# INTRODUCERE

**CLIPS** (acronim provenit din "**C** **L**anguage **I**ntegrated **P**roduction **S**ystem") este un limbaj de nivel înalt ideal pentru realizarea sistemelor expert. Proiectat inițial de Software Technology Branch (STB), în cadrul laboratoarelor NASA Johnson Space Center, cu variante distribuite actualmente gratuit prin INTERNET (versiunea FuzzyCLIPS, versiunea CLIPS 6.0) acest mediu de programare a fost dezvoltat şi în variante integrate sub JAVA, de exemplu Jess, sau sub forma unor produse comerciale mult mai performante, CLIPS RII. In cazul variantei CLIPS-ului implementate în JAVA, marele avantaj constă în compatibilitatea retrogradă a programelor (orice aplicație scrisă în JESS, va putea fi executată cu mici modificări şi în CLIPS). De remarcat faptul că îmbunătățirile performanțelor de viteză şi interfațare, s-au făcut cu respectarea generală a aceloraşi principii de programare.

Familiarizarea cu programarea în CLIPS, limbaj bazat pe reguli, în care se oferă, mai recent, suport şi pentru programarea orientată obiect şi pentru programarea procedurală, constituie obiectul cărții de față. Ca şi pentru alte generatoare de sisteme expert VP-Expert, EXSYS, GURU etc. precum şi pentru limbajele declarative cum este şi PROLOG-ul, programarea într-un astfel de limbaj este total diferită de cea dintr-un limbaj clasic (de tip procedural - C, PASCAL, etc.).

Față de programarea procedurală în care programatorul trebuie să cunoască rezolvarea problemei şi să descrie această rezolvare într-o suită de instrucțiuni executabile (proceduri), într-o programare logică, de rezolvarea problemei nu este responsabil programatorul, acesta descriind problema cât mai exact şi mai complet (în manieră declarativă) şi lăsând sistemul însuşi să caute şi să genereze soluții. Programatorul va declara legile (regulile) problemei şi datele concrete ale acesteia, iar sistemul va căuta, aplicând toate legile disponibile datelor concrete (faptelor), să găsească maximul de concluzii posibile. Pe parcurs, din numeroasele concluzii deduse, se va ajunge la una sau mai multe concluzii ce vor corespunde cerințelor problemei, acestea fiind şi soluțiile problemei. De reținut este faptul că un limbaj precum CLIPS-ul va căuta şi afişa toate soluțiile posibile.

Pentru a fi mai expliciți, ne putem raporta la modul uzual de a rezolva o problemă. Pentru început se vor căuta toate legile, formulele şi axiomele care ar putea ajuta la rezolvarea problemei. Dintr-un astfel de set izolat se vor aplica formule şi legi, obținându-se date noi ce îmbogățesc baza de cunoştințe. Pe baza unor astfel de deducții se ajunge (nu neapărat din prima încercare) la o soluție finală, aceasta fiind şi ea o relație deductibilă pornind de la setul de date concrete, dacă se elimină încercările nereuşite şi datele intermediare. O astfel de descriere este procedurală, fiind limitată de abilitățile şi experiența

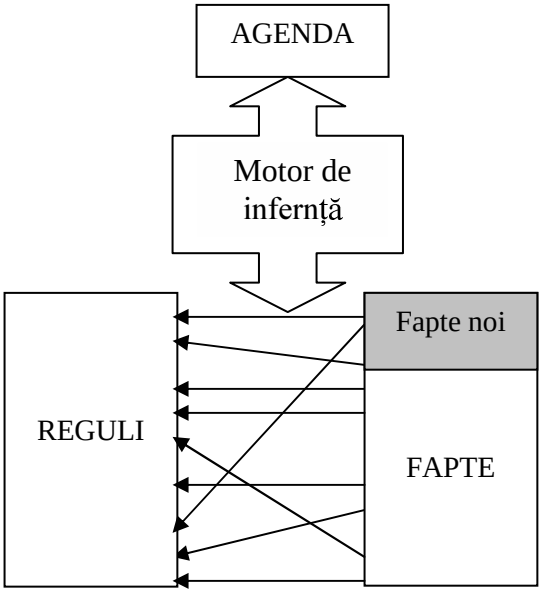
programatorului. Dacă, de exemplu, se doreşte programarea unui robot (sau a unei maşini) care să ia decizii în funcţie de context, este recomandată folosirea unui limbaj neprocedural. Un simplu robot, care trebuie să facă curat într-o cameră, trebuie să ocolească mobilierul sau alte obstacole, să deosebească un obiect util de unul care trebuie aruncat şi să ia multe alte decizii. In domeniul vast al "Inteligenţei Artificiale" pot fi găsite o mulţime de aplicaţii, care pot fi rezolvate foarte simplu de limbajele de programare logică.

Programarea într-un limbaj bazat pe reguli încearcă să simuleze maniera de gândire a omului. Se specifică doar datele, legile şi formulele problemei, sistemul căutând soluţiile. Corectitudinea şi completitudinea acestora este strâns legată de cea a datelor iniţiale (a bazei iniţiale de cunoştinţe). Dacă ne gândim la un labirint, modul de rezolvare a unei probleme pentru un astfel de limbaj, constă în încercarea tuturor căilor posibile, întoarcerea din drum când se găseşte o cale închisă, memorarea tuturor punctelor de întoarcere, a tuturor căilor de investigaţie şi astfel în mod normal găsirea tuturor soluţiilor posibile.

Partea orientată obiect numită şi COOL (CLIPS Object-Oriented Language) este o combinaţie între trăsăturile găsite şi la alte limbaje orientate obiect, cum ar fi CLOS (Common Lisp Object System) şi SMALLTALK, la care s-au adus contribuţii noi. Partea procedurală a CLIPS-ului este asemănătoare cu limbajele C, Ada, Pascal şi foarte apropiate sintactic de LISP. Versiunea 5.0 a CLIPS-ului este cea care introduce suportul de programare procedurală şi orientată obiect; versiunea 5.1 aduce cu ea suportul software pentru programarea sub Windows, MS-DOS şi interfeţele Macintosh.

Shell-ul unui limbaj bazat pe reguli cum este CLIPS-ul sau oricare alt sistem expert are practic trei componente de bază: *lista de fapte* (care conţine datele ce vor interacţiona cu programul); *baza de cunoştinţe* (care conţine toate regulile); *motorul de inferenţă* (care controlează execuţia generală a regulilor programului). CLIPS-ul este destinat dezvoltării de software care să ajute la o apropiere de modelul de cunoaştere şi expertiză umană. Avem trei modalităţi de reprezentare a cunoaşterii în CLIPS: *regula* (reprezintă cunoaşterea euristică bazată pe experienţă); *funcţiile* (reprezintă cunoaşterea procedurală); *programarea orientată obiect* (cerută tot de cunoaşterea procedurală). Software-ul realizat în CLIPS, poate să folosească fie doar reguli, fie doar obiecte, fie o combinaţie între acestea. Motorul de inferenţă este cel care va decide care dintre reguli va fi executată şi când, deşi se va vedea că utilizatorul are posibilitatea să stabilească pentru fiecare regulă o prioritate şi chiar un moment de timp la care să înceapă execuţia regulii. Datele - reprezentate în CLIPS prin fapte - sunt cele care vor stimula execuţia unei reguli, funcţii, sau a unui obiect. Ca toate generatoarele bazate pe reguli, CLIPS-ul utilizează un algoritm foarte eficient de comparare a faptelor pentru a găsi din baza de fapte pe acelea care se suprapun exact peste enunţurile din partea de premisă a regulilor, numit şi RETE Patern

Matching Algorithm. Cunoaşterea acestuia uşurează scrierea regulilor în CLIPS şi reduce redundanța temporală specifică sistemelor bazate pe reguli, oferind o similaritate structurală pentru scrierea regulilor şi oprirea în cazul unei potriviri parțiale (nu se mai verifică restul condițiilor în momentul în care o condiție este falsă).




După cum se vede în schemă, în procesul de comparare, motorul de inferență examinează fiecare regulă şi caută în baza de fapte dacă enunțurile din premisa regulii sunt satisfăcute, caz în care regula este introdusă în agendă (memoria de lucru în care sunt activate regulile). Algoritmul RETE calculează de la un ciclu la altul doar schimbările necesare asupra bazei de date, pentru faptele şterse sau adăugate eliminând sau activând regulile pentru care premisele sunt satisfăcute (sau nu). Acest lucru se realizează prin salvarea stării procesului inferențial de la un ciclu la altul şi recalcularea numai a schimbărilor specifice acestor stări. Dacă sunt analizate numai schimbările recente din lista de fapte, atunci fiecărei reguli trebuie să i se reamintească ce s-a întâmplat în ciclul anterior. Acest tip de prelucrare a informației de stare, care indică faptele comparate în cadrul unei reguli în ciclul anterior, se mai numeşte şi *comparare parțială* (partial matching). Mai există şi un alt tip de prelucrare a informației de stare numit şi *comparare totală* (patern matching), atunci când un fapt satisface în întregime un enunț din premisa oricărei reguli, fără a se lua în calcul variabilele din celelalte enunțuri, care ar putea restricționa procesul de comparare.

Algoritmul RETE este mare consumator de memorie, este foarte scump, dar îmbunătățeşte eficiența sistemelor expert bazate pe reguli. După cum s-a

mai spus prezintă avantajul *similarității structurale*, adică pentru mai multe reguli care conțin enunțuri sau grupuri de enunțuri similare, algoritmul RETE le pune împreună, realizând calculele o singură dată.

În prezent acest limbaj este îmbunătățit continuu, astfel încât noul algoritm **RT II** (soft comercial), prezentat de producători ca având viteza de aproximtiv 100 ori mai mare decât algoritmul RT obişnuit, duce la realizarea unui motor de căutare mult mai rapid.

Datorită integrării totale cu alte limbaje asemănătoare cu C şi Ada, CLIPS-ul poate fi apelat de un alt limbaj procedural, care, după execuția funcțiilor, va returna controlul înapoi, programului ce a realizat apelul.Codul procedural poate defini şi funcții externe, care, după ce vor fi apelate de CLIPS şi îşi vor termina execuția, vor returna controlul înapoi CLIPS-ului.

In primele capitole ale acestei lucrări sunt prezentate noțiuni introductive despre limbaj, însoțite pe parcurs de un număr important de exemple. Partea teoretică, realizată în conformitate cu documentația, respectiv help-ul limbajului, şi alte surse de documentație menționate în bibliografie, este însoțită de explicații consistente şi aplicații menite să uşureze procesul de învățare. Spre finalul cărții vor fi exemplificate diverse rezolvări ale unor probleme din domeniul inteligenței artificiale, pentru care poate fi aplicată cu succes programarea declarativă.

Lucrarea prezentă este în excelență un ghid de aplicații realizate în CLIPS adresându-se în general specialiştilor care vor să realizeze propriile lor sisteme expert şi în special studenților şi elevilor care vor însuşi astfel bazele programării pe reguli. CLIPS-ul este un limbaj susceptibil a fi îmbunătățit continuu, acest lucru constituind de fapt o invitație şi o provocare. Specialiştii în inteligență artificială şi cadrele didactice (pentru nivel universitar şi liceal) vor găsi în această lucrare un real instrument de lucru, abundența exemplelor, a explicațiilor şi a exercițiilor rezolvate transformând în accesibil un domeniu aparent arid.

Aceasta este prima dintr-o serie de astfel de lucrări editate de acelaşi colectiv, menite să ajute la formarea tinerilor specialişti în inteligență artificială, prin aprofundarea unor domenii actuale, de interes, în cercetarea ştiințifică.