C4

Limbajul Haskell – de ce ?

Introducem pe lista instrumentelor utile pentru cercetarea interpretoarelor adaptabile ale limbajelor extensibile un limbaj mai puțin folosit în România: Haskell. Acest capitol explică de ce.

O serie de considerente pragmatice ne-au condus la a considera limbajul Haskell (mai exact nou standardizatul Haskell 98) drept un bun instrument de laborator pentru realizarea studiilor și prototipurilor și interpretoarelor adaptabile.

Rezumând: Haskell este un limbaj funcțional pur, de uz general (un veritabil C++ al programării funcționale), numit astfel după numele logicianului Haskell B. Curry. A fost proiectat în 1988 de un comitet de 15 membri care doreau să înlocuiască mulțimea de limbaje funcționale ale perioadei cu un singur limbaj. Acesta urma să:

- fie potrivit pentru învățământ, cercetare și aplicații inclusiv în construcția de sisteme soft de mari dimensiuni (care reclamă module sau clase de obiecte)
- să fie disponibil free, gratuit, liber accesibil
- să implementeze concepte care erau considerate *valoroase* de către comitet (și de comunitatea dezvoltatorilor)
- să poată deveni un limbaj standard al programării funcționale,

reducând diversitatea de limbaje care începea să împartă, în acea perioadă, comunitatea programatorilor în "religii", în tabere de programatori.

Dintre trăsăturile sale sunt amintite drept caracteristice următoarele, pe care le-am găsit enumerate foarte concis într-un articol al profesorului A. Aaby, Haskell Tutorial, disponibil la adresa :

http://www.cs.wwc.edu/~cs_dept/KU/PR/Haskell.html

Pe scurt, limbajul Haskell

- permite utilizarea de funcții de nivel înalt
- are o semantică nestrictă, cu lazy-evaluation
- permite tipizarea polimorfică, mai exact polimorfismul parametric
- permite construirea de tipuri utilizator (și de constructori parametrizați pentru tipurile utilizator)
- permite realizarea de module "type-safe"
- funcțiile se scriu în manieră "curried function" altfel spus admite scrierea fără paranteze
- folosește pattern-matching-ul, adică permite utilizarea de șabloane și are un mecanism de unificare pentru potrivirea acestora
- permite descrierea implicită a mulțimilor reprezentate ca liste – așa numitele "list comprehensions"
- mulțimea operatorilor este extensibilă
- oferă un bogat set de tipuri primitive
- modulele pot fi recursive, în sensul recursivității indirecte

Zece ani mai târziu a devenit standard noua versiune, Haskell 98, iar utilizatorii au fost sfătuiți să nu mai folosească versiunile anterioare.

Actualmente, utilizatorul limbajului Haskell poate conta de asemenea pe:

- programarea în stil imperativ (folosind **do**-notația și monada de

I/O) sau programarea cu sintaxă imperativă folosind **do**-notația și o altă monadă
- monade și monade cu zerou și operator plus
- noțiunea de clasă, noțiunea de instanță a unei clase
- tipuri de date recursive, inclusiv arbori (cu frunze de un tip parametrizat polimorfic)
- liste infinite

Poate fi notat ți faptul că un limbaj așa cum este Haskell poate fi folosit cu succes la "generic programming", programarea generică.

Cerințele la adresa unui limbaj destinat construcției de interpretoare adaptabile pentru limbaje extensibile sunt:

- să permită combinarea efectelor funcțiilor de analiză sintactică (sau chiar a parserelor) și funcțiilor semantice
- să permită definirea de modele de calcul modulare, prin compuneri similare cu cele din lucrarea "Semantic Lego" [Esp-95].
- să permită definirea de structuri de date recursive și de arbori cu conținut aprioric necunoscut în frunze, care poate fi de diverse tipuri
- să permită folosirea recursivității – componentele unui interpretor (compilator) fiind strâns interconectate prin recursivitate indirectă, mutuală. Gramaticile și semanticile limbajelor sunt și unele și altele recursive (din cauza modului de definire a expresiilor și instrucț iunilor imbricate)
- să permită modularizarea aplicației sau măcar structurarea ei în clase
- să permită scrierea unui cadru ("framework") în care să se combine module care conțin simultan specificații pentru: modelul

de calcul, sintaxă, semantică și eventual alte elemente

- să permită combinarea efectelor funcțiilor de analiză sintactică (a parserelor) precum și a funcțiilor semantice.

Limbajul Haskell îndeplinește toate aceste cerințe. În capitolele următoare și în exemplele anexate vom arăta cum se poate el folosi la realizarea de interpretoare, fie monolitice, neadaptabile fie adaptabile.