6

Modificarea dinamică a programului

În acest capitol se introduc unele predicate predefinite care acționează asupra programului, adăugând, eliminând şi verificând prezența clauzelor.

S-ar crea impresia că programele sunt un obiect static, îndată ce Prologul le-a încărcat spre executare. Adică, dacă o mulțime de fapte şi reguli au fost "consultate" de sistem, atunci această mulțime nu poate fi schimbată în procesul executării programului şi unica excepție ar constitui-o doar "reconsultarea" lor.

Această presupunere nu este adevărată: Prologul permite inspectarea şi modificarea dinamică a bazei de fapte şi reguli. El ne oferă unele predicate predefinite de căutare, adăugare şi eliminare a clauzelor. Aceste predicate se numesc de ordinul doi, deoarece au în calitate de argumente alte predicate. Odată modificate, predicatele-argumente îşi pierd forma inițială şi îşi păstrează forma nouă până nu sunt modificate din nou.

Dar programele ce conțin predicate de adăugare şi eliminare a clauzelor pot avea efecte colaterale. Aceste predicate pot cauza schimbări ce pot afecta executarea de mai departe a programului. Scopurile, ce reuşesc acum într-un punct, mai târziu pot eşua sau pot reuşi în diferite momente de timp. Dificultățile introduse de efectele colaterale sunt bine cunoscute şi programele, ce utilizează aceste tehnici, sunt greu lizibile, greu de depănat şi asistat. Din cele spuse reiese că utilizarea predicatelor de modificare dinamică a programului nu este recomandată programatorilor neexperimentați.

6.1. Căutarea unei clauze

Predicatul predefinit

```
clause(Cap,Corp).
```

verifică existența în baza de date a unei clauze cu capul *Cap* şi corpul *Corp*. Variabila *Cap* trebuie să fie instanțiată, în caz contrar predicatul *clause/2* eşuează. Acest predicat caută prima clauză, capul căreia se poate unifica cu *Cap,* apoi încearcă să unifice corpul clauzei cu *Corp*. Dacă clauza este un fapt, atunci *Corp* se unifică cu *true*. Dacă Corp este o conjuncție de scopuri, se utilizează paranteze suplimentare pentru a evita ca scopurile să fie considerate parametri suplimentari în predicatul *clause/2*.

Backtrackingul nu poate fi utilizat pentru a găsi toate clauzele. Motivul e că variabila *Cap* trebuie să fie instanțiată. În afară de aceasta, clauzele nu pot fi găsite prin unificarea corpului, fiindcă corpul *Corp* trebuie să fie o variabilă neinstanțiată. Aceste incomodități, însă, pot fi ocolite prin nişte tehnici artificiale, care depăşesc cadrul acestei lucrări. Menționăm, că acest predicat nu este în Turbo Prolog şi Visual Prolog.

Fie pachetul de clauze

```
pred(arg,arg).
pred(arg1,arg2):-pr(arg1,arg2,arg3).
```

atunci vom avea

```
?-clause(pred(arg,arg))
no

?-clause(pred(arg,arg),true)
yes

?-clause(predicat(arg))
no

?-clause(pred(arg1,arg2),pr(arg1,arg2,arg3))
yes

?-clause(pred(_,_), X)
X=true;
X=pr(argl,arg2,arg3)
```

6.2. Adăugarea unei clauze

Clauzele pot fi adăugate într-un program, utilizând predicatele unare *assert/1*, *asserta/1* şi *assertz/1* al căror argument nu reprezintă un termen variabilă. Aceste predicate reuşesc întotdeauna, dar nu pot fi resatisfăcute în cazul backtackingului.

Dacă se utilizează *assert/1,* poziția clauzei noi în program nu este specificată. Clauza va fi prima în pachet, dacă se foloseşte *asserta/1* şi ultima, dacă *assertz/1*. Predicatele respective vor avea o eroare de executare dacă argumentul, fiind un termen corect, în acelaşi timp, nu reprezintă o clauză sau capul clauzei indicate are în calitate de functor principal un nume de predicat de sistem. Dacă coada e constituită dintr-o conjuncție de scopuri, atunci clauza se ia în paranteze pentru a fi tratată ca un singur termen.

Unele exemple: clauza

```
assert(p(mic)).
```

adaugă un fapt şi este echivalentă clauzei

```
assert((p(mic):-true)).
```

iar clauza

```
assert((p(X):-q(X,Y),r(Y))).
```

adaugă o regulă. Însă clauza

```
assert((integer(X):-X1=X+1,q(X1)).
```

provoacă eroare deoarece integer/1 e un predicat de sistem, iar clauza

```
assert(1).
```

provoacă o eroare deoarece un întreg nu este o clauză validă.

După executarea întrebării

```
?-assert(proc(b)),assertz(proc(c)),
   asserta(proc(a)), assertz(proc(d)).
```

întrebarea

```
?- proc(X), write(X,", "),fail.
```

va produce rezultatul

```
a, b, c, d
```

6.3. Eliminarea unei clauze

Executarea clauzei

```
retract(Clauza).
```

determină căutarea în program a unei clauze ce se unifică cu variabila instanțiată *Clauza*. Dacă clauza este găsită, atunci ea este eliminată din program.

Scopul *retract(Clauza)* poate fi resatisfăcut în timpul backtrackingului, adică el este unul din predicatele predifinite de natură nondeterministă. Variabila *Clauza* poate fi instanțiată din nou parțial. Aceasta permite utilizarea predicatului *retract/1* în mod nondetreminist pentru eliminarea în creştere din program a clauzelor ce se unifică cu argumentul *Clauza*.

Urmează unele exemple de aplicare a predicatului predefinit *retract/1,* care pot avea loc în diverse versiuni Prolog. Menționăm, însă, că în Turbo Prolog şi Visual Prolog pot fi modificate numai faptele şi nu regulile. În Sicstus (şi alte versiuni) Prolog pentru a manipula predicatele cu ajutorul *assert/1* şi *retract/1* ele mai întâi se declară dinamice (dynamic).

Conjuncția de scopuri

```
retract((raspuns(atom_1,atom_2,_):-
       gasire(_,atom_1),
       cautare(atom_2,a))),fail.
```

permite eliminarea tuturor clauzelor ale căror primele două argumente ale capului se unifică cu atomii *atom_1* şi *atom_2* (al treilea argument nu este specificat) şi ale căror cozi sunt formate din subscopurile indicate.

Pentru eliminarea tuturor clauzelor specificate într-o listă poate fi folosit programul

```
eliminare_clauza([]).
eliminare_clauza([Cl|Cl_ramase]):-
       retract(Cl),
       eliminare_clauza(Cl_ramase).
```

Programul, care ar elimina toate clauzele ce au capul specificat de argumentul de intrare,

```
eliminare_clauze(Cap):-retract(Cap),fail.
eliminare_clauze(Cap):-retract((Cap:-
_)),fail.
eliminare_clauze(_).
```

este un program iterativ cu un backtraking forțat de predicatul predefinit *fail/0* care utilizează natura nodeterministă a predicatului *retract/1*. Ultima clauza ne asigură succesul în cazul absenței clauzei indicate de *Cap* sau în cazul când toate clauzele deja au fost eliminate.

În Turbo Prolog există un predicat predefinit pentru eliminarea tuturor faptelor ce se unifică cu argumentul specificat:

```
retractall(Fapt).
```