-akció párokkal) (NEM EGYENLŐ KÖVETKEZTETŐ RENDSZER, LÁSD KÉSŐBB)

Döntési ciklus: összes alkalmazható operátor kiválasztása, optimális operátor kiválasztása preferenciák alapján. Néhányszor azonban ún. *impasse* – azaz döntésképtelenség – jöhet létre:

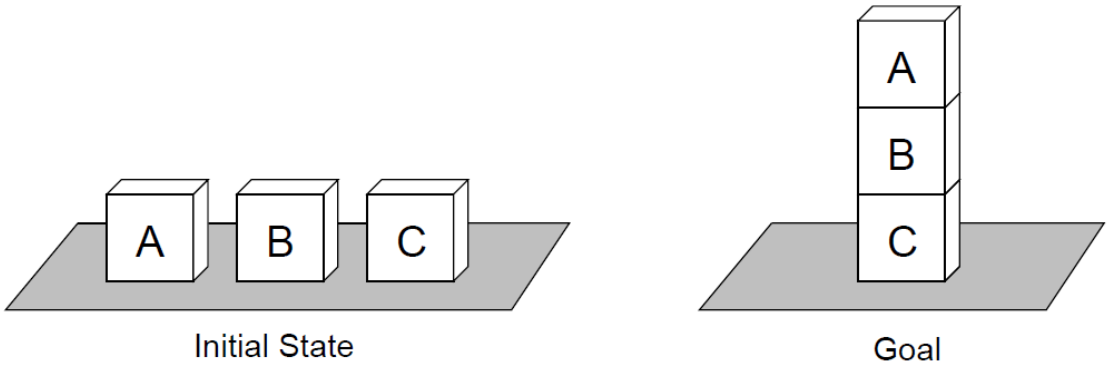
- Nem lehet operátort választani mert egy javaslat sincs
- Nem lehet operátort választani mert több javaslat van de nem áll rendelkezésre elég tudás az összehasonlításukhoz
- Operátor választás sikeres, de az alkalmazásához nem áll rendelkezésre elég tudás

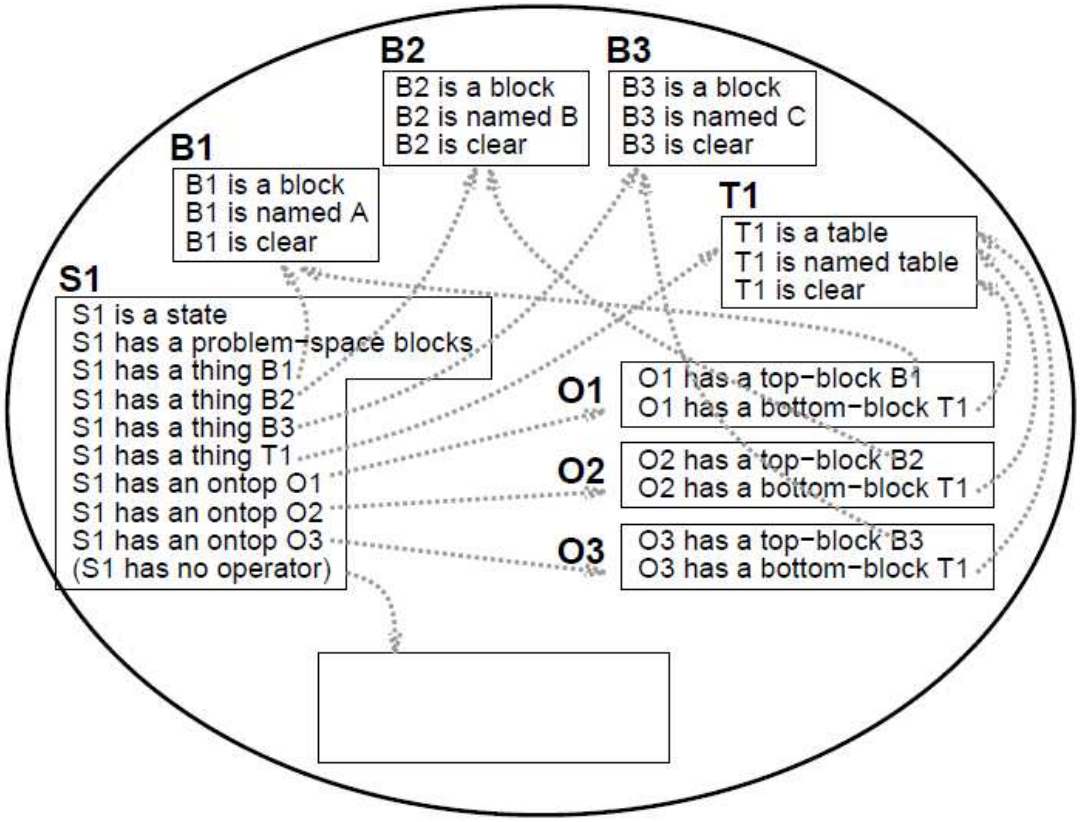
Ilyenkor ún. al-állapotot hoz létre a SOAR (substate), melyben a cél a döntésképtelenség feloldása olyan tudással, amely nem állt rendelkezésre a felsőbb szinten (pl. a SOAR előre „lejátszhatja” az egyes operátorok lehetséges kimeneteit)

3.1.2 Példa alkalmazás a fejezet során: **Blokk világ**

Operátorok tartalmazzák:

- mozgatandó blokk neve
- mozgatandó blokk jelenlegi pozíciója
- mozgatandó blokk célpozíciója



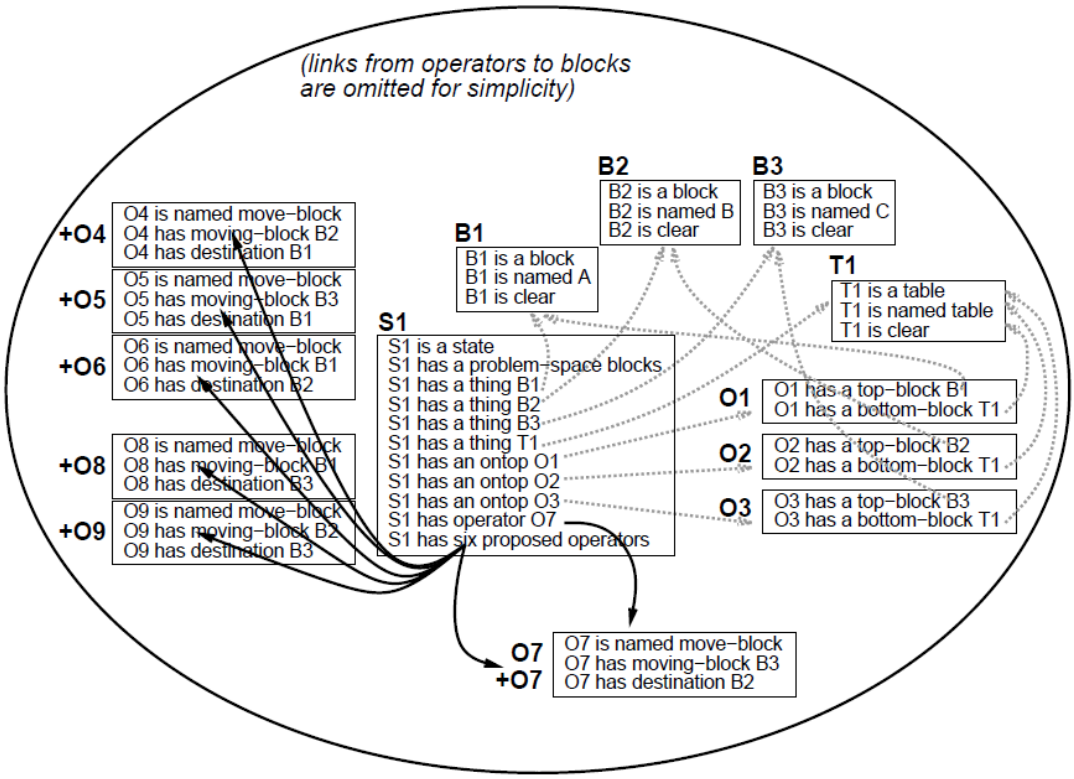


An Abstract View of Working Memory

3.1.3 Állapotok, operátorok és célok reprezentálása

Az ábrán látható a munka memória tartalma, amikor még egyetlen operátort sem javasolt a SOAR rendszer.

Egyszerre csak egy operátora lehet az állapotnak, és az operátor az állapot alstruktúrájaként kerül reprezentálásra. Egy állapotnak lehet továbbá több javasolt operátora is



A célok lehetnek explicit módon reprezentálva, úgy mint az állapot alstruktúrái, vagy implicit módon is definiálhatjuk őket a program részeként (ekkor a program ellenőrizni tudja produkciós szabályaival, hogy teljesült-e cél). A Blocks World programban ez utóbbi módszert használjuk.

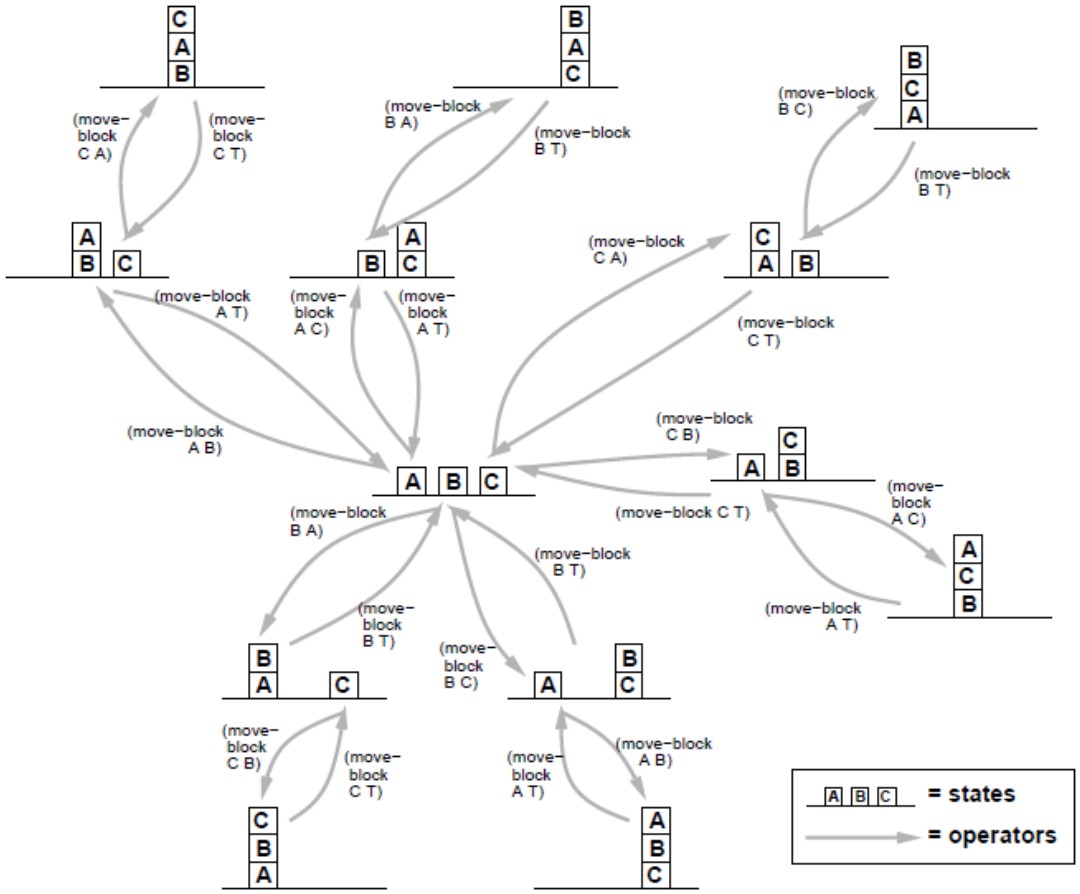
3.1.4 Feldolgozás lépései:

1. Lehetséges operátorok javaslása: 6 db operátor
2. Operátorok összevetése preferencia szerint
    a. első körben nincsenek preferenciák
3. Operátor választása
    a. van legjobb, vagy
    b. nincs legjobb, de van egy preferált részhalmaz amiből random lehet választani
    c. nincs legjobb, és sem a, sem b nem áll fent
    d. nincs javasolt operátor (c és d döntésképtelenséget eredményez)
4. Operátor alkalmazása
    a. változtatás függ a jelenlegi állapottól és az operátortól
    b. Lehet indirekt (változás az érzékelésből jön) vagy direkt (belül tudja, mi a változás). Előfordulhatnak vegyesen is. Belső problémamegoldásnál (pl. döntésképtelenség) direkt módosításokat kell végrehajtani az elvárások szerint (persze a környezet másképpen is reagálhat)
    c. A Blocks World feladatban csak direkt módosítások lesznek. 4 lehetséges módosítás van:
        i. az elmozdított blokk már nem ott van, ahol volt

ii. az elmozdított blokk új helyen van (valami tetején)
iii. az a tárgy, amiről elmozdítottuk a blokkot, most már üres
iv. az a hely, ahova a blokkot visszük, már nem üres (kivéve ha az az asztal)

d. ha szimulátort használnánk, bizonyos állapot információk onnan jönnének (mi az ami üres, mi az aminek a teteje már nem üres)

5. Következtetés: bizonyos tényekre következtetéssel jön rá a rendszer. Elönye, hogy
   a. leegyszerűsíti az operátorok kódolását (nem kell minden eshetőséget kódolni)
   b. a SOAR automatikusan visszavonja a következtetéseket, ha már nem állnak fent
   c. Például a Blocks Worldben következtetéssel jön rá a rendszer, hogy egy tárgy teteje üres vagy fedett. Az elaboráció megnézi, hogy van-e olyan tárgy, ami az adott tárgyon rajta van (ha nincs, akkor üres)
   d. új ontop típusú objektum létrehozásakor a rendszer automatikusan visszavonja a kapcsolódó elaborációt



3.1.5 Probléma tér

Ha minden egyes problémára megoldást akarnánk nyújtani, rengeteg szabály lenne. Az összes szabály közül csak egy korlátozott részhalmaz lenne releváns (pl. kocka számolásnál a mozgatás legális, azonban nem járul hozzá a feladat megoldásához). Adott problémához a releváns operátorok egy probléma teret határoznak meg. Ebbe azok az állapotok tartoznak bele, amelyek között ezekkel az operátokkal navigálhatunk.

A SOAR programokban implicit módon jelenik meg a probléma tér fogalma, mert eleve korlátozott azon operátorok száma, amelyek alkalmazása adott helyzetben felmerülhet. Az ábrán látható a teljes probléma tér. Ezt a probléma teret nem generálja le a SOAR explicit módon, és nem keres benne utakat (állapottér-robbanáshoz vezetne). Ehelyett mindig azt jegyzi meg, hogy melyik állapotban van, és hogy melyik operátorok alkalmazhatóak.

4. Munka memória

Munka memória tartalmaz WME-ket (working memory elements) – azaz azonosító-attribútum-érték hármasokat.

Több WME is tartalmazhat információt egy adott tárgyról („B1 egy blokk", „B1 piros", „B1 az asztalon van"). Ezek ilyenkor egy azonosítóhoz tartozó objektum attribútumai.

Egy WME értéke lehet egy másik objektum azonosítója. Ilyenkor a két objektum összeköttetésben áll.

Minden objektum a munka memóriában (direkt vagy indirekt) összeköttetésben kell hogy álljon állapottal – ellenkező esetben törlésre kerül

Két azonos azonosító-attribútum-érték hármas nem fordulhat elő egyszerre a munka memóriában (halmazként van definiálva). Olyan azonban lehet, hogy szerepel két WME, amelynek azonos az azonosítója és az attribútuma, csak más értékkel (*multi-attribútumok*)

FONTOS, hogy az objektumok azonosítói csupán pointerek, de nem definiálják az objektumot. Az objektumot az ún. augmentációi definiálják (a hozzá tartozó WME-k). Tehát ha létezik egy másik objektum ugyanazon augmentációkkal de más névvel, következtetéskor a rendszer ugyanúgy kezeli a két objektumot. Tehát: produkcióban sose fordulhat elő azonosító!!

Nincs előre meghatározott kapcsolat a munka memória objektumai és a világ objektumai között. A munka memória objektumai lehetnek valós objektumok leképzései (block, asztal), valós objektumok tulajdonságai (piros szín, kocka alak), valós objektumok közötti reláció (ontop).

Az attribútumok nevei nem bírnak speciális jelentéssel a SOAR számára (csak a programhoz illenek). Ez alól csak néhány WME kivétel, amelyet alapból létrehoz a rendszer.

A munka memória tartalma 4 különböző forrásból származhat:

1. Produkciós szabályok akciói (jobb oldala)
2. A döntési procedúra automatikusan létrehoz néhány augmentációt amikor új állapot jön létre (típus, szülőállapot, döntésképtelenség). Ez történhet inicializáláskor, vagy döntésképtelenség esetén.
3. A döntési procedúra az ismert preferenciák alapján augmentálhatja az állapotot lehetséges operátorokkal és kiválasztott operátorokkal
4. Szenzorikus adatok

és 6 különböző módon kerülhet ki onnan:

1. A döntési procedúra automatikusan kitöröl minden olyan augmentációt, amit döntésképtelenség miatt hozott létre, ha a döntésképtelenség feloldásra kerül

2. A jelenlegi operátor augmentáció törlésre kerül, ha már nem ugyanaz az aktuális operátor
3. *reject* preferencia alkalmazásakor törlésre kerülnek azon lehetséges operátorok, amelyek korábbi produckiós szabályok révén jöttek létre
4. következtetések, amelyek már nem állnak fenn
5. I/O rendszer megszünteti azokat a szenzorikus jeleket, amelyek már nem érvényesek
6. A rendszer automatikusan eltávolítja azokat a WME-ket, amelyek már nincsenek kapcsolatba állapottal (pl. mert egy köztes WME is megszűnt)

5. Produkciós memória

Egy produkciós szabály: feltételek halmaza + akciók halmaza

Ha a feltételek teljesülnek a munka memória állapotára, akkor a szabály tüzel

5.1 Produkciós szabályok struktúrája

Példa (egyszerűsítéssel)

Feltételek: Block A teteje üres, Block B teteje üres

Akciók: suggest operator: move block A on block B

De a feltételek tartalmazhatnak negált kifejezéseket is (nem létezik olyan kocka, ami piros, stb.)

5.1.1 Változók a produkciós szabályokban

A példában a blokkok mereven kódoltak. Ehelyett célszerű változókat használni
Egy produkciós szabályban ha a változókat konkrét azonosítókkal helyettesítjük be, és minden változóra igaz, hogy minden előfordulását ugyanazon azonosítóra le tudjuk cserélni, akkor a változó behelyettesítéseket egy *példányosításnak* nevezzük. Egyazon produkciós szabály több példányosítással is kielégíthető lehet. Ilyenkor többszörös példányosításról beszélhetünk, és több tüzelés jön létre.

5.2 Produkciós szabályok architektúrális szerepe

A produckiós szabályok szerepet játszanak az operátorok javaslatában, operátorok közötti választásokban, operátorok alkalmazásában, állapotok elaborációjában

Egyetlen produkciós szabály kizárólag egyetlen ilyen szerepet szabad hogy betöltsön.

5.3 Produkciós szabályok akciói és perzisztencia

Általában az akciók vagy preferenciát hoznak létre, vagy hozzáadnak valamit / elvesznek valamit a munka memória tartalmához/tartalmából.

Preferenciák csak addig érvényesek, amíg az őket létrehozó állapot-konfiguráció fennáll. Ha már nem áll fent, akkor a SOAR automatikusan megsemmisíti őket a munka memóriában. Az ilyen tartalmat i-támogatottnak nevezzük (i-supported, azaz instantiation-supported)

Ezzel szemben az operátorok alkalmazása perzisztens kell, hogy legyen, még akkor is amikor az operátor javaslását generáló szituáció már nem áll fent. Az ilyen produkciós szabályokat O-támogatottnak nevezzük (operátor-support).

6. Preferencia memória (szelekciós tudás)

Szemantikák:

- acceptable (+)
- reject (-)
- better (>)
- worse (<)
- best (>)
- worst (<)
- indifferent (=) (random döntés)
- numeric-indifferent (befolyásolja a random döntést)
- require (!)
- prohibit (~)

7. A SOAR végrehajtási ciklusa (alállapotok nélkül)

Minden ciklus 5 fázisból áll:

1. Input - új szenzorikus adatok
2. Javaslatok – a lehetséges produkciós szabályok tüzelnek, illetve a visszavonandók visszavonásra kerülnek. így megtörténik az elaboráció (új adatok értelmezése), az operátor javaslatok, operátorok összehasonlítása. Ezek mind i-támogatott akciók. Minden illeszkedő szabály egyszerre tüzel, és minden visszavonandó szabály visszavonása egyszerre történik. Ez egészen addig folytatódik ciklusban, amíg le nem áll a folyamat (nincs több szabály)
3. Döntés – operátor választása, vagy ha ez nem létezik, alállapot létrehozása (döntésképtelenség)
4. Alkalmazás – a kiválasztott operátor végrehajtásra kerül. Ezek az akciók o-támogatottak. A végrehajtás miatt előfordulhat, hogy a korábban tüzelt i-támogatott akciókat vissza kell

vonni, és helyettük újabbak tüzelhetnek. Ezért ciklikusan végre kell hajtani a változtatásokat, egészen addig amíg a folyamat le nem áll.

5. Kimenet (motorikus, stb.)

A ciklus egészen addig folytatódik, amíg a SOAR ki nem ad egy halt parancsot, vagy amíg a felhasználó meg nem szakítja.

8. Döntésképtelenség és alállapotok

Több okból jöhet létre döntésképtelenség. Előfordulhat, hogy nincs elég információ a döntéshez, de az is, hogy a preferenciák inkonzisztensek (elvileg előfordulhat, mert egyenként kerülnek be a rendszerbe).

Az alábbi 4 fajta döntésképtelenséget különböztetjük meg:

1. Döntetlen (több operátor kap acceptable minősítést)
2. Konfliktus (A jobb mint B; ugyanakkor B jobb mint A, és egyik sincs reject, prohibited vagy required minősítésben)
3. Korlátozás-ellenesség (több, mint egy operátor required, vagy egy operátor egyszerre required és prohibited is)
4. Változatlanság (egyetlen operátort sem javasolt a rendszer, vagy nem tudja hogyan kell alkalmazni)

8.1 Új állapotok létrehozása

Döntésképtelenség esetén új állapot jön létre. Az állapot szülőállapota a korábbi állapot, és célja a döntésképtelenség feloldása. A feloldás többféleképpen történhet: lehet keresés, egy külső ágens megkérdezése, stb. A döntésképtelenség feloldásakor előfordulhat, hogy újabb döntésképtelenség áll elő. Ilyenkor rekurzívan létrejön egy újabb alállapot. Az így létrejött stackben minden cél feldolgozás alatt áll, de a SOAR leginkább csak a legalsóhoz fog hozzányúlni (néha a felsőbb szintekhez is)

8.2 Eredmények

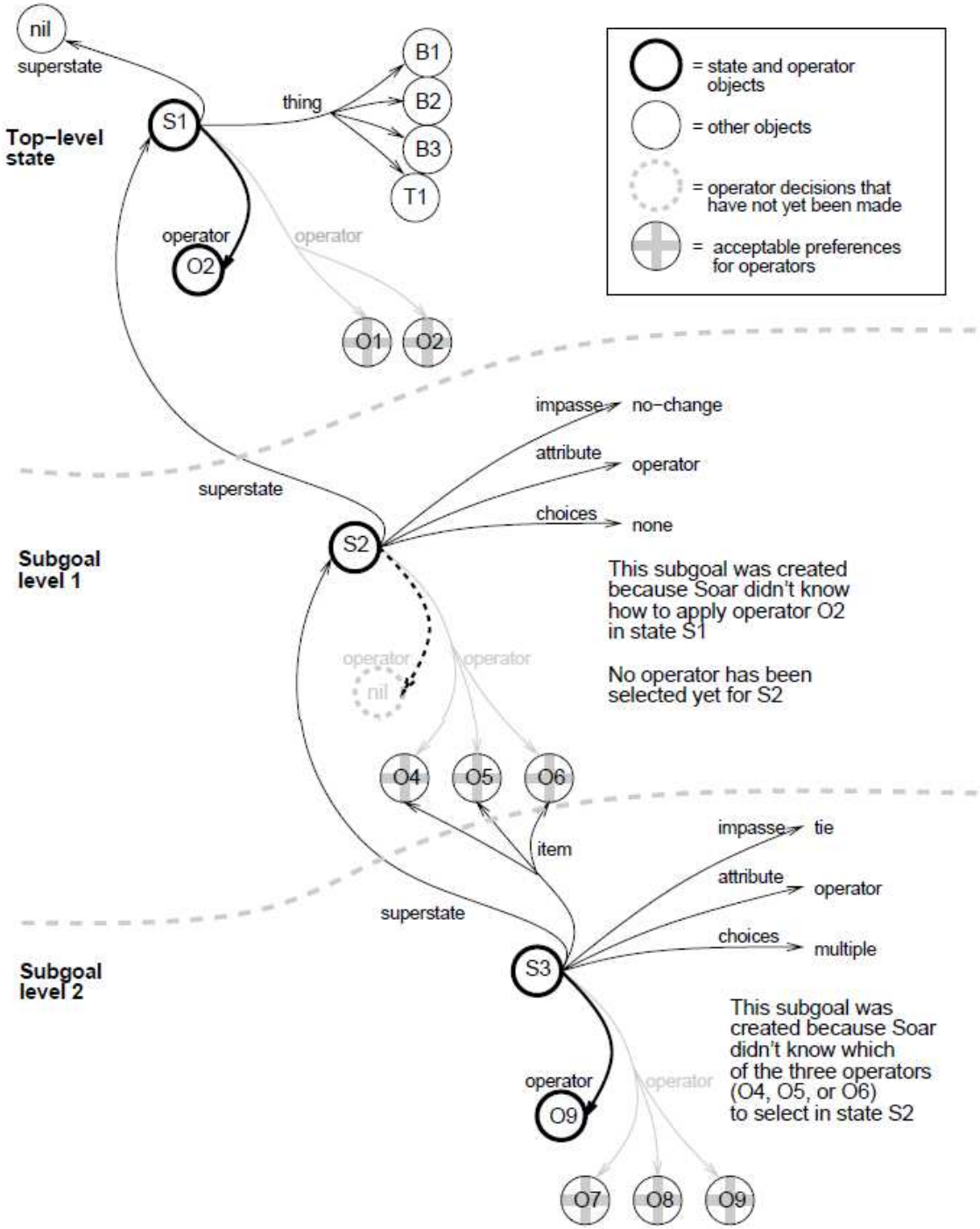
Az alállapot céljának elérése eredménnyel jár. Ezek az eredmények olyan WME-k illetve preferenciák, amelyek kapcsolódnak a felsőbb szintekhez is a stack-ben. Ez egyrészt úgy fordulhat elő, hogy a létrehozott WME azonosítója már kapcsolatban van egy szülő állapottal. Másrészt viszont akkor is előfordulhat, ha a WME létrejötte után létrejön egy újabb eredmény, amin keresztül már kapcsolódni fog szülő állapothoz. (Például létrejön egy operátor, ami még nincs kapcsolatban a szülő állapottal, és csak később jön létre rá az acceptable preferencia a szülő állapotban)

Amikor az alállapot sikeresen lefut és megszűnik, az eredmények nem kerülnek törlésre, mert még a felsőbb szintekkel is kapcsolatban állnak. Az eredményeknek vagy i-, vagy o-támogatásuk van.

8.2.1 I- és O-támogatás meghatározása

Az, hogy egy eredmény I- vagy O-támogatott lesz-e, attól függ hogy a szülő állapotban milyen szerepet tölt be (és nem abban az állapotban, amiben létrejött). Ennek eldöntéséhez a SOAR

egy ún. *indoklási produkciót* (justification production) hoz létre. Az indoklási produkció összefoglalja az eredmény létrejöttének körülményeit. Feltételrendszerében szerepel minden olyan megvizsgált feltétel (a szülő állapotokban, kezdve az eredményt majd felhasználó állapottól egészen felfelé), amely szükséges volt a tüzelés létrejöttéhez, az akció pedig maga az eredmény. Ezek után a korábbiakban tárgyalt módon eldönti a SOAR, hogy az indoklási produkció akciója a szülő csomópont kontextusában i- vagy o-támogatott.



8.2.2 Alállapotok eltávolítása – döntésképtelenség feloldása

Az alállapotok létrejötte egyáltalán nem jelez problémát a SOAR programban. Inkább azt lehet mondani, hogy az egyetlen módja a komplex feladatok részfeladatokra történő felbontásának. Továbbá lehetővé teszi az előre történő gondolkodást, illetve a tanulást, ahogy látni fogjuk.

Alállapotok megoldása

Amikor az alállapot létrehoz egy eredményt, és ez az eredmény továbblépést eredményez egy korábban megakadt szülőállapotban, akkor az alállapot elérte célját, és az összes részstruktúrájával együtt megszűntethető. Ilyenkor a szülő állapotra vonatkozó tanulságokat vonhat le a rendszer (tanulás)

Alállapotok eliminációja

Az alállapot megszűntethető anélkül hogy teljesen lefutott, ha egy felsőbb szinten létrejött döntésképtelenség megszűnik, és így okafogyottá válik az alsóbb szinten létrejött döntésképtelenség feloldása. Olyan esetekben is megszűntethető, ha olyan bemeneti jel érkezik a külvilág felől, amely módosítja az egyik szülő állapotot, és immár lehetségessé válik egyetlen operátor kiválasztása. A rendszer ilyenkor nem tud tanulni.

Alállapotok újragenerálása

Szükségessé válhat időnként egy alállapot újragenerálása, mielőtt még feloldásra került volna. Ennek lehetséges oka, hogy egy külső jel, vagy egy szülő állapotban történő lépés (eredmény okozta) inkonzisztenssé teheti a szülő állapot és a gyermek állapot struktúráit. Ilyenkor megváltozik a megoldandó probléma, ezért újra kell generálni az alállapotot.

9. A SOAR futási ciklusa – döntésképtelenség feloldásával együtt

Minden ciklus 5 fázisból áll:

1. Input - új szenzorikus adatok
2. Javaslatok – a lehetséges produkciós szabályok tüzelnek, illetve a visszavonandók visszavonásra kerülnek, sorban a stack legfelsőbb szintjétől az alsóbb szintekig. Egy alsóbb szinten csak akkor kerül végrehajtásra tüzelés, ha a felsőbb szinteken már az összes lehetséges tüzelés megtörtént. így megtörténik az elaboráció (új adatok értelmezése), az operátor javaslatok, operátorok összehasonlítása. Ezek mind i-támogatott akciók. Minden illeszkedő szabály egyszerre tüzel, és minden visszavonandó szabály visszavonása egyszerre történik. Ez egészen addig folytatódik ciklusban, amíg le nem áll a folyamat (nincs több szabály)
3. Döntés – operátor választása, vagy ha ez nem létezik, alállapot létrehozása (döntésképtelenség)

4. Alkalmazás – a kiválasztott operátor végrehajtásra kerül. Ezek az akciók o-támogatottak. A végrehajtás miatt előfordulhat, hogy a korábban tüzelt i-támogatott akciókat vissza kell vonni, és helyettük újabbak tüzelhetnek. Ezért ciklikusan végre kell hajtani a változtatásokat, egészen addig amíg a folyamat le nem áll.
5. Kimenet (motorikus, stb.)

A ciklus egészen addig folytatódik, amíg a SOAR ki nem ad egy halt parancsot, vagy amíg a felhasználó meg nem szakítja.

10. Tanulás

Feljebb említettük, hogy mely esetekben tanul a SOAR rendszer (amikor egy döntésképtelen helyzet feloldásra kerül). Ilyenkor ún. *chunk*-okat hoz létre. A chunkok nagyon hasonlítanak az indoklási szabályokhoz, azonban fontos különbségek, hogy:

- a chunkok a produkciós memóriába kerülnek, míg az indoklási szabályok nem
- az indoklási szabályok törlésre kerülnek, mihelyt megszűnik létjogosultságuk a munka memóriában
- a chunkok változókat tartalmaznak, az indoklási szabályok pedig példányosítottak

11. SOAR összehasonlítása szabályalapú rendszerekkel (Rule-Based Systems - RBS)

- Párhuzamos, asszociatív memória
  - minden releváns tudásnak szerepe van
  - nem olyan mint az RBS mert az RBS tipikusan egy szabályt választ ki végrehajtásra, és ezt általában szintaktikus szabályok alapján teszi (melyiket éppen a legkönnyebb végrehajtani). A SOAR ezzel szemben minden tüzelhető szabályt „egyszerre” végrehajt
- Belief maintenance
  - kis számítási igényű karbantartási algoritmus
  - RBS-ben minden hozzáadott szabályt külön szabályokkal kell karban tartani, és a kontextus minden változása explicit elhatározás eredménye -> RBS túl merev
- Preferencia-alapú érvelés
  - tudás-alapú választás
  - sosem választ, ha több lehetőség van és nem tud dönteni, ehelyett alállapotot hoz létre, amiben „előregondolkodás” lehetséges
- Automatikus cél generálás
  - lehetővé válik a metaszinten történő érvelés (alállapot létrehozásánál a cél nem a feladat megoldása, hanem a döntésképtelenség feloldása, mindez automatikusan történik)
  - RBS általában nem rétegezett ilyen módon
- Feladat dekompozíció
  - csak az adott feladathoz szükséges tudás van a munka memóriában
  - gyorsabb keresést eredményez
- Adaptáció általánosítás révén
  - integrált, általános és rugalmas tanulás