Lab1

```
(printout t "Hello world" crlf)
(bind ?x (read))
(bind ?y (read))
(if (< ?x ?y) then (printout t ?x crlf)
    else (printout t ?y crlf))

(deftemplate Persoana (slot Nume)(slot Prenume)(multislot mama)(multislot
tata)(multislot casatorit))
(assert (Persoana(Nume Pop)(Prenume Rafael)(mama Pop Vioica)(tata pop Viorel)(casatorit
nil)))
(facts)

(deffacts Familie
    (Persoana(Nume Pop)(Prenume Rafaela)(mama Pop Ana)(tata Pop Georgel)(casatorit Pop
Ionut))
    (Persoana(Nume Popescu)(Prenume Rafael)(mama Popescu Vioica)(tata Popescu
Viorel)(casatorit nil)))
(reset)
(facts)

(defrule nume_regula
    (Persoana(Nume ?x)(Prenume ?y)(mama $?m)(tata $?t)(casatorit $? Pop Ionut $?){Nume
== Pop})
    =>
    (printout t ?x ", " ?y ", " $?m ", " $?t crlf))
(run)
```

Familie

```
;lab1-ex1-a)
 (deftemplate Persoana (multislot Nume)(multislot mama)(multislot tata)(multislot
casatorit))
(deffacts Familie
    (Persoana(Nume Ion Popescu)(mama nil)(tata nil)(casatorit Maria Popescu))
      (Persoana(Nume Maria Popescu)(mama nil)(tata nil)(casatorit Ion Popescu))

      (Persoana(Nume Cristian Popescu)(mama Maria Popescu)(tata Ion Popescu)(casatorit
      Elena Popescu))
      (Persoana(Nume Elena Popescu)(mama nil)(tata nil)(casatorit Cristian Popescu))

      (Persoana(Nume Adrian Popescu)(mama Maria Popescu)(tata Ion Popescu)(casatorit
      Cristina Popescu))
      (Persoana(Nume Cristina Popescu)(mama nil)(tata nil)(casatorit Adrian Popescu))
            (Persoana(Nume Tudor Popescu)(mama Cristina Popescu)(tata Adrian
            Popescu)(casatorit nil))

      (Persoana(Nume Ciprian Popescu)(mama Maria Popescu)(tata Ion Popescu)(casatorit
      Adriana Popescu))
      (Persoana(Nume Adriana Popescu)(mama nil)(tata nil)(casatorit Ciprian Popescu))

      (Persoana(Nume Maria Andreescu)(mama Maria Popescu)(tata Ion Popescu)(casatorit
      Petre Andreescu))
      (Persoana(Nume Petre Andreescu)(mama nil)(tata nil)(casatorit Maria Andreescu))
            (Persoana(Nume Ionel Andreescu)(mama Maria Andreescu)(tata Petre
            Andreescu)(casatorit nil)))
(reset)
(facts)

(defrule grandp
    (Persoana(Nume $?x)(mama $?y)(tata $?z))
    (Persoana(Nume $?y)(mama $?y1)(tata $?y2))
    (Persoana(Nume $?z)(mama $?z1)(tata $?z2))
    =>
   (printout t "Nepot: " $?x "// Bunici: " $?y1 " & " $?y2 " - " $?z1 " & " $?z2 crlf))
(facts)
(run)
```

Lab2

Arca lui Noe

```
;lab1-ex1-b)
(deftemplate arca(slot animal)(slot gen)(slot inaltime))

(foreach ?animal (create$ soarece veverita pisica caine vaca elefant)
    (foreach ?gen (create$ masculin feminin)
        (printout t "Inaltimea animalului " ?animal " de genul " ?gen " este: ")
        (bind ?inaltime (read))
        (assert (arca(animal ?animal)(gen ?gen)(inaltime ?inaltime)))
    ))
(facts)
```

Suma elementelor

```
;lab1-ex3
(deftemplate element (slot valoare)(slot nume))
      (assert (element (valoare 2)(nume A)))
      (assert (element (valoare 3)(nume B)))
      (assert (element (valoare 7)(nume C)))
      (assert (element (valoare 1001)(nume D)))

(deftemplate suma (slot valoare))
      (assert (suma (valoare 0)))

(defrule CalculSuma
      ?idve <- (element (valoare ?ve))
    ?idvs <- (suma (valoare ?vs))
    =>
    (modify ?idvs (valoare (+ ?ve ?vs)))
    (retract ?idve))

(defrule AfisareSuma
      (suma (valoare ?vs))
    =>
    (printout t "Suma= " ?vs crlf))
;(watch all)
(run)
(facts)
```

Gradina Zoologica

```
;lab1-ex4
(defglobal ?*TASK_PRIORITY_1* = 50)
(defglobal ?*TASK_PRIORITY_2* = 250)

(deftemplate animal
      (slot denumire)
    (slot mancare) ;ierbivor/carnivor
    (slot modViata) ;diurn/nocturn
    (slot mediuViata) ;acvatic/semiacvatic/terestru/aerian
    (slot modReproducere) ;pui/oua
    (slot zona)
    (slot medieViata)
    (slot putere) ;0-ierbivore/1/2
)

(assert (animal(denumire leu)(mancare carnivor)(putere 2)))
(assert (animal(denumire hiena)(mancare carnivor)(putere 1)))
(assert (animal(denumire caprioara)(mancare ierbivor)))
(assert (animal(denumire broasca)(mancare ierbivor)(modViata diurn)(mediuViata
semiacvatic)(modReproducere oua)))
(assert (animal(denumire ariciMare)(mancare ierbivor)(modViata diurn)(mediuViata
semiacvatic)(zona mediteraneana)))
(assert (animal(denumire foca)(zona polara)))
```

```
(assert (animal(denumire ursPolar)(zona polara)))

(defrule antipattern
    ?p <- (animal {zona == polara}(denumire ?name))
    =>
    (printout t "*** Animale polare: " ?name crlf))

(defrule MediuSemiacvatic
    (declare (salience ?*TASK_PRIORITY_1*))
    ?p <- (animal {mediuViata == semiacvatic}(denumire ?name))
    =>
    (printout t "*** Animale care au mediul de viata semiacvatic: " ?name crlf))

(defrule IerbivorMediteranean
    (declare (salience ?*TASK_PRIORITY_2*))
    ?p <- (animal {mancare == ierbivor && zona == mediteraneana}(denumire ?name))
    =>
    (printout t "*** Animalele ierbivore din zona mediteraneana: " ?name crlf))

(defrule pradator-prada
    ?animal1 <- (animal)
    ?animal2 <- (animal {mancare == animal1.mancare && mancare == carnivor && (putere < 
animal1.putere)})
    ?animal3 <- (animal {mancare == carnivor})
    ?animal4 <- (animal {mancare == ierbivor})
    =>
    (printout t "*** Pradatorul este " ?animal1.denumire " si prada este " 
?animal2.denumire crlf)
    (printout t "*** Pradatorul este " ?animal3.denumire " si prada este " 
?animal4.denumire crlf))
(run)
```

Consiliere alegere facultate

```
;lab2-ex4
(deftemplate note
    (slot matematica(type INTEGER))
    (slot informatica(type INTEGER))
    (slot fizica(type INTEGER))
    (slot chimie(type INTEGER))
    (slot limbaRomana(type INTEGER))
    (slot limbaEngleza(type INTEGER))
    (slot limbaGermana(type INTEGER))
    (slot economie(type INTEGER))
    (slot psihologie(type INTEGER))

(assert (note (matematica 0)(informatica 0)(fizica 0)(chimie 0)(limbaRomana 
0)(limbaEngleza 0)(limbaGermana 0)(economie 0)(psihologie 0)))

(defrule citeste
    ?id <- (note (matematica 0)(informatica 0)(fizica 0)(chimie 0)(limbaRomana 
0)(limbaEngleza 0)(limbaGermana 0)(economie 0)(psihologie 0))
    =>
    (printout t "Introduceti notele elevului:" crlf)
    (printout t "-matematica: ") (bind ?m (read))
    (printout t "-informatica: ") (bind ?i (read))
    (printout t "-fizica: ") (bind ?f (read))
    (printout t "-chimie: ") (bind ?c (read))
    (printout t "-limbaRomana: ") (bind ?lr (read))
    (printout t "-limbaEngleza: ") (bind ?le (read))
    (printout t "-limbaGermana: ") (bind ?lg (read))
    (printout t "-economie: ") (bind ?e (read))
    (printout t "-psihologie: ") (bind ?p (read))
    (retract ?id)
    (assert (note (matematica ?m)(informatica ?i)(fizica ?f)(chimie ?c)(limbaRomana 
?lr)(limbaEngleza ?le)(limbaGermana ?lg)(economie ?e)(psihologie ?p))))

(defrule facultateaMateInfo
    (note (matematica ?m & :(>= ?m 8))(informatica ?i & :(>= ?i 8))(fizica ?f & :(>= ?f
8)))
    =>
    (printout t "Studentul poate sa isi depuna dosarul la Facultatea de Matematica si
Informatica" crlf))
(run)
```

Previziuni meteo

```
(deftemplate Meteo (slot perioada)(slot tipNor)(slot vreme)(slot vant)(slot durata))

(assert (Meteo (perioada seara)(tipNor cerRosu)(vreme senin)))
(assert (Meteo (perioada dimineata)(tipNor ceata)(vreme cald)))
(assert (Meteo (perioada miezNoapte)(tipNor cumulonimbus)(vreme zapada)))
(assert (Meteo (tipNor cumulus)(vreme buna)))
(assert (Meteo (tipNor cirrocumulus)(vreme ploaie)))
(assert (Meteo (tipNor nimbostratus)(vreme ploaieLunga)(vant NE-S)))
(assert (Meteo (tipNor cirrostratus)(vreme ploaie)(vant NE-S)(durata 15-24h)))
(assert (Meteo (tipNor cirrostratus)(vreme inorat)))
(assert (Meteo (tipNor altostratus)(vreme ploaie)(vant NE-S)(durata 24h)))
(assert (Meteo (tipNor altostratus)(vreme inorat)))
(assert (Meteo (tipNor altocumulus)(vreme ploaie)(vant NE-S)(durata 15-20h)))

(facts)

(defrule PrevMeteoPerioada
      ?p1 <- (Meteo {perioada == seara && tipNor == cerRosu})
      ?p2 <- (Meteo {perioada == dimineata && tipNor == ceata})
      ?p3 <- (Meteo {perioada == miezNoapte && tipNor == cumulonimbus})
      =>
      (printout t ?p1.perioada " si " ?p1.tipNor ": cer senin, fara precipitatii dar cu
      ger." crlf)
      (printout t ?p2.perioada " si " ?p2.tipNor ": o zi calduroasa." crlf)
      (printout t ?p3.perioada " si nor de tip " ?p3.tipNor ": o zi plina de zapada."
      crlf))

(defrule PrevMeteoNor
      ?p4 <- (Meteo {tipNor == cumulus})
      ?p5 <- (Meteo {tipNor == cirrocumulus})
      ?p6 <- (Meteo {tipNor == altostratus})
      =>
      (printout t "Nor de tip " ?p4.tipNor ": o zi buna." crlf)
      (printout t "Nor de tip " ?p5.tipNor ": o zi ploioasa." crlf)
      (printout t "Nor de tip " ?p6.tipNor ": o zi innorata." crlf))

(defrule PrevMeteoNorVant
      ?p7 <- (Meteo {tipNor == nimbostratus && vant == SV-N})
      ?p8 <- (Meteo {tipNor == cirrostratus && vant == NE-S} (durata ?name))
      ?p9 <- (Meteo {tipNor == altostratus && vant == NE-S} (durata ?name1))
      =>
      (printout t "Nor de tip " ?p7.tipNor " si vant " ?p7.vant ": o ploaie scurta."
      crlf)
      (printout t "Nor de tip " ?p8.tipNor " si vant " ?p8.vant ": o ploaie peste "
      ?name crlf)
      (printout t "Nor de tip " ?p9.tipNor " si vant " ?p9.vant ": o ploaie peste "
      ?name1 crlf))
(run)
```

Simptome pacienti

```
(deftemplate pacient (slot temperatura)(slot tusa)(slot zonaDurere)(slot
tipDurere)(slot tipRespiratie))

(assert (pacient (temperatura 40) (tusa seaca) (zonaDurere piept) (tipDurere toracica)
(tipRespiratie superficiala )))
(assert (pacient (temperatura 39) (tusa seaca) (zonaDurere piept) (tipDurere arsura)
(tipRespiratie suieratoare )))
```

```
(assert (pacient (temperatura 38) (tusa productiva) (zonaDurere piept) (tipDurere
intepatura) (tipRespiratie suieratoare )))

(defrule citeste
      ?id <- (pacient (temperatura 0)(tusa 0)(zonaDurere 0)(tipDurere 0)(tipRespiratie
      0))
      =>
      (printout t "-temperatura: ")
      (bind ?te (read))
      (printout t "-tusa: ")
      (bind ?tu (read))
      (printout t "-zonaDurere: ")
      (bind ?zd (read))
      (printout t "-tipDurere: ")
      (bind ?td (read))
      (printout t "-tipRespiratie: ")
      (bind ?tr (read))
      (retract ?id)
      (assert (pacient (temperatura ?te) (tusa ?tu) (zonaDurere ?zd) (tipDurere ?td)
      (tipRespiratie ?tr))))
(facts)

(defrule previziune
      ?p1 <- (pacient {temperatura >= 38 && tusa == seaca && zonaDurere == piept &&
      tipDurere == toracica && tipRespiratie == superficiala})
      ?p2 <- (pacient {temperatura >= 39 && tusa == seaca && zonaDurere == piept &&
      tipDurere == arsura && tipRespiratie == suieratoare})
      ?p3 <- (pacient {temperatura >= 38 && tusa == productiva && zonaDurere == piept
      && tipDurere == intepatura && tipRespiratie == suieratoare})
      =>
      (printout t "Pentru simptomele:" ?p1 ", boala este pneumonie." )
      (printout t "Tratamentul este: odihna, antibiotice si consumul lichidelor" crlf)
      (printout t "Pentru simptomele:" ?p2 ":, boala este bronsita." )
      (printout t "Tratamentul este: odihna, aerosoli, evitarea cafelei, alcoolului,
      fumatului." crlf)
      (printout t "Pentru simptomele" ?p3 ":, boala este tbc." )
      (printout t "Tratamentul se aplica cazurilor noi diagnosticate si dureaza 6 luni:
      in primele 2 luni se iau 4 medicamente zilnic, iar 4 luni se iau doar 2
      medicamente, 3 zile/saptamana." crlf))
(run)
```

Recomandare vinuri

```
;peste, paste, legume -> vin alb sec
;somon, preparat pui/curcan/fazan/potarniche/porc -> vin alb demisec
;preparat gratar/cuptor, legume, paste -> vin rosu usor
;orice salata, friptura, garnitura, paste -> vin roze usor

(deftemplate Meniu (slot preparat)(slot garnitura)(slot salata))

(assert (Meniu (preparat peste)(garnitura orez)(salata varza)))
(assert (Meniu (preparat paste)))
(assert (Meniu (preparat pui)(garnitura legume)))
(assert (Meniu (preparat friptura)(garnitura cartofi)(salata castraveti)))
(assert (Meniu (preparat somon)(garnitura legume)))

(facts)

(defrule r1
      ?p <- (Meniu {preparat == paste})
      =>
      (printout t  "La meniurile cu paste va recomandam un vin alb sec, un vin rosu
      usor sau un vin roze usor!" crlf))

(defrule r2
      ?p <- (Meniu {preparat == legume || garnitura == legume})
      =>
      (printout t  "Meniurile cu legume se vor consuma cu un vin alb sec sau un vin
      rosu usor!"crlf))

(defrule r3
      ?p <- (Meniu {salata == varza || salata == castraveti || salata == sfecla})
      =>
      (printout t  "La meniurile cu salata de " ?p.salata " va recomandam un vin roze
      usor!"crlf))
(run)
```

Zboruri

```
(deftemplate zbor (slot plecare)(slot destinatie)(slot distanta))

(assert (zbor (plecare Toronto)(destinatie Vancouver)(distanta 3040)))
(assert (zbor (plecare Toronto)(destinatie LosAngeles)(distanta 3377)))
(assert (zbor (plecare Toronto)(destinatie Chicago)(distanta 610)))
(assert (zbor (plecare NewYork)(destinatie Toronto)(distanta 557)))
(assert (zbor (plecare NewYork)(destinatie Chicago)(distanta 1144)))
(assert (zbor (plecare NewYork)(destinatie Denver)(distanta 2616)))
(assert (zbor (plecare Chicago)(destinatie Denver)(distanta 1474)))
(assert (zbor (plecare Denver)(destinatie Memphis)(distanta 1410)))
(assert (zbor (plecare Denver)(destinatie Houston)(distanta 1350)))
(assert (zbor (plecare Denver)(destinatie LosAngeles)(distanta 1115)))
(assert (zbor (plecare Houston)(destinatie LosAngeles)(distanta 1752)))

(facts)

(printout t crlf "Zborurile cu plecare din New York sunt:" crlf)
(defrule plecareNewYork
    (zbor {plecare == NewYork}(destinatie ?d)(distanta ?dd))
    =>
    (printout t "New York -> " ?d " [" ?dd " km]" crlf))
(run)

(printout t crlf "Zborurile cu cu distante intre 800 si 2200 km sunt:" crlf)
(defrule distanta800-2200km
    (zbor (plecare ?p)(destinatie ?d)(distanta ?dd & :(>= ?dd 800) & :(<= ?dd 2200)))
    =>
    (printout t ?p " -> " ?d " [" ?dd " km]" crlf))
(run)
```

Lab3

Persoana

```
(deftemplate persoana (multislot nume)(slot ochi)(slot par)(slot nationalitate))

(assert (persoana (nume Ion Ionescu)(ochi verzi)(par saten)(nationalitate franceza)))
(assert (persoana (nume Bogdan Popescu)(ochi caprui)(par blond)(nationalitate
germana)))
(assert (persoana (nume Ionel Romanu)(ochi albastri)(par blond)(nationalitate
germana)))
(assert (persoana (nume Cristi Ardelean)(ochi verzi)(par blond)(nationalitate
germana)))
(assert (persoana (nume Mihai Popescu)(ochi purpurii)(par castaniu)(nationalitate
romana)))

(defrule rule1
    (persoana(nume $? ?prenume ?nume)  (ochi verzi)(par saten)(nationalitate franceza))
      =>
      (printout t "Regula 1 -> Nume: "?nume " / Prenume: "?prenume crlf))

(defrule rule2
    (persoana (nume $? ?n ?pn)(ochi ?o & ~albastri)(par ?p & ~negru)(nationalitate
?nt))
    (not (persoana(nume ?nn &~?n ?pnn & ~?pn)(ochi ?o)))
    (not (persoana(nume ?nnn &~?n ?pnnn & ~?pn)(par ?p))) ; ~ = negatie
    =>
    (printout t "Regula 2 -> Nume: "?n " / Prenume: " ?pn crlf))

(defrule rule3
      (persoana (nume $? ?nume1 ?prenume1)(ochi ?ochi1)(par ?par1 &
      ~blond)(nationalitate ?nt))
      (test (or (eq ?par1 saten)(eq ?ochi1 albastri)))
      (persoana (nume $? ?nume2 ?prenume2)(ochi ?ochi2 & verzi)(par ?par2 &
      ?par1)(nationalitate ?nt1))
      (test (or (eq ?par1 saten)(not (eq ?ochi2 caprui))))
      =>
      (printout t "Regula 3 -> " crlf)
      (printout t "Persoana 1: " ?nume1 " " ?prenume1 ", ochi " ?ochi1 ", par " ?par1
      ", nationalitate " ?nt crlf)
      (printout t "Persoana 2: " ?nume2 " " ?prenume2 ", ochi " ?ochi2 ", par " ?par2
      ", nationalitate " ?nt1 crlf))
(run)
```

Retea semantica

```
(defglobal ?*TASK_PRIORITY_1* = 2)
(defglobal ?*TASK_PRIORITY_2* = 1)

(deftemplate relat(slot mode(type STRING))(slot son (type STRING))(slot parent (type
STRING)))

(assert (relat (mode "ako")(son "ballon")(parent "aircraft")))
(assert (relat (mode "ako")(son "pr.driven")(parent "aircraft")))
(assert (relat (mode "ako")(son "jet")(parent "aircraft")))
(assert (relat (mode "ako")(son "blimp")(parent "ballon")))
(assert (relat (mode "ako")(son "blimp")(parent "pr.driven")))
(assert (relat (mode "ako")(son "special")(parent "pr.driven")))
(assert (relat (mode "ako")(son "dc-3")(parent "pr.driven")))
(assert (relat (mode "ako")(son "dc-9")(parent "jet")))
(assert (relat (mode "ako")(son "concorde")(parent "jet")))
(assert (relat (mode "has-shape")(son "ballon")(parent "round")))
(assert (relat (mode "has-shape")(son "blimp")(parent "ellips.")))
(assert (relat (mode "is-a")(son "good year")(parent "blimp")))
(assert (relat (mode "is-a")(son "spirit of stl")(parent "special")))
(assert (relat (mode "is-a")(son "airforce 1")(parent "dc-9")))

(defrule cauta-solutie-is-a
    (declare (salience ?*TASK_PRIORITY_1*))  ; salience stabileste prioritatea pt
    reguli
    (relat (son ?a))
    ?m <- (relat (mode "is-a")(son ?a)(parent ?q))
    ?n <- (relat (mode "ako")(son ?q)(parent ?p))
    =>
    (assert (relat (parent ?p)))
    (retract ?m ?n))

(defrule cauta-solutie
    (declare (salience ?*TASK_PRIORITY_1*))
    (relat (son ?a))
    ?v <- (relat (mode "ako")(son ?a)(parent ?p))
    =>
    (assert (relat (parent ?p)))
    (retract ?v))

(defrule parinte-de-parinte
    (declare (salience ?*TASK_PRIORITY_2*))
    (relat (parent ?p))
    ?v <- (relat (mode "ako")(son ?p)(parent ?q))
    =>
    (retract ?v)
    (assert (relat(parent ?q))))
```

```
(defrule printeaza-solutie
    (declare (salience -1))
    ?n <- (relat (parent ?p))
    =>
    (printout t ?p crlf)
    (retract ?n))
(run)
```

Arbore de decizie

```
(deftemplate intreb (slot intrebare)(slot raspuns))

(assert (intreb (intrebare nil)(raspuns nil)))

(defrule r1
    (intreb (intrebare nil)(raspuns nil))
    =>
    (printout t "It is very big? (yes/no)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "It is very big?")(raspuns ?r))))

(defrule r2.1
    (intreb (intrebare "It is very big?")(raspuns yes))
    =>
    (printout t "Does it have a long neck?" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Does it have a long neck?")(raspuns ?r))))

(defrule r2.2
    (intreb (intrebare "It is very big?")(raspuns no))
    =>
    (printout t "Does it sqeak?" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Does it sqeak?" )(raspuns ?r))))

(defrule r3.1
    (intreb (intrebare "Does it sqeak?")(raspuns yes))
    =>
    (printout t "Guess: mouse" crlf))

(defrule r3.2
    (intreb (intrebare "Does it sqeak?")(raspuns no))
    =>
    (printout t "Guess: squirrle" crlf))

(defrule r4.1
    (intreb (intrebare "Does it have a long neck?")(raspuns yes))
    =>
    (printout t "Guess: giraff" crlf))

(defrule r4.2
    (intreb (intrebare "Does it have a long neck?")(raspuns no))
    =>
    (printout t "Does it have a trunk?" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Does it have a trunk?" )(raspuns ?r))))

(defrule r5.1
    (intreb (intrebare "Does it have a trunk?")(raspuns yes))
    =>
   (printout t "Guess: elephant" crlf))

(defrule r5.2
    (intreb (intrebare "Does it have a trunk?")(raspuns no))
    =>
    (printout t "Does it like to be in water?" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Does it like to be in water?" )(raspuns ?r))))
```

```
(defrule r6.1
    (intreb (intrebare "Does it like to be in water?")(raspuns yes))
    =>
   (printout t "Guess: hippo" crlf))

(defrule r6.2
    (intreb (intrebare "Does it like to be in water?")(raspuns no))
    =>
   (printout t "Guess: rhino" crlf))
(run)
```

Problema reginelor(damelor)

```
(deftemplate Dama(slot x)(slot y)(slot used(allowed-values TRUE FALSE)))

(foreach ?x (create$ 1 2 3 4)
    (foreach ?y (create$ 1 2 3 4)
        (assert (Dama (x ?x)(y ?y)(used FALSE)))
    ))

(defrule rezolvare
    ?id1 <- (Dama (x ?x1)(y ?y1)(used FALSE))
    ?id2 <- (Dama (x ?x2 & ~?x1)(y ?y2 & ~?y1)(used FALSE))
    ?id3 <- (Dama (x ?x3 & ~?x2 & ~?x1)(y ?y3 & ~?y2 & ~?y1)(used FALSE))
    ?id4 <- (Dama (x ?x4 & ~?x3 & ~?x2 & ~?x1)(y ?y4 & ~?y3 & ~?y2 & ~?y1)(used FALSE))
    (test (neq(abs(- ?x1 ?x2)) (abs(- ?y1 ?y2))))
    (test (neq(abs(- ?x1 ?x3)) (abs(- ?y1 ?y3))))
    (test (neq(abs(- ?x1 ?x4)) (abs(- ?y1 ?y4))))
    (test (neq(abs(- ?x2 ?x3)) (abs(- ?y2 ?y3))))
    (test (neq(abs(- ?x2 ?x4)) (abs(- ?y2 ?y4))))
    (test (neq(abs(- ?x3 ?x4)) (abs(- ?y3 ?y4))))
    =>
    (printout t "Dama 1 : x1 " ?x1 " y1 " ?y1 crlf)
    (printout t "Dama 2 : x2 " ?x2 " y2 " ?y2 crlf)
    (printout t "Dama 3 : x3 " ?x3 " y3 " ?y3 crlf)
    (printout t "Dama 4 : x4 " ?x4 " y4 " ?y4 crlf crlf)

    (modify ?id1(used TRUE))
    (modify ?id2(used TRUE))
    (modify ?id3(used TRUE))
    (modify ?id4(used TRUE)))
(run)
```

Automat cafea

```
(deftemplate intreb (slot intrebare)(slot raspuns))

(assert (intreb (intrebare nil)(raspuns nil)))

(defrule start
    (intreb (intrebare nil)(intrebare nil))
    =>
    (printout t "Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r1
    (intreb (intrebare "Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 5c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Aveti 5c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r2
    (intreb (intrebare "Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "Aveti 25c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
```

```
   (assert (intreb (intrebare "Aveti 25c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r1.1
    (intreb (intrebare "Aveti 5c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 10c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 10c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r1.2
    (intreb (intrebare "Aveti 5c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "Aveti 30c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 30c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r2.1
    (intreb (intrebare "Aveti 25c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 30c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 30c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r2.2
    (intreb (intrebare "Aveti 25c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "Aveti 50c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 50c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r3.1
    (intreb (intrebare "Aveti 10c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 15c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 15c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r3.2
    (intreb (intrebare "Aveti 10c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "Aveti 35c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 35c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r4.1
    (intreb (intrebare "Aveti 30c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 35c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 35c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r4.2
    (intreb (intrebare "Aveti 30c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "**Ati depasit 55c si puteti cumpara o cafea." crlf)
    (printout t "Doriti alta cafea? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Doriti alta cafea? (da/nu)")(raspuns ?r))))

(defrule r5.1
    (intreb (intrebare "Aveti 15c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 20c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 20c.Ce moneda introduceti? (5/25)")(raspuns ?r))))
```

```
(defrule r5.2
    (intreb (intrebare "Aveti 15c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "Aveti 40c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 40c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r6.1
    (intreb (intrebare "Aveti 20c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 25c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 25c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r6.2
    (intreb (intrebare "Aveti 20c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "Aveti 45c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 45c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r7.1
    (intreb (intrebare "Aveti 35c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 40c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 40c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r7.2
    (intreb (intrebare "Aveti 35c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "**Ati depasit 55c si puteti cumpara o cafea." crlf)
    (printout t "Doriti alta cafea? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Doriti alta cafea? (da/nu)")(raspuns ?r))))

(defrule r8.1
    (intreb (intrebare "Aveti 40c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 45c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 45c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r8.2
    (intreb (intrebare "Aveti 40c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "**Ati depasit 55c si puteti cumpara o cafea." crlf)
    (printout t "Doriti alta cafea? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Doriti alta cafea? (da/nu)")(raspuns ?r))))

(defrule r9.1
    (intreb (intrebare "Aveti 45c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "Aveti 50c.Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
   (assert (intreb (intrebare "Aveti 50c.Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r9.2
    (intreb (intrebare "Aveti 45c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "**Ati depasit 55c si puteti cumpara o cafea." crlf)
    (printout t "Doriti alta cafea? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Doriti alta cafea? (da/nu)")(raspuns ?r))))
```

```
(defrule r10.1
    (intreb (intrebare "Aveti 50c.Ce moneda introduceti? (5/25)")(raspuns 5))
    =>
    (printout t "**Ati depasit 55c si puteti cumpara o cafea." crlf)
    (printout t "Doriti alta cafea? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Doriti alta cafea? (da/nu)")(raspuns ?r))))

(defrule r10.2
    (intreb (intrebare "Aveti 50c.Ce moneda introduceti? (5/25)")(raspuns 25))
    =>
    (printout t "**Ati depasit 55c si puteti cumpara o cafea." crlf)
    (printout t "Doriti alta cafea? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Doriti alta cafea? (da/nu)")(raspuns ?r))))

(defrule r11.1
    (intreb (intrebare "Doriti alta cafea? (da/nu)")(raspuns da))
    =>
    (printout t "Ce moneda introduceti? (5/25)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Ce moneda introduceti? (5/25)")(raspuns ?r))))

(defrule r11.2
    (intreb (intrebare "Doriti alta cafea? (da/nu)")(raspuns nu))
    =>
    (printout t "Enjoy!" crlf))
(run)
```

Credite bancare

```
(deftemplate intreb (slot intrebare)(slot raspuns))

(assert (intreb (intrebare nil)(raspuns nil)))

(defrule r1
      (intreb (intrebare nil)(raspuns nil))
      =>
      (printout t "Ce tip de credit doriti? (nevoi/imobiliar/business)" crlf)
      (bind ?r (read))
      (assert (intreb (intrebare "Ce tip de credit doriti?
      (nevoi/imobiliar/business)")(raspuns ?r))))

(defrule r1.1
    (intreb (intrebare "Ce tip de credit doriti? (nevoi/imobiliar/business)")(raspuns
    nevoi))
    =>
    (printout t "Aveti 18 ani impliniti? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Aveti 18 ani impliniti? (da/nu)")(raspuns ?r))))

(defrule r1.2
    (intreb (intrebare "Ce tip de credit doriti? (nevoi/imobiliar/business)")(raspuns
    imobiliar))
    =>
    (printout t "Aveti 18 ani impliniti? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Aveti 18 ani impliniti? (da/nu)")(raspuns ?r))))

(defrule r1.3
    (intreb (intrebare "Ce tip de credit doriti? (nevoi/imobiliar/business)")(raspuns
    business))
    =>
    (printout t "Aveti 18 ani impliniti? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Aveti 18 ani impliniti? (da/nu)")(raspuns ?r))))
```

```
(defrule r2.1
    (intreb (intrebare "Aveti 18 ani impliniti? (da/nu)")(raspuns da))
    =>
    (printout t "Aveti toate actele solicitate? (da/nu)" crlf)
    (bind ?r (read))
    (assert (intreb (intrebare "Aveti toate actele solicitate? (da/nu)")(raspuns ?r))))

(defrule r2.2
    (intreb (intrebare "Aveti 18 ani impliniti? (da/nu)")(raspuns nu))
    =>
    (printout t "Credit respins." crlf))

(defrule r3.1
    (intreb (intrebare "Aveti toate actele solicitate? (da/nu)")(raspuns da))
    =>
    (printout t "Credit acceptat" crlf))

(defrule r3.2
    (intreb (intrebare "Aveti toate actele solicitate? (da/nu)")(raspuns nu))
    =>
    (printout t "Credit respins" crlf))
(run)
```

Stramosi

```
(deftemplate Persoana (slot nume)(slot mama)(slot tata)(slot partener)(slot frate))

(deffacts Familie
    (Persoana(nume Ion)(mama Maria)(tata Toma)(partener nil))
    (Persoana(nume Maria)(mama Elena)(tata Mircea)(partener Toma)(frate Geta))

    (Persoana(nume Geta)(mama Elena)(tata Mircea)(partener Gheorghe)(frate Maria))
    (Persoana(nume Gheorghe)(mama nil)(tata nil)(partener Geta))
    (Persoana(nume Sanda)(mama Geta)(tata Gheorghe)(partener nil))

    (Persoana(nume Toma)(mama Ana)(tata Vasile)(partener Maria))
    (Persoana(nume Elena)(mama nil)(tata nil)(partener Mircea))
    (Persoana(nume Mircea)(mama nil)(tata nil)(partener Elena))
    (Persoana(nume Ana)(mama nil)(tata nil)(partener Vasile))
    (Persoana(nume Vasile)(mama nil)(tata nil)(partener Ana)))
(reset)
(facts)

(defrule bunici
      (Persoana(nume ?x)(mama ?y)(tata ?z))
      (Persoana(nume $?y)(mama $?y1)(tata $?z1))
      (Persoana(nume $?z)(mama $?y2)(tata $?z2))
      =>
      (printout t "Bunicii lui " ?x " sunt : " ?y1 " , " ?z1 " , " ?y2 " , " ?z2 crlf))
(run)

(defrule verisoriMama
      (Persoana(nume ?x)(mama ?y & ~nil)(tata ?z & ~nil))
      (Persoana(nume ?v)(mama ?y1 & ~nil)(tata ?z1 & ~nil))
      (Persoana(nume ?y)(mama ?yx & ~nil)(tata ?zx & ~nil))
      (Persoana(nume ?y1)(mama ?yx & ~nil)(tata ?zx & ~nil))
      =>
      (if (neq $?x $?v) then (printout t "Verisori din partea mamei: " $?x " & " $?v
      crlf)))
(run)

(defrule verisoriTata
    (Persoana(nume ?x)(mama ?y & ~nil)(tata ?z & ~nil))
    (Persoana(nume ?v)(mama ?y1 & ~nil)(tata ?z1 & ~nil))
    (Persoana(nume ?z & ~nil)(mama ?z1x & ~nil)(tata ?z1z & ~nil))
    (Persoana(nume ?z1 & ~nil)(mama ?z1x)(tata ?z1z & ~nil))
    =>
   (if (neq ?x ?v) then (printout t "Verisori din partea tatalui: " ?x " & " ?v crlf)))
(run)
```

```
(defrule unchiMatusaMama
      (Persoana(nume ?x)(mama ?y & ~nil)(tata ?z & ~nil))
      (Persoana(nume ?v)(mama ?y1 & ~nil)(tata ?z1 & ~nil))
      (Persoana(nume ?y)(mama ?y1 & ~nil)(tata ?z1 & ~nil))
       =>
      (if (neq ?v ?y) then (printout t "Unchiul/matusa din partea mamei lui " ?x " este
      " ?v crlf)))
(run)

(defrule unchiMatusaTata
      (Persoana(nume ?x)(mama ?y & ~nil)(tata ?z & ~nil))
      (Persoana(nume ?v)(mama ?y1 & ~nil)(tata ?z1 & ~nil))
      (Persoana(nume ?z)(mama ?y1 & ~nil)(tata ?z1 & ~nil))
       =>
      (if (neq ?v ?z) then (printout t "Unchiul/matusa din partea tatalui lui " ?x "
      este " ?v crlf)))
(run)
```

Lab4

Maxim

```
;varianta 1 - prin definirea unei functii
(deffunction calculMaxim ($?lista) ;parametri
    (bind ?maxVal (nth$ 1 ?lista)) ; nth$ extrage el de pe poz 1
    (foreach ?n ?lista
        (if(> ?n ?maxVal) then (bind ?maxVal ?n)))
    (return ?maxVal))

(printout t "Elementul maxim folosind definirea unei functii: " (calculMaxim 4 5 2 9 0
7 3 6 4)crlf)
(run)

;varianta 2 - folosind functii de biblioteca
(printout t "Elementul maxim folosind functia predefinita: " (max 4 5 2 9 0 7 3 6
4)crlf)
(run)

;varianta 3 - cu ajutorul unei reguli
;Aceasta varianta presupune ca numerele sunt introduse sub forma de fapte, pt fiecare
nr inserandu-se cate un fapt in baza de cunostinte
(deftemplate nr (slot n) )
(deftemplate maxim (slot n))
(assert (nr (n 5)))
(assert (nr (n 3)))
(assert (nr (n 7)))
(assert (nr (n 2)))
(assert (nr (n 9)))
(assert (nr (n 0)))
(assert (maxim (n -1)))

(defrule max
    (declare (salience 2))
    ?nr<-(nr(n ?x))
    ?max<- (maxim (n ?m))
    (test (> ?x ?m))
    =>
    (modify ?max (n ?x))
    (retract ?nr))

(defrule print
    (declare (salience 1)) ; stab regula
    (maxim (n ?max))
    =>
    (printout t "Elementul maxim folosind reguli: " ?max crlf))
(run)
```

```
;varianta 4 - folosind liste
(deftemplate lista (multislot elemente))
(deftemplate max1 (slot m))
(deffacts initial
    (max1 (m 0))
    (lista (elemente 1 2 6 4 5 3)))
(reset)

(defrule maxim
    (lista (elemente $? ?u &:(integerp ?u) $?))
    ?idMax<-(max1 (m ?max &:(> ?u ?max)))
    =>
    (modify ?idMax (m ?u)))

(defrule print1
    (declare (salience -1))
    (max1 (m ?max))
    =>
    (printout t "Elementul maxim folosind liste: " ?max crlf))
(watch all)
(run)
```

SEND+MORE=MONEY

```
;varianta 1 - suma
(assert (number 0)(number 1)(number 2)(number 3)(number 4)(number 5)(number 6)(number
7)(number 8)(number 9)(letter S)(letter E)(letter N)(letter D)(letter M)(letter
O)(letter R)(letter Y))

(deftemplate combination ( slot letter (type SYMBOL)) (slot number (type INTEGER)))

(defrule generate-combinations
    (number ?x)
    (letter ?a)
    =>
    (assert (combination (letter ?a)(number ?x))))

(defrule find-solution
    (combination (letter M)(number ?m&:(= ?m 1)))
    (combination (letter O)(number ?o&~?m))
    (combination (letter S)(number ?s&~?o&~?m))
    (combination (letter D)(number ?d&~?s&~?o&~?m))
    (combination (letter E)(number ?e&~?d&~?s&~?o&~?m))
    (combination (letter Y)(number ?y&~?e&~?d&~?s&~?o&~?m))
    (combination (letter N)(number ?n&~?y&~?e&~?d&~?s&~?o&~?m))
    (combination (letter R)(number ?r&~?n&~?y&~?e&~?d&~?s&~?o&~?m))
     (test (= (+(+(* 1000 ?s)(* 100 ?e)(* 10 ?n)?d)
                (+(* 1000 ?m)(* 100 ?o )(* 10 ?r)?e ))
            (+(* 10000 ?m)(* 1000 ?o)(* 100 ?n)(* 10 ?e)?y)))
    =>
    (printout t "A solution is: " crlf)
    (printout t " S=" ?s)
    (printout t " E=" ?e)
    (printout t " N=" ?n)
    (printout t " D=" ?d)
    (printout t " M=" ?m)
    (printout t " O=" ?o)
    (printout t " R=" ?r)
    (printout t " Y=" ?y)
    (printout t crlf)
    (printout t "     " ?s ?e ?n ?d crlf)
    (printout t " +   " ?m ?o ?r ?e crlf)
    (printout t "    " "------" crlf)
    (printout t " = " ?m ?o ?n ?e ?y crlf))

(run)
(reset)
```

```
;varianta 2
(assert (number 0)(number 1)(number 2)(number 3)(number 4)(number 5)(number 6)(number
7)(number 8)(number 9)(letter S)(letter E)(letter N)(letter D)(letter M)(letter
O)(letter R)(letter Y))

(deftemplate combination ( slot letter (type SYMBOL)) (slot number (type INTEGER)))

(defrule generate-combinations
     (number ?x)
     (letter ?a)
     =>
     (assert (combination (letter ?a)(number ?x))))
(defrule find-solution
     (combination (letter M)(number ?m&:(= ?m 1)))
     (combination (letter O)(number ?o&:(= ?o 0)))
     (combination (letter S)(number ?s&:(= ?s 9)))
     (combination (letter D)(number ?d&~?s&~?o&~?m))
     (combination (letter E)(number ?e&~?d&~?s&~?o&~?m))
     (combination (letter Y)(number ?y&~?e&~?d&~?s&~?o&~?m&: (=(mod (+ ?d ?e) 10)?y)))
     (combination (letter N)(number ?n&~?y&~?e&~?d&~?s&~?o&~?m&: (=(+ ?e 1)?n)))
     (combination (letter R)(number ?r&~?n&~?y&~?e&~?d&~?s&~?o&~?m&:(=(mod (+ ?n ?r (/
(+ ?d ?e)10))10)?e)))
     =>
     (printout t "A solution is: " crlf)
     (printout t " S=" ?s)
     (printout t " E=" ?e)
     (printout t " N=" ?n)
     (printout t " D=" ?d)
     (printout t " M=" ?m)
     (printout t " O=" ?o)
     (printout t " R=" ?r)
     (printout t " Y=" ?y)
     (printout t crlf)
     (printout t "   " ?s ?e ?n ?d crlf)
     (printout t " + " ?m ?o ?r ?e crlf)
     (printout t "   " "------" crlf)
     (printout t " = " ?m ?o ?n ?e ?y crlf))
(run)
(reset)
```

Colorare harta

```
(deftemplate combination (slot tara (type STRING))(slot culoare (type STRING)))

(defrule startup
       =>
       (assert (tara Romania)(tara Ungaria)(tara Slovacia)(tara Polonia)(tara
       Cehia)(tara Austria)(tara Germania)(tara Elvetia)(tara Franta)(tara Italia)
       (culoare verde)(culoare albastru)(culoare rosu)(culoare violet)))

(defrule generate-combinations
     (tara ?tara)
       (culoare ?culoare)
       =>
       (assert (combination (tara ?tara)(culoare ?culoare))))

(defrule find-solution
       (combination (tara Romania)(culoare ?RO & :(= verde ?RO)))
       (combination (tara Ungaria)(culoare ?HU & ~?RO ))
       (combination (tara Slovacia)(culoare ?SK & ~?HU))
       (combination (tara Polonia)(culoare ?PL & ~?SK ))
       (combination (tara Cehia)(culoare ?CZ & ~?SK & ~?PL))
       (combination (tara Austria)(culoare ?AT & ~?HU & ~?SK & ~?CZ ))
       (combination (tara Germania)(culoare ?DE & ~?AT & ~?CZ & ~?PL))
       (combination (tara Elvetia)(culoare ?CH & ~?AT & ~?DE & ~?HU))
       (combination (tara Franta)(culoare ?FR & ~?DE & ~?CH))
       (combination (tara Italia)(culoare ?IT & ~?AT & ~?CH & ~?FR))
       =>
```

```
      (printout t "Romania = " ?RO ", ")
      (printout t "Ungaria = " ?HU ", ")
      (printout t "Slovacia = " ?SK ", ")
      (printout t "Polonia = " ?PL ", ")
      (printout t "Cehia = " ?CZ ", ")
      (printout t "Austria = " ?AT ", ")
      (printout t "Germania = " ?DE ", ")
      (printout t "Elvetia = " ?CH ", ")
      (printout t "Franta = " ?FR ", ")
      (printout t "Italia = " ?IT ", " crlf))
(reset)
(run)
```

TWO+TWO=FOUR

```
(assert (number 0)(number 1)(number 2)(number 3)(number 4)(number 5)(number 6)(number
7)(number 8)(number 9)(letter T)(letter W)(letter O)(letter F)(letter U)(letter R))

(deftemplate combination ( slot letter (type SYMBOL)) (slot number (type INTEGER)))

(defrule generate-combinations
      (number ?x)
      (letter ?a)
      =>
      (assert (combination (letter ?a)(number ?x))))

(defrule find-solution
      (combination (letter F)(number ?f&:(= ?f 1)))
      (combination (letter O)(number ?o&~?f))
      (combination (letter U)(number ?u&~?o&~?f))
      (combination (letter R)(number ?r&~?u&~?o&~?f))
      (combination (letter T)(number ?t&~?r&~?u&~?o&~?f))
      (combination (letter W)(number ?w&~?t&~?r&~?u&~?o&~?f))
      (test (= (+(+(* 100 ?t)(* 10 ?w)?o)
      (+(* 100 ?t)(* 10 ?w )?o ))
      (+(* 1000 ?f)(* 100 ?o)(* 10 ?u)?r)))
      =>
      (printout t "A solution is: " crlf)
      (printout t " T=" ?t)
      (printout t " W=" ?w)
      (printout t " O=" ?o)
      (printout t " F=" ?f)
      (printout t " U=" ?u)
      (printout t " R=" ?r)
      (printout t crlf)
      (printout t "   " ?t ?w ?o crlf)
      (printout t " + " ?t ?w ?o crlf)
      (printout t " " "------" crlf)
      (printout t " = " ?f ?o ?u ?r crlf crlf))
(run)
(reset)
```

<u>Lab5</u>

Diagrama de stare

```
(deftemplate masinaSpalat
    (slot apa (allowed-values TRUE FALSE))
    (slot electricitate (allowed-values TRUE FALSE))
    (slot detergent (allowed-values TRUE FALSE))
    (slot stare))

(assert (masinaSpalat (apa FALSE)(electricitate TRUE)(detergent FALSE)(stare init)))

(defrule alimentareEnergie ?id<-(masinaSpalat (stare init))
      =>
      (facts)
      (retract ?id)
      (assert (masinaSpalat (apa FALSE)(electricitate TRUE)(detergent FALSE)(stare
      cuElectricitate)))
      (printout t "Masina de spalat alimentata cu electricitate "  crlf)
      (facts))

(defrule alimentareApa ?id<-(masinaSpalat (stare cuElectricitate))
      =>
      (facts)
    (retract ?id)
      (assert (masinaSpalat (apa TRUE)(electricitate TRUE)(detergent FALSE)(stare
      cuApaSiElectricitate)))
      (printout t "Masina de spalat alimentata cu electricitate si apa"  crlf)
      (facts))

(defrule alimentareDetergent ?id<-(masinaSpalat (stare cuApaSiElectricitate))
      =>
      (facts)
      (retract ?id)
      (assert (masinaSpalat (apa TRUE)(electricitate TRUE)(detergent TRUE)(stare
      functionabila)))
     (printout t "Masina de spalat alimentata cu electricitate, apa si detergent" crlf)
      (facts))

(defrule prespalare ?id<-(masinaSpalat (stare functionabila))
      =>
      (facts)
      (retract ?id)
      (assert (masinaSpalat (apa TRUE)(electricitate TRUE)(detergent TRUE)(stare
      prespalare)))
      (printout t "Masina de spalat in prespalare "  crlf)
      (facts))

(defrule spalare ?id<-(masinaSpalat (stare prespalare))
      =>
      (facts)
      (retract ?id)
      (assert (masinaSpalat (apa TRUE)(electricitate TRUE)(detergent TRUE)(stare
      spalare)))
      (printout t "Masina de spalat in stare de spalare "  crlf)
      (facts))

(defrule stoarcere ?id<-(masinaSpalat (stare spalare))
    =>
      (facts)
      (retract ?id)
      (assert (masinaSpalat (apa TRUE)(electricitate TRUE)(detergent TRUE)(stare
      stoarcere)))
      (printout t "Masina de spalat in stare de stoarcere "  crlf)
      (facts))

(defrule finalizareCiclu ?id<-(masinaSpalat (stare stoarcere))
    =>
    (facts)
    (retract ?id)
   (assert (masinaSpalat (apa TRUE)(electricitate TRUE)(detergent TRUE)(stare FINALA)))
    (printout t "Masina de spalat in stare finala "  crlf)
    (facts))
(run)
```

Blocuri (cu stive)

```
(deftemplate goal(slot move)(slot on-top-of))

(deffacts initial-data
    (stack A B C D)
    (stack E F G)
    (goal (move C) (on-top-of G)))
(reset)
```

```
(deffunction print-moved-message (?A ?B)
    (printout t  ?A " moved on top of " ?B "." crlf ))

; Operatii de baza pentru stiva
(defrule push-value
    ?push-value <- (push-value ?value)
    ?stack <- (stack $?rest)
    =>
    (retract ?push-value ?stack)
    (assert (stack ?value $?rest))
    (printout t "Pushing value " ?value crlf))

(defrule pop-value-valid
    ?pop-value <- (pop-value)
    ?stack <- (stack ?value $?rest)
    =>
    (retract ?pop-value ?stack)
    (assert (stack ?rest))
    (printout t "Popped value " ?value " " $?rest crlf))

(defrule pop-value-invalid
    ?pop-value <- (pop-value)
    (stack)
    =>
    (retract ?pop-value)
    (printout t "Popping from empty stack" crlf))

;Mutari pt blocuri
(defrule move-directly
    ?goal <- (goal (move ?block1)(on-top-of ?block2))
    ?stack1 <- (stack ?block1 $?rest1)
    ?stack2 <- (stack ?block2 $?rest2)
    =>
    (retract ?goal ?stack1 ?stack2)
    (assert (stack $?rest1))
    (assert (stack ?block1 ?block2 ?rest2))
    (print-moved-message ?block1 ?block2))

(defrule move-to-floor
    ?goal <- (goal(move ?block1)(on-top-of floor))
    ?stack1 <- (stack ?block1 $?rest)
    =>
    (retract ?goal ?stack1)
    (assert (stack $?block1))
    (assert (stack $?rest))
    (print-moved-message ?block1 "floor"))

(defrule clear-upper-block
    (goal (move ?block1))
    (stack ?top $?X ?block1 $?Y)
    =>
    (assert (goal (move ?top)(on-top-of floor))))

(defrule clear-lower-block
    (goal (on-top-of ?block1))
    (stack ?top $?X ?block1 $?Y)
    =>
    (assert (goal (move ?top)(on-top-of floor))))

(facts)
(run)
(facts)
```

Triunghiuri

```
(deffunction det-distance ($?lista)
    (bind ?x1 (nth$ 1 ?lista))
    (bind ?y1 (nth$ 2 ?lista))
    (bind ?x2 (nth$ 3 ?lista))
    (bind ?y2 (nth$ 4 ?lista))
    (bind ?x3 (nth$ 5 ?lista))
    (bind ?y3 (nth$ 6 ?lista))
    (printout t ?x1 "," ?y1 crlf)
    (printout t ?x2 "," ?y2 crlf)
    (printout t ?x3 "," ?y3 crlf)
    (bind ?d1 (sqrt(+(** (- ?x2 ?x1)2)
                (**(- ?y2 ?y1) 2))))
    (bind ?d2 (sqrt(+(** (- ?x3 ?x1)2)
                (**(- ?y3 ?y1) 2))))
    (bind ?d3 (sqrt(+(** (- ?x3 ?x2)2)
                (**(- ?y3 ?y2) 2))))
    (printout t "Distanta intre A si B este " ?d1 crlf)
    (printout t "Distanta intre C si A este " ?d2 crlf)
    (printout t "Distanta intre C si B este " ?d3 crlf)
    (if(or (=(* ?d1 ?d1) (+ (* ?d2 ?d2) (* ?d3 ?d3)))
           (=(* ?d2 ?d2) (+ (* ?d1 ?d1) (* ?d3 ?d3)))
           (=(* ?d3 ?d3) (+ (* ?d2 ?d2) (* ?d1 ?d1)))
       )then
           (printout t "Triunghi dreptunghic" crlf)
           (bind ?b dreptunghic)
       else
           (if(= ?d1 ?d2 ?d3) then
                 (printout t "Triunghi echilateral" crlf)
                 (bind ?b echilateral)
           else
                 (if(or (=?d1 ?d2)(=?d3 ?d2)(=?d1 ?d3))then
                 (printout t "triunghi isoscel" crlf)
                 (bind ?b isoscel)
              else
                       (printout t "Triunghi oarecare" crlf)
                       (bind ?b oarecare)
              )
           )
    )
    (return ?b))

(deftemplate figuri(slot tip)(multislot coordonate))

(assert(figuri(tip triunghi) (coordonate  2 7 7 2 2 5)));oarecare
(assert(figuri(tip triunghi) (coordonate  3 4 2 5 1 6)));isoscel
(assert(figuri(tip triunghi) (coordonate  5 0 -2 4 2 -2)));dreptunghic
(assert(figuri(tip triunghi) (coordonate  0 0 3 3 3 3)));dreptunghic

(defrule determinare_tip_triunghi
    ?id<- (figuri(tip ?m)(coordonate $?lista))
    =>
    (printout t crlf)
    (bind ?t (det-distance $?lista))
    (modify ?id (tip ?t))
    (retract ?id))

(run)
(facts)
```

Operatii cu cozi

```
(deffacts initial-data
    (queue 7 5 8 1 4 6)
    (pop-value)
    (push-value 12)
    (push-value 10)
)
(reset)
```

```
(defrule push-value
    ?push-value <- (push-value ?value)
    ?queue <- (queue $?rest)
    =>
    (retract ?push-value ?queue)
    (assert (queue $?rest ?value))
    (printout t "Pushing value " ?value crlf))

(defrule pop-value-valid
    ?pop-value <- (pop-value)
    ?queue <- (queue ?value $?rest)
    =>
    (retract ?pop-value ?queue)
    (assert (queue ?rest))
    (printout t "Popped value " ?value " Rest: " $?rest crlf))

(defrule pop-value-invalid
    ?pop-value <- (pop-value)
    (queue)
    =>
    (retract ?pop-value)
    (printout t "Popping from empty queue" crlf))
(facts)
(run)
(facts)
```

Lab6

Combinari

```
(deftemplate number(slot value))
(assert(number(value 1)))
(assert(number(value 2)))
(assert(number(value 3)))

(defrule same-rule
    ?fact1 <- (number (value ?n1))
    ?fact2 <- (number (value ?n2))
    ?fact3 <- (number (value ?n3))
    (test (neq ?fact1 ?fact2 ?fact3))
    (test (neq ?n1 ?n2))
    (test (neq ?n3 ?n2))
    (test (neq ?n1 ?n3))
    =>
    (printout t ?fact1.value ?fact2.value ?fact3.value crlf))
(run)

(deffacts data (set a b c) (combination_length 3))
(reset)

(defrule initialize_combination
    (set $? ?x $?) => (assert (combination ?x)))

(defrule extend_combination
    (combination_length ?k)
    (combination $?x&:(<(length$ $?x)?k))
    (set $? ?e&:(not (member$ ?e $?x))$?)
    =>
    (assert (combination ?e $?x)))

(defrule print
    (combination_length ?k)
    (combination $?x&:(=(length$ $?x)?k))
    =>
    (printout t $?x crlf))
(run)
```

Intersectie

```
(deftemplate element (slot item)(slot multime)(slot folosit (default FALSE)))

(assert (element (item 1)(multime A)))
(assert (element (item 2)(multime A)))
(assert (element (item 3)(multime A)))
(assert (element (item 4)(multime A)))
(assert (element (item 5)(multime A)))
(assert (element (item 6)(multime A)))
(assert (element (item 7)(multime A)))
(assert (element (item 8)(multime A)))
(assert (element (item 9)(multime A)))
(assert (element (item 10)(multime A)))
(assert (element (item 2)(multime B)))
(assert (element (item 4)(multime B)))
(assert (element (item 6)(multime B)))
(assert (element (item 8)(multime B)))
(assert (element (item 10)(multime B)))
(assert (element (item 12)(multime B)))
(assert (element (item 14)(multime B)))
(assert (element (item 16)(multime B)))

(defrule intersectie
    ?idmf <-(element (item ?i)(multime ?m)(folosit FALSE))
    ?idmfo<-(element (item ?i)(multime ?m1 & ~?m)(folosit FALSE))
    =>
    (modify ?idmf (folosit TRUE))
    (modify ?idmfo (folosit TRUE))
    (assert(element (item ?i)(multime I)))
    (printout t "Elementele multimii I: "?i crlf))
(run)

(deftemplate suma(slot valoare))
(assert (suma (valoare 0)))

(defrule duplicat
    ?idm<-(element (item ?i)(multime ?m & ~R)(folosit FALSE))
    ?idmm1<-(element(item ?i)(multime ?m1 & ~?m & ~R)(folosit FALSE))
    =>
    (modify ?idm (folosit TRUE))
    (modify ?idm1 (folosit TRUE))
    (assert (element (item ?i)(multime R))))

(defrule reuniune
    ?idr<-(element(item ?i)(multime ?M & ~R)(folosit FALSE))
    =>
    (assert (element (item ?i)(multime R)))
    (modify ?idr (folosit TRUE)))

(defrule printare
    (element (item ?i)(multime R)(folosit FALSE))
    =>
    (printout t "Elementele multimii R = "?i crlf))

(defrule CalculSuma
    ?idi <- (element(item ?i))
    ?idvs <- (suma(valoare ?vs))
    (element (item ?i &: (> ?i 10 ) &: (< ?i 50)))
    =>
    (modify ?idvs (valoare (+ ?i ?vs)))
    (retract ?idi))

(defrule afisaresuma
    (suma (valoare ?vs))
    =>
    (printout t "Suma = "?vs crlf))
(run)
```

Multimi

```
(deffacts multimi
    (multime A 1 2 7)
    (multime B 3 4 5 6))

;(deftemplate suma(slot sum))
(deftemplate cardinal(slot card))

(deffacts reuniune (reuniune AB))
(reset)

;(assert (suma(sum 0)))
(assert (cardinal(card 0)))

(defrule descompunere
    (declare (salience 2))
    ?n <- (multime ?A ?p $?rest)
    (test (not (eq ?p nil)))
    =>
    (retract ?n)
    (assert (element ?p))
    (assert (multime ?A $?rest)))

(defrule reuniune
    (declare(salience 1))
    ?n <- (element ?p)
    ?m <- (reuniune ?A $?rest)
    =>
    (retract ?m ?n)
    (assert (reuniune ?A ?p $?rest)))

(defrule cardinal
    ;(declare(salience -2))
    ?idvs <- (cardinal (card ?vs))
    ?idve <- (reuniune ?A ?p $?rest)
    =>
    (retract ?idve)
    (modify ?idvs (card (+ ?vs 1)))
    (assert (reuniune ?A $?rest)))

/*
(defrule suma
      (declare(salience -2))
      ?idvs <- (suma (sum ?vs))
      ?idve <- (reuniune ?A ?p&:(integerp ?p)$?rest)
      =>
      (retract ?idve)
      (modify ?idvs (sum (+ ?vs ?p)))
      (assert (reuniune ?A $?rest)))
*/
(run)
(facts)
```