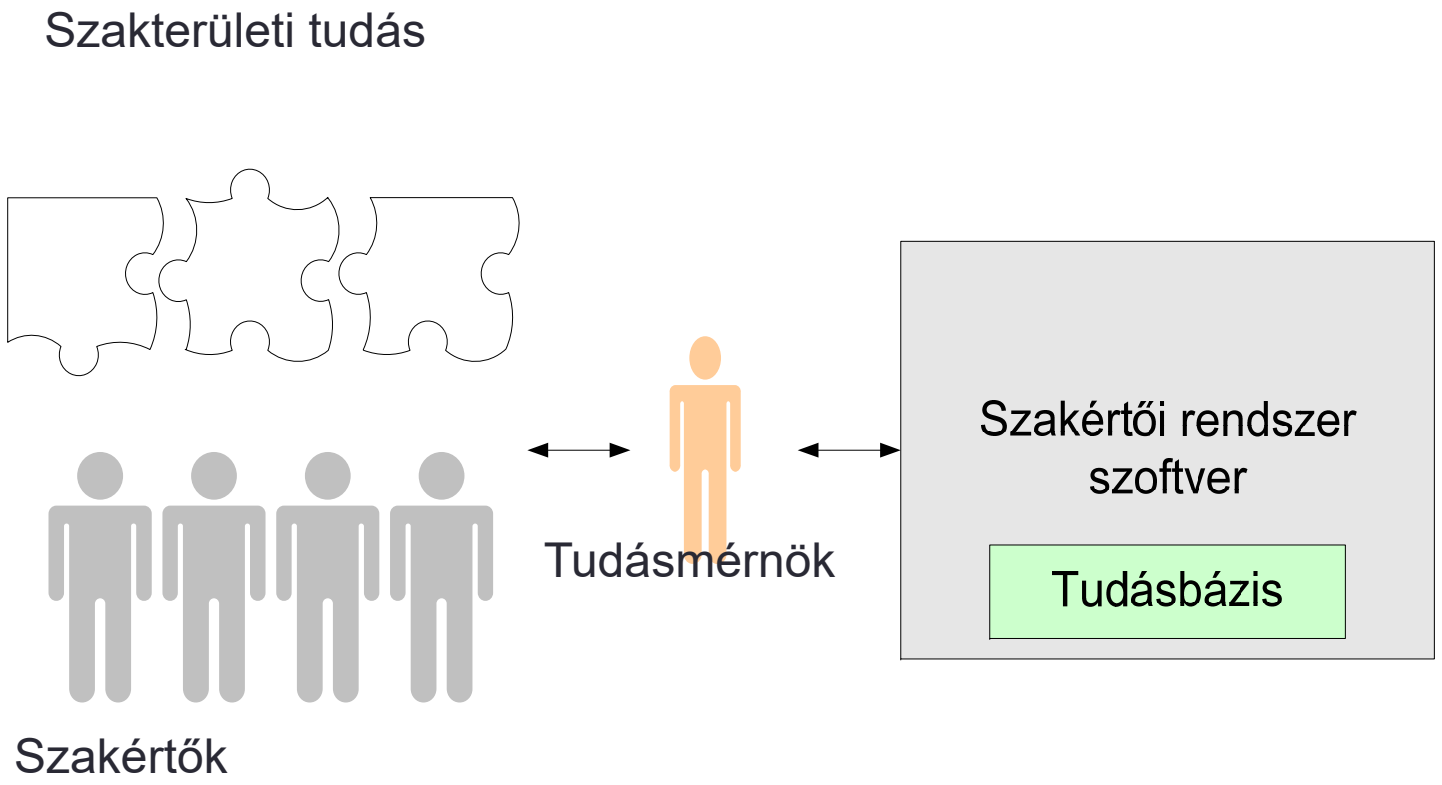
SZAKÉRTŐI RENDSZEREK, LOGIKAI KÖVETKEZTETÉS ALAPÚ MEGOLDÁSOK

Mi is az a szakértői rendszer?

- mesterséges intelligencián alapuló szoftver rendszer
- jól körülhatárolt, viszonylag szűk szakterület ismeretanyagára és humán szakértők tapasztalati tudására épül (adatok, tények, szabályok, összefüggések, általános és különleges esetek stb.)
- a felhasználó által szolgáltatott adatok alapján képes viszonylag bonyolult problémákat megoldani, döntéseket hozni, tanácsot adni, válaszolni a felhasználó kérdéseire
- magyarázatadásra képes (feltett kérdések oka, kikövetkeztetett eredmény)

Hogy készül a szakértői rendszer?



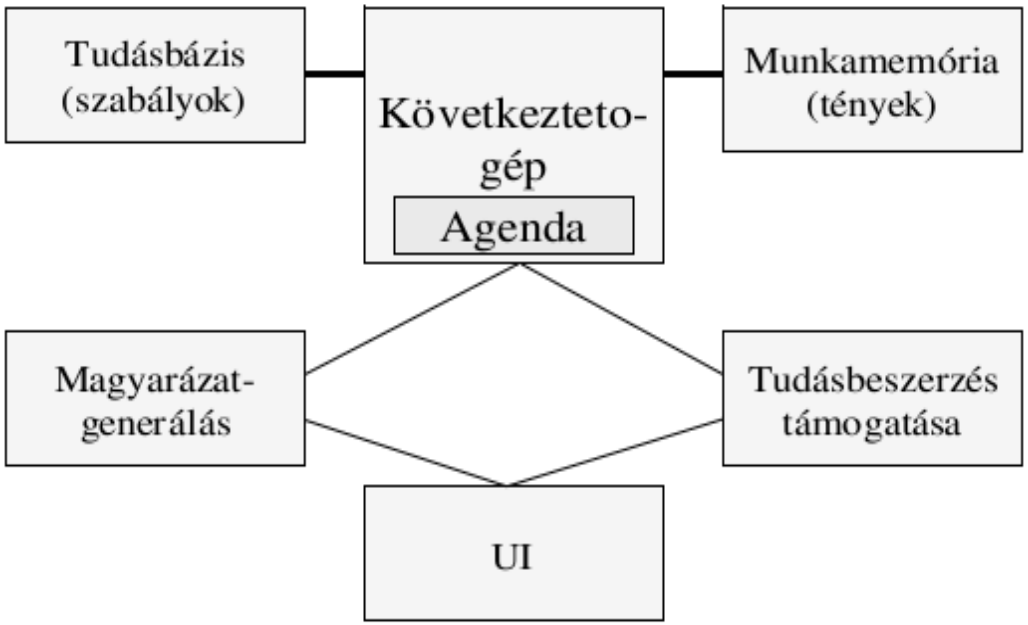
Szabályalapú szakértői rendszer komponensei

Az eredeti CLIPS két modult tartalmazott:

- produkciós szabályleíró nyelv
- procedurális nyelv

A szabályleíró nyelv fontosabb komponensei

- ténybázis
- szabálybázis
- következtetőgép



Szabályalapú szakértői rendszer komponensei

- Ténybázis (ténylista): a probléma kezdeti vagy aktuális állapotát reprezentálja. Adat, ami alapján következtethetünk.

- Szabálybázis (tudásbázis): szabályok halmaza, amely leírja, hogy hogyan juthatunk el a problémától a megoldásig.
  (A CLIPS csak előrefele következtetést támogat.)

Szabályalapú szakértői rendszer komponensei

- Következtetőgép: vezérli a végrehajtást. Illeszti a tényeket a szabályokhoz, hogy meghatározza mely szabályok alkalmazhatóak.
- Felismer-végrehajt ciklusban dolgozik:
- Illesztés (match): tények szabályok feltétel részéhez illesztése (konfliktus halmaz létrehozása)
- Választás (choose): mely szabályok alkalmazhatóak (konfliktus feloldás)
- Végrehajtás (execute): a győztes szabályban leírt akciók végrehajtása

Szabályalapú szakértői rendszer komponensei

Konfliktus feloldási stratégia

- Frissesség
  - Utoljára felkerült szabályok preferálása.
- Specifikusság
  - Az a szabály, amely a legjobban illeszkedik az adott helyzetre preferált.

Hasznos, ha általános és kivételkezelő szabályokkal dolgozunk

- •Ciklusmentesség
  - Ugyanazokra az adatokra csak egyszer hajtja végre a szabályokat.
  - Megakadályozza a ciklusokat

SZR fejlesztés követelményei

- A feladat jelentős részben igényel kognitív képességeket
- Legalább egy megfelelő szakértő hajlandórészt venni
- A résztvevőszakértők képesek definiálni a problémát
- A résztvevőszakértők képesek közös véleményt kialakítani a szakterületi tudásról
- A feladat nem túl bonyolult és jól megfogalmazott
- A feladat alapvetően állandó jellegű
- Konvencionális (algoritmikus) megoldások nem adnak kielégítő megoldást
- A környezet képes nem optimális, esetleg helytelen megoldásokat tolerálni.
- Adat (minta) és teszt információk rendelkezésre állnak
- A feladatot leírófogalmak száma nem haladja meg a pár százat

A SZR megközelítés helyességének igazolása

- A probléma megoldás magas megtérülésű
- A SZR megoldás megőrzi a szakértői tudást, nem veszik el értékes információ
- Számos helyen van szükség a szakértői tudásra
- A szakértelmet barátságtalan, vagy kockázatos környezetben kell-e alkalmazni
- A rendszer szakértelme növeli a minőséget, teljesítményt
- A rendszer alkalmazható-e oktatásra
- A SZR megoldás gyorsabban fejleszthető-e mint a humán szakértői
- A SZR megbízhatóbb, pontosabb mint a humán szakértőé

CLIPS – egy SZR fejlesztői környezet

- **CLIPS = C Language Integrated Production System**
  - **Fejlesztették: NASA's Johnson Space Center (80-as évek közepén)**
  - **C nyelvet alkalmazták a megvalósításra, LISP szintaktikát követtek**
  - (C nyelv választása: hatékonyabb kód, LISP fordítóktól való függetlenség, más nyelveken írt modulokkal integrálás)
- **Alap változat: produkciós szabály interpreter.**
  - **Objektum orientált kiterjesztés: COOL = CLIPS Object- Oriented Language**

CLIPS környezet

Mi is ez valójában?

- Klasszikus szabályalapú szakértői rendszer
- CLIPS – szabályleíró nyelv
- CLIPS – kiegészítő komponensek
- Előrekövetkeztetés (CLIPS) vs. visszakövetkeztetés (pld. MY-CIN)

CLIPS környezet

A CLIPS előnyei:

- Magas-szintű interpreter
- Produkciós szabály interpreter
- Objektum orientált programozási nyelv
- LISP-szerű procedurális nyelv
- Számos különbözo platformon fut: UNIX, Linux,
- Windows, MacOS
- Egy public-domain jól dokumentált szoftver

CLIPS következtetés

- A CLIPS egy előre-következtető rendszer.
  - Illeszti a szabály feltétel részét (LHS) a munkamemória tartalmához, és végrehajtja az akció részét (RHS) a kiválasztott szabálynak.
  - Kiinduló tények -> konklúziók.
- Hátrakövetkeztetésnél (híres példa MYCIN) a cél előre ismert és a cél felderítéséhez ennek előfeltételeit vizsgáljuk.

  Ha egy előfeltételnek további előfeltételei vannak, akkor ezt részcélnak tekintjük és ugyanúgy felderítjük, mint a célt, mindaddig, amíg nincsenek további részcélok.
- Sok esetben használható mindkettő, vagy a két módszer kombinációja

Néhány fontos Clips utasítás

- Tény hozzáadása a ténybázishoz:

```
(assert (first-fact asserted))
```

- Szabály definiálása:

```
(defrule first_rule
    (first-fact asserted)
=>
    (assert (second-fact asserted)))
```

“Hello World” CLIPS-ben

```
(defrule start
  (initial-fact)
=>
  (printout t “Hello, world!” crlf))
```

Tények

- A tények egyszeru kifejezések, amelyek tartalmaznak egy relációnevet, és opcionálisan további rekeszeket. Példa:
  `(person (name "John"))`
- Gyakran használunk keret struktúrákat: `(deftemplate)` konstrukció

Példák tényekre

```
CLIPS> (deftemplate course "electives" (slot number))

CLIPS> (assert (course (number comp674)))
            (course (number    comp672)))
<Fact-1>


CLIPS> (facts)
f-0(course    (number comp674))
f-1(course    (number comp672))
For a total   of 2 facts

CLIPS> (retract 1) CLIPS> (facts)
f-0 (course (number comp674)) For a total of 1 fact
```

Tények törlése

- Tényeket lehet törölni, visszavonni

```
(retract <fact-index>)
(retract 2)
```

- Több tény törlése

```
(retract 1 2)
```

(deftemplate) példa

- Több deklaráció egyszerre:

```
(deffacts student-Ids
  (student (name Tarzan))
  (student (name Jane) (age 19)))
```

- Eredmény:

```
(student (name Tarzan) (age 18))
(student (name Jane) (age 19))
```

Szabályok (1)

- Feltételek (LeftHandSide) => Következmények (RightHandSide)
- Szintaktika:

```
(defrule <rule-name> [<comment>]
    [<declaration>] ;  salience
     <patterns>*        ; LHS, premises, patterns,
                                ; conditions, antecedent
   =>
     <actions>*)     ; RHS, actions, consequent
```

Szabályok (2)

- Példa:

```
(defrule class-A-fire-emergency
    (emergency fire)
=>
    (printout t "FIRE!!!" crlf))
```

Egy szabálynak több premisszája lehet:

```
(defrule class-B-fire-emergency
    (emergency fire)
    (fire-class B)
=>
    (printout t "Use carbon dioxide
extinguisher crlf))
```

Agenda

- Ha a LHS mintái illeszkednek az érvényes tényekhez, akkor a szabályt aktiválja a CLIPS és felrakja az agendára.
- Szabályok sorrendje:
  salience (*prioritás*).
- Ha nincs végrehajtható szabály, azaz az agenda üres, akkor a program leáll.
- **Refraction**: minden szabályt egyszer hajt végre a rendszer egy adott tényhalmazra

Salience (prioritás)

- Alaphelyzetben az agenda egy dinamikusan kezelt verem
- Alapértelmezés: (mélységi fabejárás) A legutoljára aktivált szabályt hajtja végre
- Salience, prioritás felülírja a dinamikus sorrendet.
- Alapértelmezett érték: 0.

Konfliktus feloldási stratégia

- Frissesség
  - Utoljára felkerült szabályok preferálása.
  - CLIPS time-tag-ek
- Specifikusság
  - Az a szabály, amely a legjobban illeszkedik az adott helyzetre preferált.
  - Hasznos, ha általános és kivételkezelo szabályokkal dolgozunk
- Cilkusmentesség
  - Ugyanazokra az adatokra csak egyszer hajtja végre a szabályokat.
  - Megakadályozza a ciklusokat
  - Használt szabályok újratöltése `(refresh)`)

Konfliktus feloldási stratégia

- Elsődleges a prioritás.
- 7 stratégia állítható:
  - The depth strategy
  - The breadth strategy
  - The simplicity strategy
  - The complexity strategy
  - The LEX strategy
  - The MEA strategy
  - Szabadon további stratégia definiálható
- szintaktika: (set-strategy <strategy>)

Változók

- Változó neve `?` Karakterrel kezdodik:
- Példa:

```
(course (number ?cmp))
```

- Változókat használunk
  - Mintaillesztés
  - I/O
  - Tények mutatói

Példák változókra

- ```
  (defrule grandfather
      (is-a-grandfather ?name)
  =>
      (assert (is-a-man ?name))
  )
  ```
- ```
      (defrule
      grandfather (is-  ?name)
      a-grandfather
  =>  (assert (is-a-father ?name))
      (assert (is-a-man ?name))
      (printout t ?name " is a grandfather" crlf)
  )
  ```

Példák tények törlésére

```
(defrule change-grandfather-fact
  ?old-fact <- (is-a-grandfather ?name)
=>
  (retract ?old-fact)
  (assert (has-a-grandchild ?name)
  (is-a-man ?name))
)
```

- Tények törlése: nem monoton logika
- Kényelmes dinamika modellezésére
  (állapot jelző tények hozzáadása és törlése)