publikacje

Programowanie w języku systemów eksperckich CLIPS (1)

Podstawy składni języka CLIPS

Tomasz Pytlak

Samodzielna Pracownia Sztucznej Inteligencji

Instytut Podstaw Elektroniki

Politechnika Warszawska

Język CLIPS jest typowym przedstawicielem języków systemów eksperckich. Został opracowany przez Sekcję Sztucznej Inteligencji w NASA. Nazwa CLIPS pochodzi od słów *C Language Integrated Production System*. Ze względu na wiele zalet oraz łatwość integracji ze środowiskiem języka C, język ten wypiera ostatnio inne narzędzia służące do realizacji systemów eksperckich. Wbudowane mechanizmy przetwarzania list, mechanizmy wnioskowania i samomodyfikacji stawiają CLIPS-a przed językami sztucznej inteligencji, takimi jak LISP czy Prolog. Język ten znajduje szerokie zastosowanie, głównie ze względu na rozbudowaną bibliotekę, łatwość programowania i niski koszt. CLIPS z powodzeniem został użyty do realizacji ponad 4000 systemów eksperckich m.in. w obronności, we wspomaganiu prac biurowych, w weryfikacji kontraktów na szczeblu rządowym, we wspomaganiu prac naukowych, w zarządzaniu, w przemyśle, w pracach wielu małych firm prywatnych. Język CLIPS jest reklamowany przez NASA jako najlepszy produkt umożliwiający realizację systemów eksperckich. Również w Polsce zostały opracowane pierwsze systemy eksperckie w tym języku.

Niniejsza praca otwiera krótki cykl artykułów poświęconych językowi CLIPS. Cykl ten został tak pomyślany, aby Czytelnik mógł opracować swój własny prosty system ekspercki już po przeczytaniu pierwszej części, a następnie dobudowywać do niego coraz bardziej skomplikowane struktury. W drugim artykule z tej serii Czytelnik pozna zasady sterowania wnioskowaniem w języku CLIPS poparte kilkoma przykładami. W trzecim zostaną wprowadzone elementy programowania obiektowego. Wreszcie, na zakończenie cyklu, przedstawimy przykłady zaawansowanych technik programowania i komunikacji systemu eksperckiego opracowanego w języku CLIPS, z programami opracowanymi w innych językach i środowiskach. Jednak podstawowym środowiskiem uruchomieniowym, pozwalającym napisać i przetestować system ekspercki jest środowisko windows/menu/mouse[1].

Elementy składni języka CLIPS

System ekspercki można uruchomić na jeden z trzech sposobów:

a) w trybie interakcyjnym,
b) korzystając z edytora,
c) dołączając gotowy produkt do programu nadrzędnego.

[1] Po wywołaniu programu na ekranie pojawia się okienkowy system podpowiedzi. Wyjściowa pozycja kursora znajduje się w okienku edycyjnym. Tutaj można wprowadzić konstuktory systemu eksperckiego. Jest to również podstawowe okno pozwalające na uruchomienie własnego systemu eksperckiego.

Mgr inż. Tomasz PYTLAK ukończył w 1986 r. Wydział Elektroniki Politechniki Waszawskiej, kierunek Aparatura Elektroniczna. Od trzech lat jest członkiem Samodzielnej Pracowni Sztucznej Inteligencji w Instytucie Podstaw Elektroniki PW. Jest autorem wielu opracowań i programów z dziedziny sztucznej inteligencji, systemów eksperckich i doradczych.

Zastosowaniom trzeciego ze sposobów poświęcimy jeden z dalszych artykułów cyklu.

Obecnie zajmijmy się uruchomieniem prostych funkcji i reguł w **trybie interakcyjnym.** Interakcyjny sposób pracy dotyczy raczej prostych implementacji. Datego też stosuje się go, na przykład, do kontroli pracy systemu eksperckiego, zapisanego wcześniej w pamięci. Przykład wywołania funkcji w trybie interakcyjnym może być następujący:

```
CLIPS>(+4 5)
9
CLIPS>
```

W tym wypadku po znaku zachęty wywołuje się funkcję dodawania, która jest zapisana w nawiasach. Należy zwrócić uwagę, że – jak każda funkcja arytmetyczna – jest ona **zapisana w postaci prefiksowej.** Jest to generalna zasada tworzenia oprogramowania w języku CLIPS. Użytkownik zechce sprawdzić następujące przykłady funkcji:

```
CLIPS>(eq 3 4)                          ;porównanie,
                                        ;nierówne: neq
FALSE
CLIPS>(> 5 3 1)                         ;inne często używane:
                                        ;<, <=, >=
TRUE
CLIPS>(create$ młot pilnik piła)        ;zwraca zmienną
                                        ;typu multiple[2)]
(młot pilnik piła)
CLIPS>(nth$ 3 (create$ a b c d e f))
c                                       ;zwraca element wskazany
                                        ;w zmiennej typu multiple
CLIPS>(member$ c (create$ a b c d e f))
3                                       ;zwraca numer elementu
                                        ;w zmiennej typu multiple
CLIPS>(str-cat „abc" „def")             ;łączy argumenty
                                        ;w jeden łańcuch
abcdef
CLIPS>(+ 2 3.0 5)                       ;dodawanie
10.0
CLIPS>(- 12 3 4)                        ;odejmowanie
5
CLIPS>(* 3.1 4.7)                       ;mnożenie
14.57
CLIPS>(/ 24 3 4)                        ;dzielenie, dzielenie
                                        ;w zbiorze liczb
                                        ;naturalnych: div
2.0
CLIPS>(max 2 3 5 4 1)                   ;maksimum, minimum:
                                        ;min, moduł: abs
5
CLIPS>
```

Dostępne są również funkcje jednoargumentowe, zwracające liczbę rzeczywistą, jak np. arccos: **acos**, arccos hiperboliczny: **acosh, cos**, pierwiastek kwadratowy: **sqrt**, logarytm dziesiętny: **log10**, np.

```
CLIPS>(round 3.6)                       ;zaokrąglenie
4
CLIPS>
```

Bardzo przydatne jest deklarowanie zmiennych globalnych. Sekwencja instrukcji, wykorzystująca zmienną globalną oraz funkcję przypisywania jej wartości, jest następująca:

```
CLIPS>(defglobal ?*x* = 5)              ;definicja zmiennej
                                        ;globalnej
CLIPS>?*x*
5
CLIPS>(bind ?*x* (** 3 2))              ;uaktualnienie wartości,
                                        ;potęgowanie: **
9
CLIPS>?*x*
9
CLIPS>
```

Instrukcja **bind** służy do przypisywania zmiennym wartości, a jej ogólna składnia wygląda następująco:

(**bind** <*zmienna*> <*wyrażenie*>)

Pełny zestaw instrukcji można znaleźć w CLIPS Reference Manual, dostępnym w zbiorach instalacyjnych.

Do zapamiętania struktury systemów eksperckich, gdzie wykorzystuje się sekwencje instrukcji, służy edytor. Wywoływany jest on po wybraniu opcji **File:**
Alt-F, Editor.

W tym miejscu warto zauważyć, że opcja **Editor** znajduje się w grupie innych, pozwalających na ładowanie tekstu programu do bufora, czy zapamiętywanie tekstu w pliku zewnętrznym. Opcja **Editor** pozwala na manipulacje tekstem programu, podobnie jak to jest w innych znanych edytorach. Po zapamiętaniu tekstu programu należy wrócić do środowiska CLIPS i załadować program w celu uruchomienia go. Służy do tego opcja:
Alt-F, Load Constructs
w wyniku której w okienku dialogowym pojawia się tekst programu poddany kompilacji. Na końcu tekstu znajdzie się informacja o poprawności tekstu, sygnalizowana napisem True w wypadku braku błędów, bądź False w wypadku błędów w tekście programu.

Podstawowa konstrukcja systemu eksperckiego w języku CLIPS

Program, napisany w języku CLPS, składa się z *konstruktorów*. W języku CLIPS dostępnych jest 11 konstruktorów: **defmodule, defrule, deffacts, deftemplate, defglobal, deffunction, defclass, definstances, defmessagehandler, defgeneric, defmethod.** Do napisania prostych systemów potrzebne są dwa: **deffacts** i **defrule**. O innych wspomnimy, opisując bardziej zaawansowane metody programowania. Generalną zasadą jest umieszczenie całego konstruktora w nawiasach okrągłych. Podobnie, jak w innych językach, tekst zapisany między znakiem; a końcem linii traktowany jest jako komentarz.

Konstruktor **deffacts** służy do określenia faktów opisujących bazę wiedzy:

(deffacts <*nazwa-grupy-faktów*> [<*komentarz*>] <RHS-*wzorzec*>)

Cały konstruktor jest ujęty w nawiasy. Napis **deffacts** należy do symboli zastrzeżonych i oznacza początek definicji grupy faktów o wspólnej nazwie, zawartej w drugim atomie konstruktora. Listę faktów umieszcza się w polu *RHS-wzorzec*. Komentarz nie jest obowiązkowy.

Jako przykład można podać następującą listę faktów zapamiętaną w pliku zewnętrznym przy wykorzystaniu poleceń edytora:

```
(deffacts przyczyny-pożaru
(temperatura bardzo wysoka)(dym bardzo gęsty)(wiatr bardzo silny))
```

Powyższa deklaracja określa pewne zjawiska atmosferyczne. Interpretacja tych zjawisk może doprowadzić

[2)] Takiej deklaracji używa się do definiowania zmiennych zawierających wiele elementów różnego typu.

do określenia przyczyn lub przesłanek związanych ze skutkiem ich działania, np. pożarem. Lista faktów może być dowolnie długa. Zbiór list określający bazę wiedzy może być dowolnie liczny. Fakty, zgodnie z deklaracją, umieszczane są na *stosie faktów*. Fakty, zadeklarowane zgodnie z instrukcją **deffacts**, są rekonstruowane po każdym poleceniu systemowym **reset** służącym do wyzerowania stanów wewnętrznych maszyny wnioskującej.

Za interpretacje faktów odpowiedzialny jest *zbiór reguł*. *Reguła* jest podstawowym konstruktorem języka CLIPS, odpowiedzialnym za reprezentację wiedzy. Konstruktor reguły jest następujący:

```
(defrule <nazwa-reguły> [<komentarz>]
        [<deklaracja>] ;specyfikacje reguły
        <warunek>      ;Left-Hand Side (LHS)-pole warunków
        =>
        <akcja>)       ;Right-Hand Side (RHS) –
                       ;lista wykonywanych akcji
```

Lewa strona reguły składa się z *warunków* (*conditional-elements* – **Ces**). Jeżeli warunki te są spełnione, to aktywowane są komendy z części RHS reguły. Separatorem między warunkami (Ces) a RHS jest znak = >. Przykładem mogą być reguły wykorzystujące wcześniej zadeklarowane fakty:

```
(defrule reguła-alarmowa-1 „To przykład prostej reguły"
        (temperatura bardzo wysoka)
        (wiatr bardzo silny)
        =>
        (printout t „Pożar rozprzestrzenia się")
        (assert (pożar rozprzestrzenia się)))
(defrule reguła-alarmowa-2
        (temperatura ~bardzo duża  ;zaprzeczenie stwierdzenia: ~
        (deszcz dość duży)
        =>
        (printout t „Zmniejszona gotowość straży pożarnej")
        (assert (zmniejszona gotowość straży-pożarnej)))
```

W tekście przedstawiono inną formę deklaracji faktu za pomocą słowa **assert**. Deklaracja ta nie zezwala na rekonstrukcję faktu po poleceniu **reset**.

Zbiór reguł umieszczonych w pliku, określającym wiedzę systemu eksperckiego, pracując kolektywnie określa sposób rozwiązania problemu. Reguły powinny być zatem opracowane zgodnie z ogólnymi zasadami, ujętymi w formie następujących zaleceń:

- reguły bardzo szczegółowe powinny znajdować się na początku stosu reguł,
- reguły ogólne, oparte na schematach, na końcu stosu reguł,
- reguły mające taką samą postać (LHS lub RHS) powinny być połączone w jedną.

Powyższe konstruktory, zapamiętane w pliku zewnętrznym, można załadować do pamięci komputera. Preferowana sekwencja instrukcji jest następująca:

```
CLIPS>(clear)                    ;zerowanie pamięci
                                 ;komputera
CLIPS>(load „nazwa-pliku")       ;załadowanie
                                 ;konstruktorów do pamięci
CLIPS>(reset)                    ;zerowanie stanu maszyny
                                 ;i licznika poleceń
CLIPS>(run)                      ;uruchomienie systemu
CLIPS>Pożar rozprzestrzenia się  ;komunikat, który
                                 ;powinien pojawić się
                                 ;na ekranie
```

Konsekwencją *odpalenia* reguły *reguła-alarmowa-1* jest dopisanie do bazy wiedzy faktu (pożar rozprzestrzenia się)

Reguła ta została odpalona, ponieważ konstruktorem **deffacts** wprowadziliśmy do pamięci komputera fakty zgodne z jej LHS.

Istnieją dwa polecenia, służące do przeglądania listy faktów i reguł:

```
CLIPS>(facts)
f-0  (initial-fact)
f-1  (temperatura bardzo wysoka)
f-2  (dym bardzo gęsty)
f-3  (wiatr bardzo silny)
f-4  (pożar rozprzestrzenia się)
For a total of 4 facts.
CLIPS>(rules)
rule-1  reguła-alarmowa-1
rule-2  reguła-alarmowa-2
CLIPS>
```

Polecenia systemowe można wypisywać na ekranie w nawiasach lub korzystać z edytora poleceń **Execution**. Oprócz wspomnianych, do wyboru są jeszcze: **Step**, pozwalająca na pracę krokową (w trybie interakcyjnym polecenie **run** *n*, gdzie *n* – liczba odpalonych poleceń); **Watch**, pozwalająca na śledzenie każdej operacji wykonanej w trakcie uruchomienia (analogicznie **watch**); **Clear**, kasująca wszystkie konstruktory z pamięci. Po wykonaniu tej instrukcji konieczne jest ponowne wprowadzenie konstruktorów do pamięci. Do pełnej kontroli pracy programu służą ponadto okienka deklarowane w opcji **Window**. Możliwe jest uaktywnienie np. okienka, odpowiedzialnego za prezentację faktów. Stan zawartości okienek aktualizuje się po wykonaniu akcji zgodnej z listą poleceń systemowych.

Numery faktów są zgodne z kolejnością ich deklaracji. *Fakt zerowy* (initial-fact) pojawia się po wykonaniu polecenia (**reset**) i jest konieczny do uruchomienia każdego systemu. W bazę wiedzy można ingerować, kasując wybrane fakty poleceniem (**retract** ?*adres-faktu*). Również kolejność reguł jest zgodna z deklaracją. Od kolejności umieszczenia reguł na stosie zależy kolejność ich odpalania. Ma to duże znaczenie, gdyż z kolei od kolejności odpalenia może zależeć sposób rozwiązania danego problemu. Strategiom działania systemu w wypadku reguł o tym samym priorytecie poświęcony będzie następny artykuł cyklu.

Reguły systemu mogą być odpalane w pętli. Kontrolę nad pracą systemu można zorganizować za pomocą polecenia **test**. Poniższe dwie reguły realizują system pracujący w pętli kontrolowanej instrukcją test:

```
defrule inicjuj
        (initial-fact)
        =>
        (printout t „Wprowadź liczbę pętli ?max:„)
        (bind ?max (read))                  ;czyta liczbę z klawiatury
        (assert (stan-pętli 1 ?max)))       ;deklaracja faktu
                                            ;określającego stan pętli
(defrule druk-stanu-pętli
        ?loo <-(stan-pętli ?licznik ?max)   ;?loo przyjmuje wartość
                                            ;adresu faktu w pamięci
        (test (<= ?licznik ?max))           ;kryterium odpalenia
                                            ;reguły
        =>
        (printout t „Kolejny numer odpalenia reguły „ ?licznik crlf)
        (retract ?loo)                      ;polecenie skasowania
                                            ;faktu o adresie ?loo
        (assert(stan-pętli=(+?licznik 1)?max))) ;deklaracja faktu
                                            ;z aktualnym stanem
                                            ;pętli
```

Po uruchomieniu systemu procedura druk-stanu-pętli zostanie odpalona *?max* razy. Warto zwrócić uwagę na formę realizacji pętli. Reguła druk-stanu-pętli zostaje odpalona pierwszy raz po wygenerowaniu faktu (stan-pętli 1 *?max*). Ten fakt jest kasowany przez tę regułę. Następnie, po wykonaniu dowolnej akcji, jest generowany nowy fakt z inkrementowaną wartością licznika pętli. Pojawienie się nowego dotychczas nie interpretowanego faktu spowoduje uaktywnienie procesu przeglądania reguł. Proces przeszukiwania spowoduje ponowne odpalenie reguły druk-stanu-pętli jako pierwszej zawierającej w polu LHS odpowiedni wzorzec. Funkcja test jest tu przykładowo użyta do badania kryterium stopu.

Przykłady prostych systemów eksperckich

Na zakończenie przedstawimy kilka przykładów użycia języka CLIPS wraz z opisami.

- Powtórzenie sekwencji akcji

```
(defrule powtórzenie
        (initial-fact)
        =>
        (printout t „Wprowadź daną" ?dana crlf);   przykład
                                                   ;danej
        bind ?dana (read))                         ;nadaje danej
                                                   ;wartość
                                                   ;z klawiatury
        (printout t „Wprowadzona dana wynosi „ ?dana crlf)
        (printout t „Czy chcesz wprowadzić kolejną daną? Y/N")
        (assert (powtórzenie =(read))))            ;generuje
                                                   ;fakt
                                                   ;określający
                                                   ;kontynuację
(defrule kontynuacja
        ?powt <-(powtórzenie Y/y)                  ;test
                                                   ;kontynuacji
                                                   ;z zastosowa-
                                                   ;niem opera-
                                                   ;tora or ?init
<-(initial-fact)
        =>
        (retract ?powt ?init)                      ;zerowanie
                                                   ;stosu
                                                   ;faktów
        (assert (initial-fact)))                   ;ustawienie
                                                   ;warunku
                                                   ;z pola LHS
                                                   ;reguły
                                                   ;powtórzenie
```

W tym wypadku w pętli pracują dwie reguły. Jedna wywołuje drugą do momentu, aż z klawiatury podamy polecenie wyjścia z pętli.

- Przykład kilku reguł, interpretujących światła na skrzyżowaniu;

```
(defrule idź
        (światło zielone)
        => (printout t„Możesz iść" crlf))
(defrule uwaga
        (światło żółte|zielone-migające)
        => (printout t „Uwaga" crlf))
(defrule stój
        (światło ~ green)
        => (printout t „Stój" crlf))
```

- Przykład reguły będącej uzupełnieniem powyższego systemu eksperckiego o możliwość kreowania nowych:

```
(defrule kreator
        (światło ?kolor)
        =>
        (bind ?lista-1 „(defrule nowa-reguła-") (bind ?lista-2 (read))
        (bind ?lista-3 „(światło„)              (bind ?lista-4 (kolor)
        (bind ?lista-5 „) => (printout t'„       (bind ?lista-6(read)) ;rodzaj akcji
        (bind ?lista-7,„'crlf))")
        (bind ?lista(str-cat ?lista-1 ?lista-2 ?lista-3 ?lista-4 ?lista-5 ?lista-6 ?lista-7))
                                    ;skleja listy w jedną
        (build ?lista))             ;buduje nową strukturę według
                                    ;deklaracji ?lista
```

W wyniku zadziałania tej reguły pod wpływem faktu nie zinterpretowanego przez wcześniejsze, zostanie utworzony i dołączony do systemu nowy konstruktor. Po wykonaniu instrukcji **reset** konstruktor ten zostanie dołączony do systemu jako nowa reguła. Przykład ten pokazuje, jak istotna jest kolejność ułożenia reguł na stosie. Umieszczenie reguły *kreator* na początku systemu spowoduje, że żadna reguła z przedstawionych wcześniej nie zostanie odpalona. Dlatego też konieczne jest umieszczenie jej na końcu jako reguły, uzupełniającej dany system. Każda reguła utworzona przez regułę *kreator* zostanie umieszczona na początku systemu. Przykład ten pokazuje możliwości CLIPS-a jako języka umożliwiającego samomodyfikację systemów w czasie rzeczywistym.

LITERATURA

[1] CLIPS Reference Manual. Vol I: Basic Programming Guide, CLIPS Version 6.0, Software Technology Branch, Lyndon B. Johnson Space Center, 1995

[2] Ignizio J.P.: Introduction to expert system. McGraw-Hill Inc. New York, 199[illegible]

Laboratorium komputerowe Akademii Ekonomicznej w Poznaniu

Akademia Ekonomiczna w Poznaniu wzbogaciła się w tym roku akademickim o nowe laboratorium komputerowe, które pozwoli na wprowadzenie do programu zajęć z zakresu zintegrowanych systemów informacyjnych zarządzania. Laboratorium zostało wyposażone w oprogramowanie firmy SAP, umożliwiające projektowanie, wdrażanie i eksploatację w pełni zintegrowanych systemów zarządzania przedsiębiorstwami, bankami, towarzystwami ubezpieczeniowymi, jednostkami samorządu terytorialnego. Oprogramowanie to wykorzystują w zarządzaniu takie firmy, jak IBM, SIEMENS, Deutsche Bank.

Laboratorium umożliwia też wspomaganie modelowania systemów informacyjnych, co jest szczególnie przydatne przy wdrażaniu zintegrowanych systemów lub przy reorganizacji procesów gospodarczych. Można wówczas modelować nowy proces, porównać go z istniejącymi, wybrać korzystniejszy wariant. Narzędzia do modelowania – ARISToolSet – zostały opracowane przez Człowieka Roku Gospodarki Niemiec w 1993 r. – prof. W.A. Scheera.

Prof. dr Witold Abramowicz, kierownik Katedry Informatyki Ekonomicznej w AE w Poznaniu podkreśla, iż nowe laboratorium pozwoli na rozpoczęcie procesu integracji przedmiotów informatycznych z takimi przedmiotami, jak rachunkowość, finanse, zarządzanie, marketing i inne, a tym samym zmodernizuje kształcenie w zakresie tych nauk. (k)

Programowanie w języku systemów eksperckich CLIPS (2)

Strategie i mechanizmy wnioskowania w języku CLIPS

Tomasz Pytlak

Samodzielna Pracownia Sztucznej Inteligencji

Instytut Podstaw Elektroniki

Politechnika Warszawska

Język CLIPS mając wbudowane mechanizmy wnioskowania i samomodyfikacji może służyć do realizacji rozbudowanych systemów eksperckich. W przeciwieństwie do innych języków, specyficzna konstrukcja programu pozwala na łatwą implementację procesów i zagadnień z dziedziny sztucznej inteligencji.

Program napisany w języku CLIPS składa się z niezależnych od siebie konstruktorów. O jakości i poprawności systemu eksperckiego decyduje sposób opracowania konstruktorów, z których dwa: fakty (ang. *deffacts*) i reguły (ang. *defrule*) mają podstawowe znaczenie. Kolejność odpalania reguł może być inna w przypadku wyboru różnych strategii wnioskowania. Jest jednak dokładnie zdefiniowana dla każdej strategii. Listę reguł, które spełniają warunki i nie zostały jeszcze odpalone zawiera *agenda*. Można przyjąć, iż agenda jest czymś podobnym do stosu. Prezentowana jest tam lista reguł zgodnie z wybraną strategią. Reguła znajdująca się na szczycie agendy jest odpalana jako pierwsza. Jeżeli aktywowana jest nowa reguła, jej pozycja jest określona zgodnie z następującymi zaleceniami:

a) nowo aktywowana reguła zajmuje pozycję nad wszystkimi o mniejszym priorytecie (ang. *salience*) i pod regułami o wyższym priorytecie,
b) wśród reguł o tym samym priorytecie pozycję determinuje bieżąca strategia,
c) jeżeli reguła jest aktywowana (równocześnie z innymi regułami) przez to samo twierdzenie lub fakt, natomiast kroki *a* i *b* nie umożliwiają określenia kolejności, wtedy reguła jest arbitralnie (nie losowo!) ustawiana w relacji do innych, aktywowanych reguł.

Ma to szczególny związek z umieszczaniem reguły w implementacji software'owej. Przedstawiony mechanizm działania programu nie jest możliwy do realizacji w tradycyjnych językach, gdzie punkt startu, stopu i sekwencja instrukcji do wykonania są określone przez programistę. W CLIPS-ie dokładny układ programu nie jest potrzebny. Lista faktów i lista reguł są odseparowane i maszyna wnioskująca wbudowana w CLIPS używa wiedzy zawartej w danych. Inny niż w tradycyjnych językach jest też proces działania programu. Wnioskowanie odbywa się na bieżąco, a użytkownik w każdej chwili może poznać poziom zaawansowania problemu. Ze względu na otwartą postać programu[1] warto poznać podstawowe cykle egzekucji reguł.

Podstawowe cykle egzekucji reguł

A. Jeżeli limit odpalania reguł został osiągnięty lub nie ma więcej bieżących faktów inicjujących, egzekucja zostaje zatrzymana. W innym przypadku jest selekcjonowana reguła ze szczytu agendy aktualnie aktywnego modułu[2]. Jeżeli nie ma tam reguł, to wskaźnik na stosie przesuwa się na następny moduł. Jeżeli jest on pusty, egzekucja zostaje wstrzymana, w przeciwnym wypadku krok *a* jest egzekwowany ponownie.
B. Egzekwowane są akcje prawej strony reguły. Użycie funkcji **return** w części RHS może usunąć aktualny wskaźnik stosu. Liczba odpalonych reguł zostaje zwiększona o 1 i porównana z limitem liczby odpalonych reguł.
C. Rezultatem kroku *b* może być aktywacja lub dezaktywacja reguł. Aktywowane reguły, czyli te, w których satysfakcjonujące są ich części LHS, są umieszczone w agendzie modułu, w którym są zdefiniowane. Ich położenie jest determinowane przez priorytet i bieżącą strategię. Dezaktywowane reguły są pomijane i usuwane z agendy. Jest możliwe włączenie bieżącego przeglądania komentarzy określających stan aktywacji reguł.
D. Jeżeli jest używana dynamiczna zmiana priorytetu, to po odpaleniu reguły ustala się ponownie ich listę w agendzie. Wartości priorytetu są modyfikowane. Stan ten powtarza się cyklicznie od kroku *a*.

Strategia wyboru kolejności reguł

System ekspercki jest realizowany w postaci zbioru reguł. Reguły mogą być podzielone na moduły będące zbiorami reguł posegregowanymi tematycznie. Na ogół reguły należące do jednego modułu mają ten sam priorytet. W związku z tym może zdarzyć się konflikt w wyborze kolejności odpalania reguł. W przypadku języków strukturalnych zjawisko to nie występuje, ponieważ instrukcje są wykonywane sekwencyjnie. W tym przypadku o kolejności odpalania reguł decyduje wybrana strategia. Domy-

[1] Do bazy reguł można w dowolnym momencie i miejscu dołączyć nowe reguły

[2] Moduł to grupa tematyczna reguł o tej samej ważności w procesie wnioskowania

ślnie jest ustawiana strategia w głąb (ang. *depth*), ale za pomocą instrukcji *setstrategy* mamy możliwość wyboru jednej z siedmiu innych: **depth** (w głąb), **breadth** (wszerz), **simplicity** (naturalna), **complexity** (złożona), **lex, mea,** random (losowa).

Depth strategy

Nowo aktywowana reguła zajmuje miejsce ponad wszystkimi regułami o tym samym priorytecie:
fact-a aktywuje rule-1 rule-2
fact-b aktywuje rule-3 rule-4
Jeżeli fakt fact-a jest wykazywany przed faktem fact-b, to kolejność reguł w agendzie będzie następująca: rule-3, rule-4, rule-1, rule-2.
Kolejność rule-1, rule-2 i rule-3, rule-4 jest określona arbitralnie przez programistę.

Breadth strategy

Nowo aktywowana reguła zajmuje miejsce pod wszystkimi regułami o tym samym priorytecie:
fact-a aktywuje rule-1 rule-2
fact-b aktywuje rule-3 rule-4
Jeżeli fakt fact-a jest wykazywany przed faktem fact-b, to kolejność reguł w agendzie będzie następująca: rule-1, rule-2, rule-3, rule-4.
Kolejność rule-1, rule-2 i rule-3, rule-4 jest określona arbitralnie przez programistę.

Simplicity strategy

Wśród reguł o tym samym priorytecie nowo aktywowana reguła zajmuje miejsce ponad wszystkimi aktywowanymi regułami o tej samej lub wyższej specyficzności. Specyficzność jest determinowana przez liczbę warunków i jest określona na podstawie LHS. Każde porównanie do stałej lub wcześniej obliczonej zmiennej dodaje 1 do specyficzności. Każda wykonana funkcja w LHS, jak np.:, =, test *CE* dodaje 1 do specyficzności. Funkcje logiczne **and, or, not** nie zmieniają specyficzności. Funkcje wykonywane wewnątrz wywoływanych funkcji też nie zmieniają specyficzności.

```
(defrule przykład
    (zmienna ?x ?y ?x)
    (test (and (numberp ?x) (> ?x (+ 10 ?y)) (< ?x 100)))
```

ma specyficzność 5: 1) – porównanie do literału (*zmienna*), 2) – porównanie ?x na drugiej i czwartej pozycji, 3) – wywołanie funkcji (**numberp**), 4) – >, 5) – <; + nie zmienia specyficzności.

Complexity strategy

Wśród reguł o tym samym priorytecie nowo aktywowana reguła zajmuje miejsce ponad wszystkimi aktywowanymi regułami o równej lub mniejszej specyficzności.

LEX strategy

Strategia LEX obowiązuje w przypadku, gdy reguły mają kilka wzorców faktów i warunków w polu LHS. Po każdym odpaleniu reguły są ustalane relacje pomiędzy faktami w systemie. Fakty i warunki wchodzące w skład pola LHS reguł są sortowane w porządku malejącym. System kieruje się następującymi zasadami:

a) pierwszeństwo mają reguły zawierające w polu LHS najpóźniej wykazane fakty,
b) w przypadku jednakowej listy faktów w polu LHS pierwszeństwo odpalenia przysługuje regule mającej dodatkowe warunki,
c) jeżeli jedna reguła zawiera w polu LHS listę faktów będącą podzbiorem listy faktów innej reguły, pierwszeństwo ma reguła o dłuższej liście faktów.

Aby dokładnie prześledzć kolejność odpalania wyobraźmy sobie system zawierający sześć reguł (fakty są wykazywane w kolejności wzrastania indeksu):

rule-1 aktywowana przez fakty f-1, f-2, f-3
rule-2 aktywowana przez fakty f-3, f-1
rule-3 aktywowana przez fakty f-2, f-1
rule-4 aktywowana przez fakty f-1, f-2,
rule-5 aktywowana przez fakt f-1, f-2, f-3,
rule-6 aktywowana przez fakty f-1, f-4

Przecinek na końcu pola LHS oznacza istnienie warunku (np. instrukcja **test**). W tym momencie następuje proces sortowania faktów. Agenda zgodnie ze strategią LEX będzie więc wyglądać następująco:

rule-6 aktywowana przez fakty f-4, f-1
rule-5 aktywowana przez fakty f-3, f-2, f-1,
rule-1 aktywowana przez fakty f-3, f-2, f-1
rule-2 aktywowana przez fakty f-3, f-1
rule-4 aktywowana przez fakty f-2, f-1,
rule-3 aktywowana przez fakty f-2, f-1

O pierwszeństwie reguły rule-6 decyduje fakt f-4. Reguła rule-5 będzie odpalona przed regułą rule-1, ponieważ ma warunek w polu LHS. Reguła rule-1 będzie odpalona przed regułą rule-2 ze względu na fakt f-2.

MEA strategy

Ta strategia jest bardzo podobna w działaniu do strategii LEX. Zasadniczą różnicą jest brak sortowania faktów przed ustaleniem kolejności odpalania reguł. W celu prześledzenia procesu ustalania kolejności odpalania przeanalizujmy reguły analizowanego wyżej systemu:

rule-2 aktywowana przez fakty f-3, f-1
rule-3 aktywowana przez fakty f-2, f-1,
rule-6 aktywowana przez fakty f-1, f-4
rule-5 aktywowana przez fakty f-1, f-2, f-3,
rule-1 aktywowana przez fakty f-1, f-2, f-3
rule-4 aktywowana przez fakty f-1, f-2

O pierwszeństwie reguły rule-2 decyduje fakt f-3 na pierwszej pozycji pola LHS, to samo decyduje o pozycji reguły rule-3 w agendzie. Fakt f-4 ustawia regułę rule-6 na pozycji trzeciej. Reguła rule-5 będzie odpalona przed regułą rule-1 ze względu na warunek w polu LHS. Reguła rule-4 jest odpalona za regułą rule-4 ze względu na krótszą listę faktów.

Random strategy

Każda aktywacja jest wyznaczana z losowo określoną kolejnością, dzięki której są ustalane reguły.

Postać pola LHS reguł

Warunki są umieszczane w polu LHS (ang. *left side hand*) każdej reguły. W języku CLIPS zdefiniowano osiem typów elementów warunkowych: **pattern, test, and, or, not, exists** (forall, logical). Poniżej zostaną przedstawione proste przykłady użycia większości z nich w praktycznych implementacjach.

Odpalenie reguły zgodnie z podanym wzorcem faktu (pattern)

Poniżej przedstawiono przykład programu realizującego wprowadzanie znaków z klawiatury i ich prezentację na monitorze.

```
(defrule repetycja
        (initial-fact)
;inicjacja reguły po wykonaniu instrukcji reset
        =>
        (printout t "Wpisz znak z klawiatury : ")
        (bind ?at (read))
;do zmiennej ?at podstawiony został znak z klawiatury
        (printout t " Wpisano następujący znak : " ?at crlf)
        (printout t " Czy chcesz wpisać następny? (t/n) ")
        (assert (resp =(read)) ))

(defrule następny-znak
        ?res <- (resp t)
;zmiennej ?res przypisano adres faktu (resp t)
        ?init <- (initial-fact)
        =>
        (retract ?init)
        (assert (initial-fact))
        (retract ?res))
;fakt o adresie ?res został skasowany
```

Druga reguła zostanie odpalona tylko wtedy, gdy zostanie wygenerowany fakt (resp t). W wyniku jej działania zostanie wygenerowany nowy fakt o wzorcu zgodnym z polem warunkowym pierwszej reguły.

Test w polu warunkowym reguły

W polu warunkowym reguły można umieścić instrukcję **test.** Reguła zostanie odpalona, gdy wartość logiczna tej instrukcji będzie mieć wartość non-FALSE.

```
(defrule wielkość-pętli
        (initial-fact)
        =>
        (printout t " Podaj liczbę określającą wielkość pętli " )
        (bind ?max (read))
        (assert (pętla 0 ?max)))
;generacja faktu o ogólnym wzorcu (pętla ?nr-kolejny ?max)

 (defrule druk-numeru-pętli-i-kwadratu-liczby
        (loop ?count ?max)  ;wzorzec faktu
        (test (<= ?count ?max))
;test działania sprawdzającego osiągnięcie znacznika końca pętli
        =>
        (bind ?kwadrat (* ?count ?count))
;obliczenie kwadratu liczby
        (printout t " kwadrat liczby " ?count
                " wynosi : " ?kwadrat crlf)
        (assert (loop = (+ ?count 1) ?max)))
;generacja faktu o zwiększonej o 1 wartości ?count.
```

Przykład pokazuje prostą realizację pętli za pomocą reguły, która sama dla siebie generuje fakt inicjujący. Wzorzec faktu zawiera parametry pracy pętli.

Funkcje logiczne w polu warunkowym reguł

Poniższe przykłady pokazują sposób użycia funkcji logicznych **and, or, not** w polu warunkowym reguł.

```
(defrule pożar
        (or (temperatura wysoka)(zadymienie duże)(alarm działa))
        =>
        (printout t "Niebezpieczeństwo pożaru,
                włącz system przeciwpożarowy " crlf)
        (assert (diagnoza pożar)))
```

Reguła *pożar* jest równoważna trzem regułom, z których każda zawiera tylko jeden z użytych w instrukcji or wzorców.

```
(defrule uszkodzenie-sprzętu
        (temperatura w-normie)
        (alarm nie-działa)
        (not (zadymienie w-normie))
        =>
        (printout t "Sprawdź, czy nie ma zwarcia elektrycznego "))
```

Reguła *uszkodzenie-sprzętu* sprawdza stan monitorowanego obiektu wykluczając pożar i sugeruje zlokalizowanie awarii.

```
(defrule klimatyzacja
(or     (and (temperatura wysoka)(alarm nie-działa))
        (and (temperatura niska)(alarm nie-działa)) )
        =>
        (printout t "Włącz klimatyzację"))
```

Reguła *klimatyzacja* nie jest odpalana w przypadku zagrożenia. Eliminując stany alarmowe podaje komunikat o przekroczeniu pewnego przedziału temperatury wewnątrz monitorowanego obiektu.

Test na istnienie faktu (exist)

Funkcja **exist** sprawdza, czy grupa wyspecyfikowanych faktów w polu LHS występuje co najmniej raz. Poniżej jest podany przykład reguł zawierających w polu warunkowym ekwiwalentną postać warunku:

```
(defrule przykład-exist
        (exist (a ?x) (b ?x))
        =>)
(defrule przykład-ekwiwalentny
        (not (not (and ((a ?x) (b ?x))))
        =>)
```

Na zakończenie zostaną przedstawione przykłady programów realizujących pętle wewnątrz reguły oraz sposób korzystania z plików zewnętrznych.
Do realizacji pętli wewnątrz reguły służy instrukcja **while:**

```
(defrule kwadraty-liczb
        (initial-fact)
        =>
        (printout t " Podaj wielkość pętli ? ")
        (bind ?max (read))
        (bind ?count 0)
;inicjacja pętli od wartości ?count równej 0
        (while (<= ?count ?max)
                (bind ?kwadrat (* ?count ?count))
                (printout t " Kwadrat liczby " ?count
                        " wynosi " ?kwadrat crlf)
                (bind ?count (+ ?count 1)) ))
```

Poniższy przykład prezentuje sposób wykorzystania danych zapamiętanych w pliku wewnętrznym. Sekwencja instrukcji służąca do współpracy z zewnętrznymi plikami jest podobna do instrukcji z innych języków. Za każdym razem należy pamiętać o otworzeniu pliku, a po wykonaniu operacji o jego zamknięciu.

```
(defrule praca-z-plikiem-zewnętrznym
        (initial-fact)
        =>
        (printout t "Wprowadź nawę pliku w cudzysłowiu : ")
        (bind ?name (read))
        (printout t "Wprowadź identyfikator pliku: ")
        (bind ?idname (read))
        (call (open ?name ?idname "w" ))
;plik przygotowany do zapisu danych
        (printout t "Otworzono plik" ?name crlf
                "Użyj EOF jako znacznika końca" )
        (bind ?input (read))
        (while (!(eq ?input EOF))
;! jest równoważny funkcji not
                (fprintout t ?idname ?input crlf)
                (bind ?input (read)))
        (call (close))
;zamknięto plik o identyfikatorze ?idname
        (printout t " Plik " ?name " został zamknięty" crlf))
```

W instrukcji open można użyć innych parametrów pozwalających na odczyt danych (domyślnie) r, zapis danych w, zapis i odczyt r+, uzupełnienie na końcu zbioru a.

*

Informacje zawarte w dotychczas prezentowanych opisach poświęconych językowi CLIPS wystarczają do opracowania systemów eksperckich. W dalszych opracowaniach będą one systematycznie uzupełniane o bardziej zaawansowane techniki programowania. Równocześnie zostaną wprowadzone elementy programowania obiektowego, co pokaże niespotykane możliwości języka CLIPS związane z połączeniem inteligentnego przetwarzania informacji zawartej w bazach wiedzy z wnioskowaniem i tworzeniem nowej wiedzy.

WordPerfect dla DOS-u

Novell zakończył prace nad procesorem tekstów WordPerfect 6.1 dla DOS, który został ulepszony pod kątem zwiększenia prędkości działania i zmniejszenia zapotrzebowania na zasoby systemu. Wprowadzono też wiele funkcji ułatwiających posługiwanie się procesorem. Można na przykład edytować tabele w oknie dokumentu bez konieczności przełączania się do osobnego trybu edycji tabel. Dostępne w oknach dialogowych nowe listy zawierają nie tylko nazwy wybranych przez użytkownika opcji, co przyspiesza korzystanie z ostatnio używanych plików, opcji wyszukiwania, stosowania czcionek czy też drukowania adresów na kopertach. WordPerfect 6.1 wyposażono także w graficzny interfejs z systemem rozwijanych menu, umożliwiających wybór pracy w trybie WYSIWYG. Ponadto nowa wersja umożliwia dokonywanie konwersji plików arkusza kalkulacyjnego Quattro Pro 5.5 dla DOS. WordPerfect 6.1 dla DOS jest zgodny z WordPerfect dla UNIX i dla Macintosha. Ocenia się, że z różnych wersji WordPerfecta dla DOS korzysta na całym świecie ponad 10 mln użytkowników. (k)

Atuty kart elektronicznych

dokończenie ze s. 5

sprzętem oraz oprogramowaniem, które nie przystaje do nowego rozwiązania? Problem kolejny jest również związany z rachunkiem ekonomicznym wdrożenia elektronicznych kart płatniczych. Porównując koszt zakupu kart magnetycznych z kartami procesorowymi, okazuje się, że te ostatnie są co najmniej sześciokrotnie droższe. Z drugiej strony długofalowy rachunek wskazuje na znacznie niższe koszty eksploatacji systemu kart płatniczych opartych na technologii kart elektronicznych. Pojawiają się również wątpliwości co do bezpieczeństwa obrotu elektronicznym pieniądzem zapisanym w karcie. Problem ten wydaje się być najmniej jednak znaczący, ponieważ dotychczasowe doświadczenia dowodzą znacznie wyższego poziomu bezpieczeństwa zapisu w karcie elektronicznej niż na karcie magnetycznej. Wątpliwości związane z powyższymi problemami były na tyle silne, że jeszcze w roku 1988 międzynarodowi potentaci w zakresie systemów kartowych twierdzili zdecydowanie, że *dobrowolnie nie zainteresują się technologią kart elektrotechnicznych.*

Przełom nastąpił w grudniu 1993 roku, kiedy to dwa brytyjskie banki NATWEST oraz MIDLAND ogłosiły oficjalnie rok 1995 jako datę wprowadzenia systemu MONDEX, międzynarodowego produktu bankowego opartego na technologii kart elektronicznych. Reakcja nawet najbardziej zagorzałych przeciwników tej technologii była natychmiastowa. Konkurencyjne systemy VISA i MASTER CARD stworzyły wspólną komisję, której celem było opracowanie specyfikacji (opublikowanej w końcu 1994 r.) sprzętu, kart, procedur oraz protokołów wymiany informacji pomiędzy kartami elektronicznymi pracującymi w tych systemach. MASTER CARD, jako zawsze pierwszy wprowadzający innowacje w międzynarodowych systemach kartowych ogłosił dziesięcioletni, bardzo napięty plan wdrożenia kart elektronicznych. Zakładał on np. wprowadzenie układu elektronicznego na wszystkie karty – produkty systemu MASTER CARD – do końca roku 1996 oraz wprowadzenie możliwości obsługi kart nowej technologii przez wszystkie urządzenia (EETPOS, ATM itd.) do końca roku 2002.

O ile wszystkie wyżej wymienione działania dotyczą zastosowania układu elektronicznego jako dodatkowego zabezpieczenia do funkcjonującego dalej paska magnetycznego, to przedstawiciele MASTER CARD już głośno twierdzą, że *MC analizuje i nie wyklucza żadnego z innych potencjalnych produktów opartych na tej technologii,* zauważając jednocześnie, że oprócz znacznego wzrostu bezpieczeństwa systemów karta elektroniczna wprowadzi „nowe możliwości zysków". MASTER CARD mówi o nowych produktach, nowych usługach oraz nowych rynkach dla produktów opartych na kartach elektronicznych. Gdy się analizuje inne dostępne źródła, uwagę zwracają zaawansowane prace nad wprowadzeniem elektronicznej karty debetowej o bardzo powszechnym zasięgu przed rokiem 2000 (np. system INTERAC). W tej sytuacji powraca zadawane w Polsce od kilkunastu miesięcy pytanie: *czy stać nas teraz na inwestowanie w przestarzałą technologię kart magnetycznych, kiedy wiadomo już, że za kilka lat będziemy musieli wejść w systemy kart elektronicznych?*
Odpowiedź zobaczymy już w tym roku w swoich portfelach.