# Bevezetés a CLIPS használatához

## *1.1 A CLIPS alapelemei*

Ebben a fejezetben egy LISP nyelven elkészített fuzzy szabályalapú szakértői rendszert mutatunk be példaként

A CLIPS (C Language Integrated Production System) egy produkciós rendszer keret, amelyet beágyazott szakértői alkalmazások fejlesztésének megkönnyítésére fejlesztettek ki. A rendszer jelentős támogatást és rugalmasságot ad, hisz a teljes rendszer forráskódja hozzáférhető C nyelven, így igényeinknek megfelelően módosíthatjuk, másrészt működés közben is támogatja új struktúrák, függvények dinamikus létrehozását. A nyelv támogatást nyújt az objektum-orientált paradigma használatához is.

A rendszerben jól elkülönülő struktúrák tárolják a tényeket, objektum példányokat, szabályokat, aktuálisan végrehajtható szabályokat.

### *1.1.1 Tények (facts)*

A CLIPS-ben aktuálisan létező tényeket a ténylistán találhatjuk. Tényeket betehetünk (assert), visszavonhatunk (retract), módosíthatunk (modify), és másolhatunk (duplicate).

A tényekre indexükkel vagy azonosítójukkal hivatkozhatunk. Az index minden új tény eltárolásakor egyesével növekszik (fact-n). A clear és reset parancsok a tényindexet visszaállítják 0-ra, és törlik a tényeket. Általában ezt praktikus használni, bár a tény címét - a tény azonosítót - is el lehet menteni.

A tényeknek két típusa van:

- rendezett, és
- nem rendezett.

A rendezett tények esetében a tényben szereplő szimbólumok sorrendje kötött. Nem rendezett tények esetén nincs ilyen megkötés, mert az egyes mezőkre nevekkel hivatkozhatunk.

Példa:

Rendezett: (the pump is on)
Nem rendezett: (point-mass (x-velocity 100) (y-velocity 100))

Kezdeti tényeket is meg lehet adni, vagyis olyan tényeket, amelyek a rendszer alapállapotában is igazak, jelen vannak. Ennek módja a deffact konstrukció használata.

### *1.1.2 Objektumok*

Ezek lehetnek:

- primitívek, illetve
- felhasználó által definiáltak.

Az egyedek egy - a ténylistához hasonló - egyed listán tárolódnak. A definstance függvénnyel definiált egyedek inicializáláskor illetve reset-kor automatikusan az instancelist-hez adódnak.

### *1.1.3 Globális változók*

Ahhoz, hogy bizonyos adatokat minden függvényben, minden nyelvi konstrukción belül "láthassunk", globális változókban kell tárolni azokat. Ehhez nyújt segítséget a defglobal konstrukció.

### *1.1.4 Szabályok*

A CLIPS-ben, mint más következtető rendszerekben is, a szabályok egy BAL OLDAL-ra és egy JOBB OLDAL-ra tagolódnak. A baloldalon fogalmazzuk meg a

feltételeket, a jobboldalon pedig az akciókat. A feltételrészben megadhatunk egyszerû mintát vagy függvényeket is.
A következtetőgép folyamatosan nyilvántartja, hogy mely szabályok alkalmazhatók.

A szabályok szerkezete:

(defrule név [deklaráció]
    <feltétel>*
    ===>
    <akció>* )

A rendszer az agendán tartja nyilván az alkalmazható szabályokat. Futtatáskor meg kell adni egy korlátot, hogy maximum hány szabályt alkalmazzon a rendszer. A következtetés akkor áll le, ha elértük ezt a korlátot, vagy ha nincs alkalmazható szabály, vagyis az agenda üres.

### *1.1.5 Mintaillesztés a szabályok feltétel részével*

A szabályok alkalmazhatósága az alapján dönthető el, hogy megvizsgáljuk, hogy a szabály baloldala illeszkedik-e a ténylistán található tényekre. A szabályok baloldalán az alábbi típusú minták jelenhetnek meg:

- <minta>
- <hozzárendelt minta>
- <not-feltétel>
- <and-feltétel>
- <or-feltétel>
- <test-feltétel>
- <exists-feltétel>
- <forall-feltéte>

Az illeszkedésnél megfogalmazhatjuk elvárásainkat konstansokkal, joker karakterekkel (wildcards) és változókkal. Konstans csak azonos konstansra illeszkedik.
Kétféle wildcard karaktert használhatunk: a ? és a $? karaktereket. A ? karakter egy tetszőleges elemre illeszkedik, míg a $? tetszőleges számú tetszőleges elemre illeszkedik. Hasonló módon illeszthetünk változókat is: ?változónév egy elemre illeszkedik, és az elem a változóhoz kötődik, míg a $?változónév tetszőleges számú elemre illeszkedik, és az illeszkedő elemek a változóhoz kötődnek.

A szabályok bal oldalán megfogalmazott feltételeket logikai operátorokkal összekapcsolhatjuk (&, |, ~). Ne felejtsük el, hogy a feltétel részek között eleve egy implicit ÉS kapcsolat van.
A feltételek lehetnek predikátumok is, de ezeket :-tal kell bevezetni.

Például:

```
(defrule example
        (data ?x&: (numberp ?x))
        ===>)
```

A feltételekben használhatjuk külső függvények visszaadott értékeit is. Ilyenkor a függvényhívást az = jellel kell bevezetni.

Például:

```
(defrule twice
        (data (x ?x) (y =(* 2 ?x)))
        ===>)
```

Egy egyszerû tesztet is elhelyezhetünk a szabály bal oldalán.

Példa:

```
(defrule példa
```

```
(daa ?x)
(value ?y)
(test (>= (abs (- ?y ?x)) 3))
===>)
```

### *1.1.6 A legfontosabb függvények*

- (numberp kifejezés)
- (lexemep kifejezés) T ha a kifejezés string vagy szimbólum
- (symbolp kifejezés)
- (multifieldp kifejezés) (a listp megfelelője)
- (eq kifejezés kifejezés+) egyenlőséget vizsgál (típusegyezést is)
- (= szám szám+)
- (and kifejezés+)
- (create$ kifejezés*) multifield értéket hoz létre (lista)
- (nth$ egész kifejezés)
- (member$ single-field multifield) az indexet adja vissza
- (printout stream kifejezés*) t standard output (crlf soremelés)
- (max kifejezés+)
- (min kifejezés+)
- (bind változó kifejezés*) értéket ad a változónak. Ha a kifejezést elhagyjuk, akkor a változónak visszaállítja a globális értékét.
- (if kifejezés then tevékenység* [else tevékenység*])
- (while kifejezés [do] tevékenység*)
- (assert tény+) tényeket helyez el a ténylistára
- (retract tényazonosító) tényt töröl. Az azonosító lehet tény index, vagy cím.
- (help [témakör])