Predikatske funkcije

- Predikatske funkcije su sve funkcije koje vraćaju simbol TRUE ili simbol FALSE. Mogu biti predefinisane ili definisane od strane korisnika. Sve predefinisane predikatske funkcije su date u delu 12.1 Basic programming guide-a.

PRIMERI

- **(= <brojni-izraz> <brojni-izraz> +)** vraća TRUE ako njen prvi argument ima vrednost koja je jednaka vrednostima svih ostalih argumenata, a u suprotnom vraća FALSE.

```
CLIPS> (= 3 3.0)
TRUE
```

- **(<> <brojni-izraz> <brojni-izraz> +)** vraća TRUE ako njen prvi argument ima vrednost koja je različita od vrednosti svih ostalih argumenata, a u suprotnom vraća FALSE.

- **(> <brojni-izraz> <brojni-izraz> +)** vraća TRUE ako za sve njene argumente važi da (n-1)-vi argument ima vrednost koja je veća od vrednosti n-tog argumenata, a u suprotnom vraća FALSE.

```
CLIPS> (> 5 4 3)
TRUE
CLIPS> (> 5 3 4)
FALSE
```

- **(< <brojni-izraz> <brojni-izraz> +)** vraća TRUE ako za sve njene argumente važi da (n-1)-vi argument ima vrednost koja je manja od vrednosti n-tog argumenata, u suprotnom vraća FALSE.

- **(and <izraz> +)** vraća TRUE ako svi njeni argumenti imaju vrednost TRUE, a u suprotnom vraća FALSE.

```
CLIPS> (and (> 4 3)(> 4 5))
FALSE
```

- **(or <izraz> +)** vraća TRUE bar jedan njen argument ima vrednost TRUE, u suprotnom vraća FALSE.

```
CLIPS> (or (> 4 3)(> 4 5))
TRUE
```

PRIMER: Igra STICKS

- STICKS je igra za dva igrača.
- Cilj je da igrač izbegne da uzme poslednji štapić sa gomile.
- U svakom krugu igrač uzima 1, 2 ili 3 štapića sa gomile.
- Trik (heuristika) za pobedu je da drugom igraču ostavimo 5 štapića na gomili, odnosno da posle svakog našeg poteza na gomili ima 5 plus neki umnožak od 4.

PRIMER: Igra STICKS

- Program igra protiv čoveka.
- Mora se odrediti ko prvi igra i koliko je štapića na gomili na početku. Ove informacije unosi korisnik.

PRIMER: Igra STICKS

```
(deffacts initial-phase
   (phase choose-player))


(defrule player-select
   (phase choose-player)
   =>
   (printout t "Who moves first
  (Computer: c, Human: h)? ")
   (assert (player-select (read))))
```

PRIMER: Igra STICKS

```
; RULE good-player-choice
; IF
;   The phase is to choose the first player, and
;   The human has given a valid response
; THEN
;   Remove unneeded information, and
;   Indicate whose turn it is, and
;   Indicate that the pile size should be chosen

(defrule good-player-choice
   ?phase <- (phase choose-player)
   ?choice <- (player-select ?player&c | h)
   =>
   (retract ?phase ?choice)
   (assert (player-move ?player))
   (assert (phase select-pile-size)))
```

PRIMER: Igra STICKS

```
; RULE bad-player-choice
; IF
;   The phase is to choose the first player, and
;   The human has given a invalid response
; THEN
;   Remove unneeded information, and
;   Indicate that the first player should be chosen again,
;   and Print the valid choices
```

PRIMER: Igra STICKS

```
(defrule bad-player-choice
   ?phase <- (phase choose-player)
   ?choice <- (player-select
  ?player&~c&~h)
   =>
   (retract ?phase ?choice)
   (assert (phase choose-player))
   (printout t "Choose c or h." crlf))
```

Uslovni element **`test`**

- Koristi se za procenjivanje izraza na levoj strani pravila.

`(test <poziv-funkcije>)`

- **`Test`** je zadovoljen ako funkcija koja se u okviru njega poziva vrati vrednost različitu od FALSE.
- U okviru **`test`** uslovnog elementa može se smestiti bilo koja spoljna funkcija.

PRIMER:

```
CLIPS>(deftemplate stranice-trougla
           (slot a)(slot b)(slot c))

CLIPS>(defrule unos-stranica-trougla
(stranice-trougla (a ?a)(b ?b)(c ?c))
(test (< ?c (+ ?a ?b)))
 =>
(printout t "Stranice su pravilno unete" crlf))

CLIPS> (assert (stranice-trougla (a 3)(b 4)(c 5)))
==> f-0     (stranice-trougla (a 3) (b 4) (c 5))
==> Activation 0      unos-stranica-trougla: f-0
<Fact-0>
CLIPS> (run)
FIRE    1 unos-stranica-trougla: f-0
Stranice su pravilno unete
CLIPS>
```

test CE

- Kako se simbol test koristi da bi se naglasio ovaj tip uslovnog elementa, u pravilima se simbol test ne sme koristiti kao prvo polje u patern uslovnom elementu.

```
CLIPS> (assert (test 1))

[PATTERN1] The symbol test has special meaning
and may not be used as a relation name.

CLIPS> (defrule t (test 1) =>)

[PRNTUTIL2] Syntax Error:  Check appropriate syntax for
function calls.
```

PRIMER: Igra STICKS

```
; RULE get-human-move
; IF
;    It is the human's move, and
;    The pile size is greater than 1
; THEN
;    Ask how many sticks to take, and
;    Get the human's response

(defrule get-human-move
   (player-move h)
   (pile-size ?size)
   (test (> ?size 1))
=>
   (printout t "How many sticks do you wish to
  take? ")
   (assert (human-takes =(read))))
```

PRIMER: Igra STICKS

```
; RULE good-human-move
; IF
;   There is a pile of sticks, and
;   The human has chosen how many sticks to take, and
;   It is the human's move, and
;   The human's choice is valid
; THEN
;   Remove unneeded information, and
;   Compute the new pile size, and
;   Update the pile size, and
;   Print the number of sticks left in the stack, and
;   Trigger the computer player's turn
```

PRIMER: Igra STICKS

```
(defrule good-human-move
   ?pile <- (pile-size ?size)
   ?move <- (human-takes ?choice)
   ?whose-turn <- (player-move h)
   (test (and (integerp ?choice)
              (>= ?choice 1)
              (<= ?choice 3)
              (< ?choice ?size)))
   =>
   (retract ?pile ?move ?whose-turn)
   (bind ?new-size (- ?size ?choice))
   (assert (pile-size ?new-size))
   (printout t ?new-size " stick(s) left in the
  pile."
               crlf)
   (assert (player-move c)))
```

PRIMER: Igra STICKS

```
; RULE bad-human-move
; IF
;   There is a pile of sticks, and
;   The human has chosen how many sticks to take, and
;   It is the human's move, and
;   The human's choice is invalid
; THEN
;   Print the valid choices, and
;   Remove unneeded information, and
;   Retrigger the human player's move
```

PRIMER: Igra STICKS

```
(defrule bad-human-move
   (pile-size ?size)
   ?move <- (human-takes ?choice)
   ?whose-turn <- (player-move h)
   (test (or (not (integerp ?choice))
             (< ?choice 1)
             (> ?choice 3)
             (>= ?choice ?size)))
   =>
   (printout t "Number of sticks must be between
               1 and 3,“ crlf
               "and you must be forced to take
               the last stick." crlf)
   (retract ?move ?whose-turn)
   (assert (player-move h)))
```

Predicate constraint :

- Upotreba predikatskih funkcija za nametanje ograničenja na vrednost polja naziva se *predicate field constraint*.

- Predicate field constraint se može kombinovati sa *field constraints*-ima ~, &, |

Predicate constraint :

U pravilu get-human-move korišćena su dva uslova da bi se proverilo da li ima više od jednog štapića na gomili:

```
(pile-size ?size)
(test (> ?size 1))
```

A dovoljan je jedan:

```
(pile-size ?size&:(> ?size 1))
```

Predicate constraint se može čitati **tako da je**

Predicate constraint se može primeniti za proveravanje tipova podataka

```
(defrule provera-za-string
(podatak ?a& : (stringp ?a))
=>
(printout t "Podatak je tipa
  string" crlf))
```

Return Value Constraints

- Moguće je koristiti vrednost koju vraća neka spoljna funkcija kako bi se nametnula ograničenja na vrednosti koje polje može da poprimi.

- *return value constraint (=)* omogućava korisniku da unutar uslova na LHS poziva spoljne funkcije.

PRIMER:

Pravilo se aktivira za sve činjenice kod kojih je vrednost slota y dva puta veća od vrednosti slota x.

```
CLIPS> (deftemplate podaci (slot x) (slot y))
CLIPS> (defrule duplo
                (podaci (x ?x) (y =(* 2 ?x)))
                 =>
       )
CLIPS> (assert (podaci (x 2) (y 4))
               (podaci (x 3) (y 5))
               (podaci (x 1) (y 2)))
==> f-0     (podaci (x 2) (y 4))
==> Activation 0      duplo: f-0
==> f-1     (podaci (x 3) (y 5))
==> f-2     (podaci (x 1) (y 2))
==> Activation 0      duplo: f-2
```

PRIMER: Igra STICKS

```
(deftemplate take-sticks
   (slot how-many)
; Number of sticks to take.
   (slot for-remainder))
; Remainder when stack is  divided by 4.

(deffacts take-sticks-information
   (take-sticks (how-many 1) (for-remainder 1))
   (take-sticks (how-many 1) (for-remainder 2))
   (take-sticks (how-many 2) (for-remainder 3))
   (take-sticks (how-many 3) (for-remainder 0)))
```

PRIMER: Igra STICKS

```
; RULE computer-move
; IF
;   It is the computers's move, and
;   The pile size is greater than 1, and
;   The computer's response is available
; THEN
;   Remove unneeded information, and
;   Compute the new pile size, and
;   Print the number of sticks left in the stack,
  and
;   Update the pile size, and
;   Trigger the human players move
```

PRIMER: Igra STICKS

```
(defrule computer-move
   ?whose-turn <- (player-move c)
   ?pile <- (pile-size ?size&:(> ?size 1))
   (take-sticks (how-many ?number)
                (for-remainder =(mod ?size 4)))
   =>
   (retract ?whose-turn ?pile)
   (bind ?new-size (- ?size ?number))
   (printout t "Computer takes " ?number "
  stick(s).“ crlf)
   (printout t ?new-size " stick(s) left in the
  pile.“ crlf)
   (assert (pile-size ?new-size))
   (assert (player-move h)))
```