Medii de programare în Inteligența Artificială

JESS

Lect. univ. dr. Mihaela Colhon

http://inf.ucv.ro/~ghindeanu

JESS = shell pentru sisteme expert

- Jess este un un limbaj de scripting scris in totalitate in Java. A fost dezvoltat de Ernest Friedman-Hill la Sandia National Laboratories in Livermore, California.
- Jess permite programarea bazată pe reguli necesară scrierii de sisteme expert, fiind astfel referit drept un shell pentru sisteme expert.
- Jess poate procesa un numar mare de reguli fiind **unul dintre cele mai rapide motoare de prelucrare a regulilor**. Acest mecanism este utilizat ca o varianta perfectionată a shell-ului CLIPS (C Language Integrated Production System) – este considerat „o clonă” a CLIPS-ului.
- Folosind Jess se pot realiza aplicatii Java în domeniul inteligentei artificiale pe baza unui set de cunostinte acumulate sub forma unor **reguli declarative** folosite pentru a obține concluzii și a face inferențe.

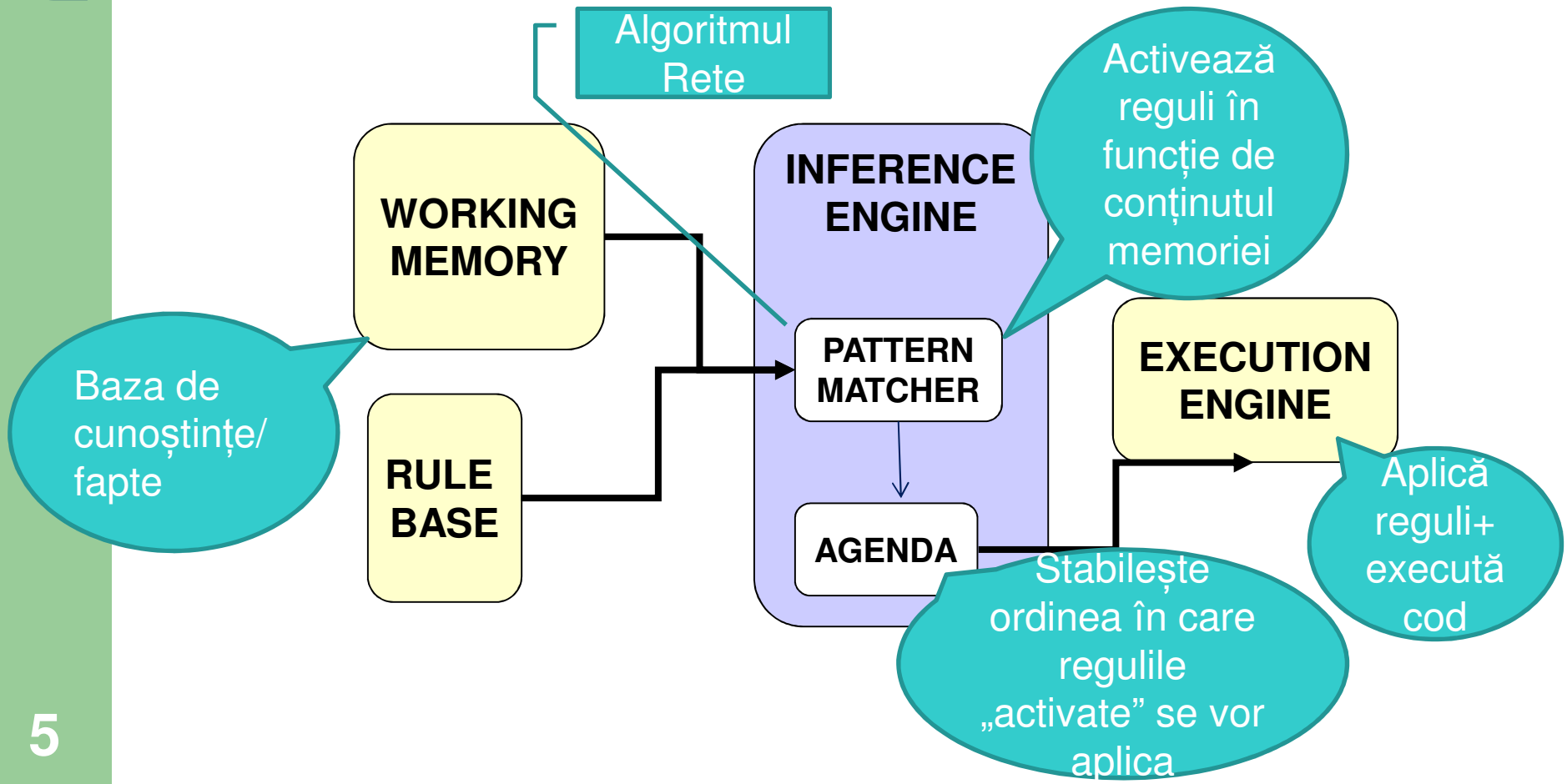
www.jessrules.com

Jess poate fi utilizat ca program stand-alone sau poate fi folosit pentru a dezvolta aplicații Java care folosesc cunoștințe în forma regulilor declarative pentru a obține concluzii noi și a face inferențe.

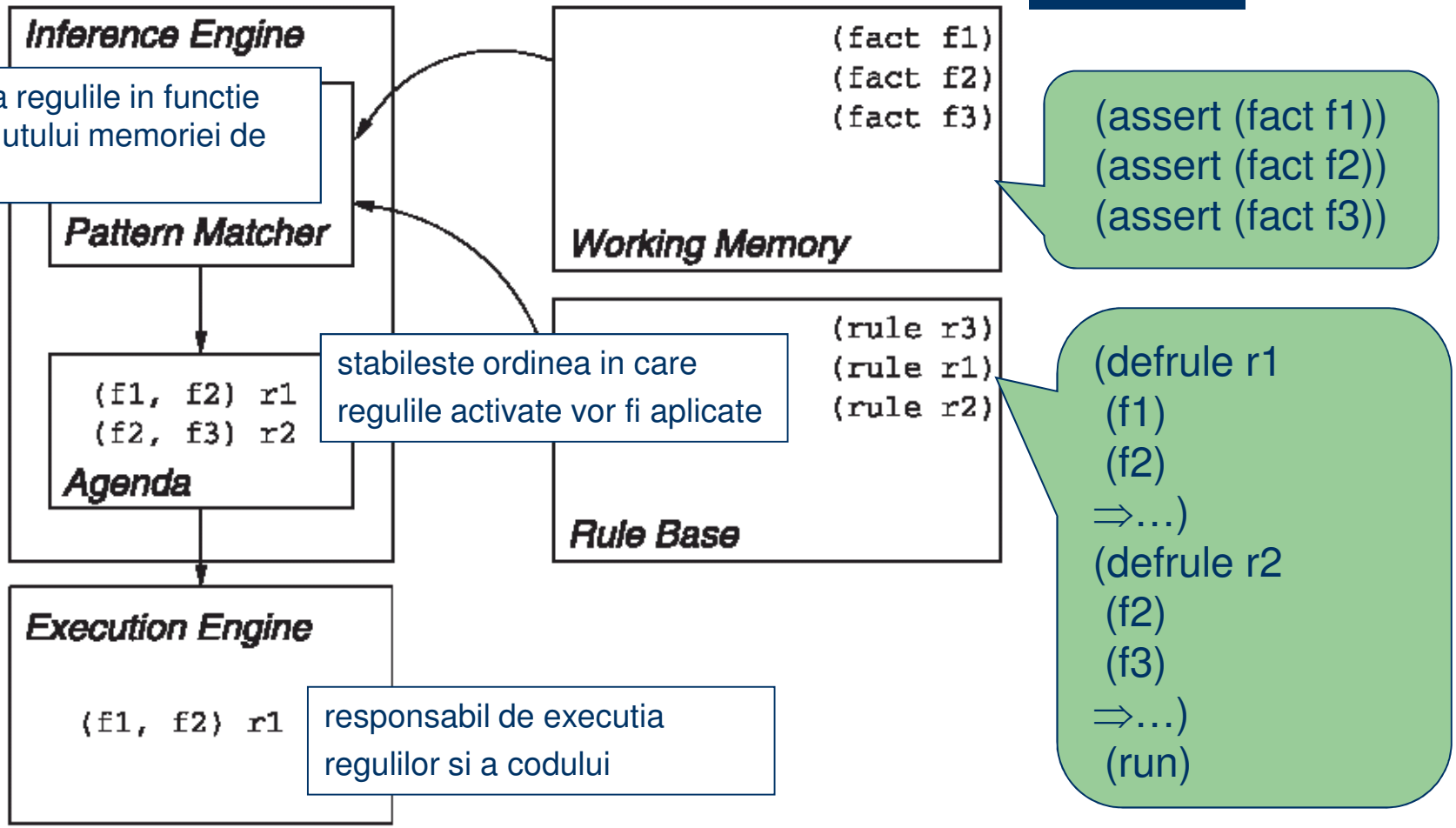
Jess – shell pentru SE bazate pe reguli

Sistemele bazate pe reguli sunt folosite pentru a modela **expertiza umană** prin implementarea de raționamente bazate pe reguli aplicate pe o mulțime de date. Structura unui sistem bazat pe reguli conține o mulțime de fapte numită *memoria de lucru* a sistemului în care sunt stocate faptele ce se cunosc a fi adevărate, o *baza de reguli* ce conține regulile folosite în inferență și un *motor de inferență* care are rolul de a decide ordinea în care se aplică regulile activate de datele de intrare.

Arhitectura Jess



Arhitectura Jess bazata pe reguli



Motorul de inferență Jess

Motorul de inferență Jess folosește o variantă îmbunatățită a *algoritmului Rete* (Rete în latină înseamnă rețea) care decide aplicarea regulilor în funcție de faptele existente în memoria de lucru.
Un sistem expert bazat pe reguli scris in Jess este un program dirijat de datele încărcate în memorie (**data-driven**) care hotăresc execuția prin intermediul motorului de inferente.
Acesta este un exemplu despre modul in care **Jess difera de limbaje procedurale** cum ar fi Java si C. În limbajele procedurale, execuția se poate realiza și fără date. Instructiunile sunt suficiente pentru a cauza executia. De exemplu, instructiunea

```
System.out.print(“Hello world!”)
```

poate fi executata imediat în Java. Este o instrucțiune completă care nu are nevoie de date suplimentare pentru a declanșa execuția. În schimb, în Jess, datele sunt necesare pentru a cauza execuția regulilor.

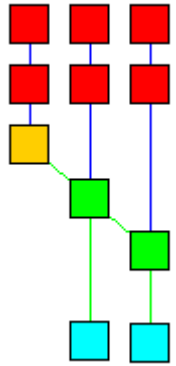
Clasa Rete

Modalitatea de a accesa motorul de inferente Jess o reprezintă clasa Rete din pachetul jess [inclusă într-o aplicație Java]:

```
import java.net.*
public class TestJess {

    public static void main(String[] args) {
        Rete r = new Rete(); //se instantiaza un nou MI
        r.executeCommand(”jess-command”);
        r.reset(); //se reseteaza motorul
        r.run(); //se executa comanda
    }
}
```

Algoritmul Rete



Algoritmul Rete ofera o descriere logică generalizată a unei implementări de funcționalitate responsabilă cu potrivirea faptelor în conformitate cu producțiile (regulile) declarate într-un sistem ce folosește **Pattern Matcher** (o categorie de motoare de inferență bazate pe reguli).
O producție e constituită din una sau mai multe condiții corelate la un set de acțiuni care trebuie sa fie executate pentru fiecare set complet de fapte ale căror atribute se potrivesc cu cele exprimate în condiții.
Algoritmul **Rete** prezintă următoarele caracteristici majore:
•Elimină anumite tipuri de reduntanță prin folosirea partajată a nodurilor
•Salvează potriviri parțiale ceea ce permite sistemelor de producție să evite reevaluarea completă a tuturor faptelor de fiecare dată când se fac schimbări în memoria de lucru a sistemului.

Jess ↔ Java

- Jess mosteneste de la Java pe langa caracteristicile de portabilitate si securitate si posibilitatea dezvoltarii aplicatiilor in programe stand-alone sau in browsere (applet-uri).
- Jess poate fi de asemenea folosit in medii multithread, fiecare obiect **Jess.Rete** reprezentand un motor de inferenta independent care poate fi rulat in paralel cu alte obiecte de acest tip.
- Cei mai importanti pasi in dezvoltarea unei aplicatii folosind shell-ul Jess vizeaza stabilirea arhitecturii optime pentru aplicatie si anume stabilirea ponderii codul scris in Jess si a celui in Java. Jess permite dezvoltarea de aplicatii (cu sau fara interfata GUI) fara a fi necesara compilarea nici unui rand de cod Java, sau invers se pot scrie aplicatii Jess folosind un minim cod Jess. Structura unei aplicatii tipice se situeaza undeva la mijloc intre aceste doua extreme astfel incat sistemul expert e scris in Jess iar modulul de interfata utilizator e construit folosind Java API (Application Programming Interfaces).

Jess + Java = Great Expert Systems

Exista mai multe arhitecturi Java si Jess, iar mai jos le enumeram in ordine crescatoare a ponderii codului Java implicat in dezvoltarea aplicatiei:

- **100%Jess** Exclusiv cod Jess.
- **Jess+Java API** Cod Jess dar cu interfata construita folosind Java API
- **most Jess + Java** Majoritar cod Jess dar si cod specializat Java in care sunt definite comenzi Jess.
- **main Jess + Java** Pondere egala intre codul Jess si Java. Codul Java este folosit pentru construirea interfetei si executie de comenzi Jess dar functia *main*() a aplicatiei (punctul de intrare in program) este data de Jess.
- **Jess + Java** Pondere egala intre codul Jess si Java. Codul Java este folosit pentru construirea interfetei si executie de comenzi Jess iar functia *main*() este definita de programator.
- **most Java** In marea majoritate cod Java, care incarca cod Jess in timpul executiei (laseaza in executie programe Jess).
- **100%Java** In totalitate cod Java, iar entitatile Jess sunt manipulate prin intermediul interfetei (nu este suportata in totalitate pentru versiunea Jess61b1)

Paradigma Jess

Sunt mai multe moduri de a reprezenta cunostinte in Jess:

- **reguli** – care in principal sunt pentru cunostinte euristice bazate pe experienta
- **fapte**
- **functii** – pentru cunostinte procedurale
- **programare orientata obiect**, pentru cunostinte procedurale

Paradigma declarativă folosită în Jess, aplică în mod continuu o colecție de reguli unei colecții de fapte printr-un proces numit **potrivire de sabloane (Pattern Matcher)**. Regulile pot modifica colecția de fapte sau pot executa orice cod Java.

Jess poate procesa un numar mare de reguli (e unul dintre cele mai rapide motoare de inferenta bazate pe reguli) fiind un produs software extrem de stabil.

Programarea logica

- Este o paradigma care pleaca de la ideea specificarii unei probleme in limbajul *logicii cu predicate de ordin I* si utilizarea *demonstrarii automate de teoreme* pentru rezolvarea acelei probleme. Se trece astfel de la *programarea imperativa* “cum se rezolva” la *programarea declarativa* “ceea ce trebuie rezolvat”.
- Deoarece demonstrarea automata de teoreme are niste dezavantaje legate de caracterul sau prea general, au fost necesare niste restrictii. De exemplu, in Prolog se lucreaza cu clauze Horn care reduc complexitatea calculului si nedeterminarea, introducand aspecte de control.

Programarea funcțională

- In limbajele functionale, **functiile** sunt componentele de baza ale unui program, asa cum **intructiunile** sunt parte constructiva in programele procedurale, **clauzele** in programarea logica si **obiectele** in programarea orientata obiect. **Un program functional este o compunere de functii.**
- Cel mai cunoscut reprezentant al paradigmei de programare functionala este limbajul Lisp. Deoarece Lisp este un limbaj interpretat dupa lansarea in executie a sistemului Lisp se intr-o intr-o bucla in care interpretorul asteapta introducerea unei expresii Lisp.
- Semnalul ca interpretorul Lisp asteapta instructiuni pentru a fi executate este facut prin aparitia prompterului. Dupa ce utilizatorul a introdus o expresie Lisp interpretorul o evalueaza si daca aceasta expresie este corecta lexical, sintactic si semantic tipareste rezultatul evaluarii ei. Se spune ca interpretorul Lisp functioneaza intr-o bucla CITESTE-EVALUEAZA-TIPARESTE (*read-eval-print*).

Jess vs. Lisp

- La fel ca si CLIPS, Jess-ul are la baza limbajul Lisp. Desi sintaxa Jess este similara lui CLIPS, totusi Jess si CLIPS sunt doua sisteme diferite.
- Spre deosebire de Lisp, Jess nu suportă listele imbricate (liste care conțin alte liste). Acest lucru afectează modul în care codul Java este reprezentat în Jess și vice-versa. De exemplu, să considerăm codul Java:

```
class JavaTest {
private int nr;
public String label;
public String getLabel() {return label;}
}
```

Cod Java în Jess

Clasa JavaTest poate fi scrisa folosind următoarele fapte Jess:

```
(class (name JavaTest) (fields nr label) (methods
getLabel))
(field (name nr) (class JavaTest) (access private)
       (type int))
(field (name label) (class JavaTest) (access public)
       (type String))
(method (name getLabel) (class JavaTest) (access public)
       (type String) (body "return label;"))
```

În prealabil, pentru a putea fi folosite, aceste fapte trebuiesc mai întâi declarate:

```
(deftemplate field (slot name) (slot class)
       (slot access) (slot type))
```

Jess vs. Prolog

Desi sunt amandoua limbaje declarative, Prolog-ul si Jess-ul care este un sistem bazat pe algoritmul Rete sunt foarte diferite. Conceptul central al Prolog-ului este inlantuirea inapoi. Astfel, pe baza regulilor:

```
mama(ana, maria).
copil(X, Y):- mama(Y, X).
```

cineva ar putea sa testeze daca **copil(maria, ana)** este adevarat. Pentru aceasta, Prologul aplica regula si ca rezultat va cauta sa verifice daca **mama(ana, maria)** este adevarat. Pentru o consultare ulterioara, din nou se va aplica regula, adica se va verifica relatia **mama(ana,maria)**.

Nu acelasi lucru se intampla pentru consultarile Jess datorita faptului ca in deductii se foloseste inlantuirea inainte.

Jess vs. Prolog

Codul Jess

```
(assert (mama ana maria))
(defrule copil (mama ?X ?Y) => (assert (copil ?Y ?X)))
(watch facts)
(run)
```

nu este definit pentru a raspunde la interogari relative la relatia **copil**, ci este intentionat de a calcula ce instanta a relatiei **copil** se obtine pe baza relatiei **mama(ana, maria)** asertata.

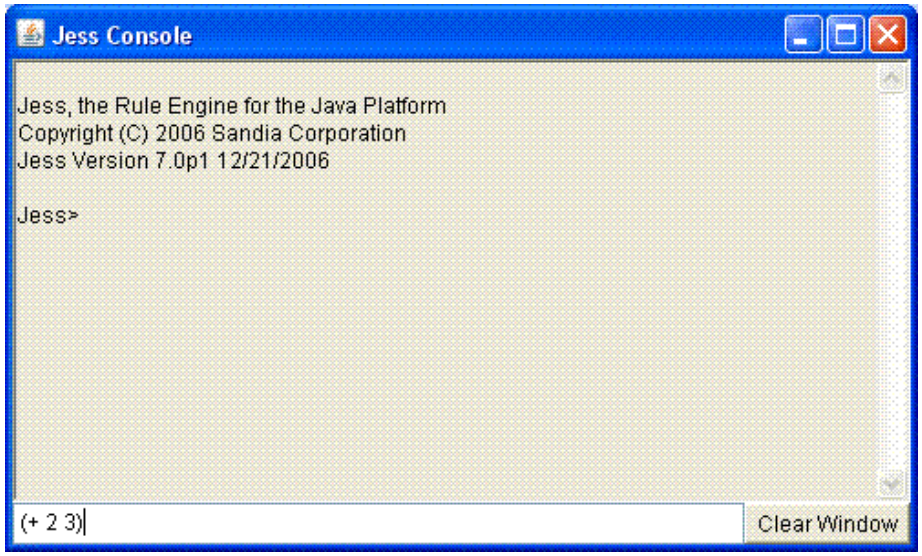
```
==> f-1 (MAIN:: copil maria ana)
1
```

Odata ce relatia **copil(maria,ana)** este obtinuta, la executia urmatoare a codului nu se va mai aplica regula din nou.

Consola Jess

Desi Jess a fost initial dezvoltat ca o librarie care poate fi integrata in aplicatii Java, Jess permite in plus si un mod de lucru interactiv la prompter Jess> (folosind comanda “java -classpath jess.jar jess.Main”)
La prompterul Jess se pot executa pe rand instructiuni sau se pot rula programa .clp dupa cum urmeaza:

```
Jess> (printout t "Hello, World!" crlf)
Hello, World!
Jess> (batch hello.clp)
Hello, World!
```



Limbajul Jess

O unitate fundamentala a fiecarui program Jess este lista. Elementele unei liste sunt incluse intre parantezele rotunde (si) iar primul element al listei se numeste *capul listei*. Dupa cum se va vedea, capul listei in marea majoritate a cazurilor are un rol special. In afara de utilizarea lor ca date pentru prelucrari, listele sunt folosite in Lisp si pentru scrierea programelor.
Orice apel de functie se scrie ca o lista folosind notatia prefixata astfel incat capul listei este numele functiei iar celelalte elemente reprezinta valorile de apel ale functiei.
Observatie *Regulile se scriu tot sub forma de liste, capul listei fiind in acest caz un cuvant rezervat*, **defrule**.

Jess. Elemente de bază

- ; se foloseste pentru cometariu
- Exemplu crearea și asignarea variabilei `?x`:

```
(bind ?x 10)
```

- Variabilele simple se noteaza cu `?` ca început de identificator sau `$`.
- Definirea funcțiilor se face cu ajutorul comenzii `deffunction`. Exemplu, definirea funcției `max`:

```
(deffunction max (?a ?b)
        (if (> ?a ?b) then ?a else ?b))
```

Funcții în Jess

Orice lista in Jess se presupune ca este un apel de functie, capul listei avand rolul de a identifica numele functiei. Jess vine cu un numar mare de functii built-in care implementeaza functii matematice, manipuleaza stringuri de caractere, controleaza executia programului, permit accesul la functiile Java API, etc. De exemplu, daca dorim sa ni se returneze rezultatul operatiei aritmetice 2+3 vom scrie:

```
Jess> (+ 2 3)
5
```

Una dintre cele mai utilizate functii este cea de tiparire la iesirea standard, **printout**. Primul argument al functiei indica unde se sa face tiparirea (la ecran, intr-un fisier, la imprimanta) iar urmatoarele valori sunt cele care vor fi tiparite, fara a se intercala spatii intre ele:

```
Jess> (printout t "Rezultatul este" (+ 2 3) "!" crlf)
Rezultatul este5!
```

O alta functie utila si des utilizata este **batch**. Aceasta evalueaza si executa cod Jess mai precis programe Jess. De exemplu, pentru a rula fisierul **hello.clp** care apartine directorului **examples** din pachetul de distributie Jess se da comanda:

```
Jess> (batch examples/hello.clp)
```

Variabile Jess

La fel ca variabilele Java, variabilele Jess pot retine o singura valoare, insa spre deosebire de primele nu sunt dependente de un tip anume. Asta inseamna ca pe parcursul existentei sale, o variabila poate fi unificata cu date de tipuri diferite, cum ar fi numere, stringuri, liste, etc. In general variabilele in Jess sunt identificate cu simboluri prefixate de caracterul **?** (astfel, ? devine parte a numelui variabilei). O variabila nu trebuie declarata pentru a fi ulterior folosita. Ea se creaza in momentul in care i se atrbuie o valoare. Atribuirea de valori pentru variabilele

Jess se face cu ajutorul functiei `bind`, ca in exemplul de mai jos:

```
Jess> (bind ?var “string”)
“string”
Jess> (bind ?var 5)
5
Jess> (+ ?var 2)
7
```

Pentru a crea o lista de elemente (care sa nu fie reprezentarea unui apel de functie) se poate folosi functia `create$` cu urmatoarea sintaxa:

```
(bind nume_variabila (create$ lista_elemente))
```

De exemplu:

```
Jess> (bind ?work_days (create$ mon tue wed thu fri))
(mon tue wed thu fri)
```

Execuția fișierelor .clp

- Command line:

```
java -classpath jess.jar jess.Main
```

- or start an applet console.html
-
```
Jess> (batch apples.clp)
TRUE
Jess> (reset)
TRUE
Jess> (run)
Please enter the apple characteristics:
   ...
```

-
```
Jess> (exit)
```

Definirea faptelor în Jess

Faptele sunt bazate pe notiunea de template cu atribute:

```
(deftemplate masina
        (slot marca) (slot culoare) (slot viteza-maxima)
        (slot pret))
```

Aici e definita o masina cu atributele culoare, viteza-maxima si pret. Un fapt ar arata de genul:

```
(deffacts masini
        (masina (marca ford) (culoare rosie)
        (viteza-maxima 100) (pret 10000)))
```

Pentru a adauga un fapt dinamic se poate folosi `assert`, cu un fapt ca parametru.

Definirea regulilor în Jess

Definirea unei reguli simple:

```
(defrule vopseste-albastru-masinile-rosii
        ?m <- (masina (marca ?marca) (culoare rosie)
                      (viteza-maxima ?vmax) (pret ?pret))
=>
        (retract ?m)
        (assert (masina (marca ?marca) (culoare albastra)
        (viteza-maxima ?vmax) (pret ?pret)))
```

Explicație: Se găsesc mașinile care au culoarea roșie și se schimbă culoarea acestora în albastru menținând celelalte caracteristici.

`retract` retrage faptul din memoria de lucru, iar `assert` încarcă faptul precizat în memorie.

Execuția codului Jess

Ciclul de executie tipic pentru un sistem expert (și implicit cel folosit de Jess) constă în următorii pași:

- Potrivește faptele din baza de fapte cu cele din premiza regulilor din baza de reguli; mută regulile care s-au potrivit în agenda (activarea regulilor)
- Ordonează regulile din agenda potrivit unei strategii de rezolvare a conflictelor
- Executa partea dreapta a regulilor din agenda în ordinea stabilită la punctul 2

De fiecare data când este introdusă comanda `run` la consolă, Jess realizeaza acești pași, potrivind fiecare regula cu fiecare fapt din memorie, până când nu mai sunt combinații de reguli și fapte care pot fi testate pentru potrivire. Implicit, Jess foloseste o politică “primul potrivit, primul executat” (first matched, first executed) pentru a ordona agenda. Exista totusi modalitati prin care programatorul poate influenta rezolvarea conflictelor.

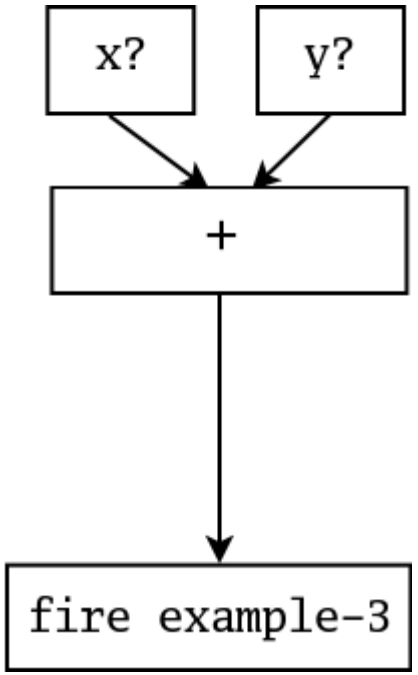
Rezolvarea conflictelor

Fiecare regulă în Jess are o prioritate, care indică importanța regulii în raport cu altele asemănătoare ei. Regulile cu o prioritate mai mare sunt declansate înaintea celor cu prioritate mai mică. În mod default, toate regulile în Jess au prioritate 0.

Programatorul poate stabili manual prioritatea unei reguli ca optiune a functiei `defrule`. Totusi, aceasta nu e paradigma **programării declarative** – programatorul mentioneaza în ce ordine să se execute instrucțiunile - deoarece se elimină astfel unul dintre avantajele programarii declarative, și anume intervenția programatorul în aplicarea regulilor de raționament. Dacă un program necesita un astfel de control asupra ordinei executiei, poate un **limbaj imperativ** ar fi mai potrivit.

Algoritmul Rete. Inferența

Prin aplicarea algoritmului Rete, Jess creaza arbori de decizie care reprezintă regulile sistemului la un nivel înalt. Pentru fiecare arbore de decizie un nod corespunde unui sub-scop din membrul stâng al unei reguli. Fiecare astfel de nod are un set de fapte care valideaza sub-scopul reprezentat în nod. Un drum complet în acest arbore trebuie să lege fiecare sub-scop al unei reguli de concluzia corespunzătoare acesteia.



De exemplu, pentru regula:

```
(defrule example-3 (x) (y) => )
```

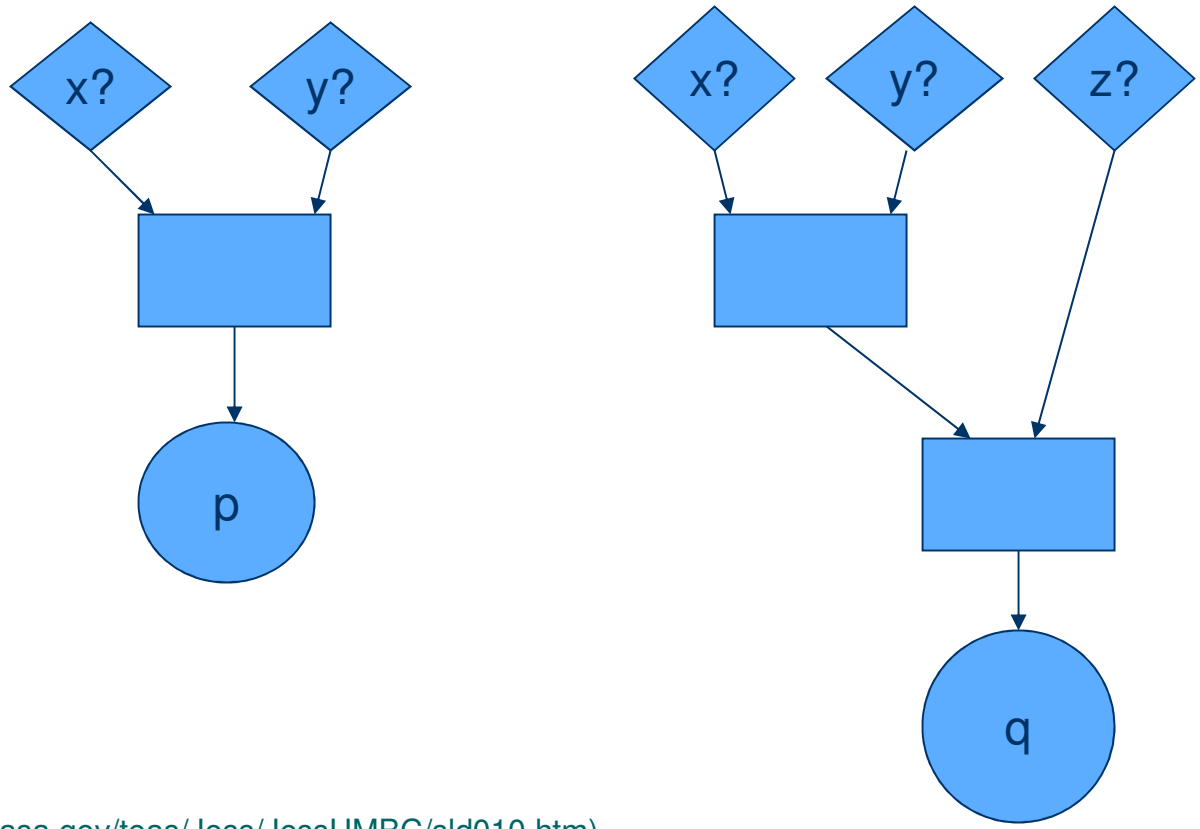
Se construieste următoarea rețea

Rules: IF x & y THEN p
IF x & y & z THEN q

Rete example

Pattern Network

Join Network



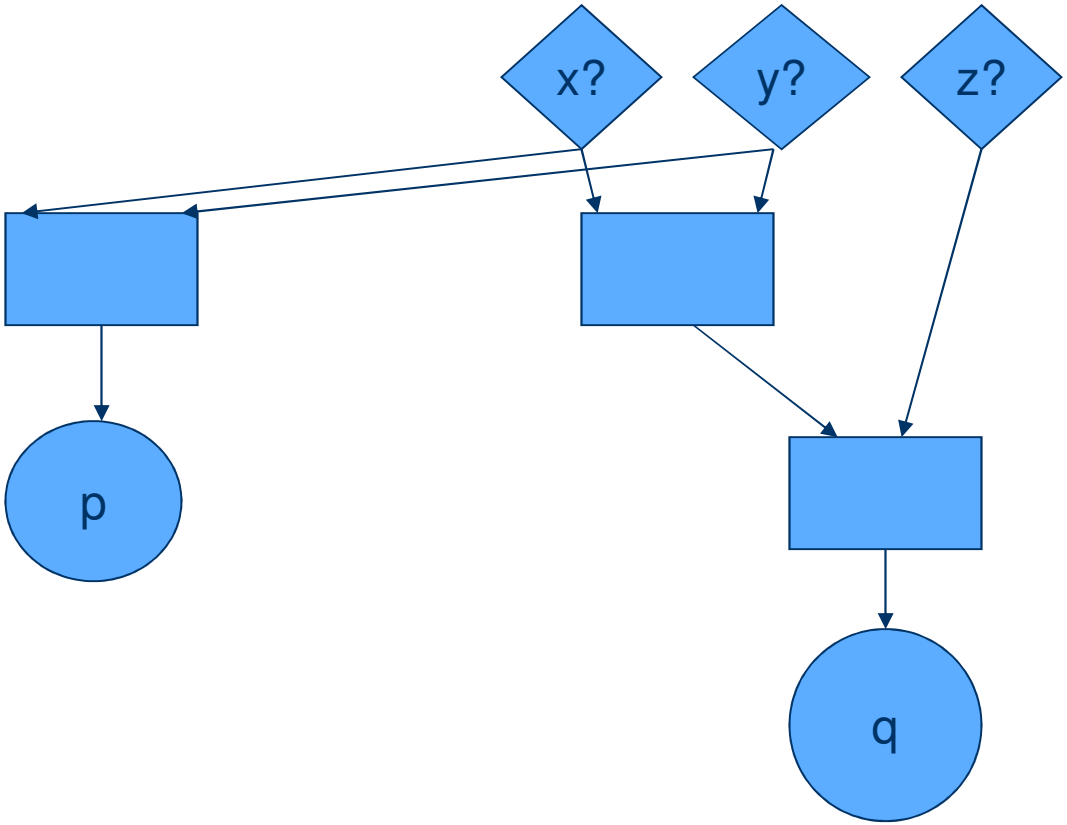
(http://aaaprod.gsfc.nasa.gov/teas/Jess/JessUMBC/sld010.htm)

Rules: IF x & y THEN p
IF x & y & z THEN q

Rete example

Pattern Network

Join Network



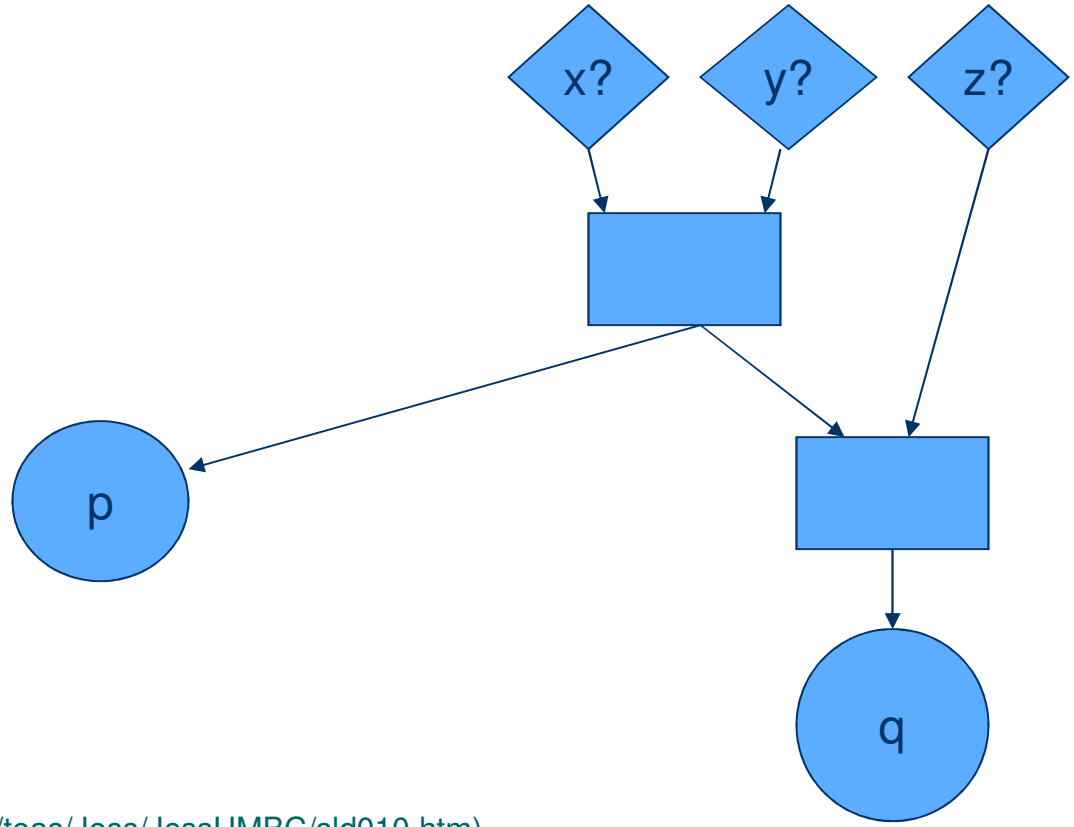
(http://aaaprod.gsfc.nasa.gov/teas/Jess/JessUMBC/sld010.htm)

Rules: IF x & y THEN p
IF x & y & z THEN q

Rete example

Pattern Network

Join Network



(http://aaaprod.gsfc.nasa.gov/teas/Jess/JessUMBC/sld010.htm)

Organizare pe module a regulilor

Jess furnizează o metodă mai puțin intrusivă care să influențeze ordinea execuției programului: ne permite să împărțim baza de reguli și cea de fapte în **module**. Astfel putem grupa împreună reguli și fapte care acționează împreună asupra aceleiași porțiuni din executia programului. Dar lăsăm propriul motorul de reguli să decidă prin rezolvarea conflictelor care regulă din acest modul trebuie declanșată prima.

Implicit, toate regulile și faptele Jess se găsesc în modulul MAIN. Pentru a defini un nou modul, dispunem de funcția `defmodule` cu sintaxa :

```
(defmodule nume-modul)
```

Summary

- Jess vs Lisp
- Arhitectura Jess
- Algoritmul Rete. Caracteristici
- Posibile arhitecturi pentru aplicații Jess+Java
- Reprezentarea cunoștințelor în Jess
- Programarea logică, declarativă, imperativă și funcțională
- Cod Java în Jess
- Cod Prolog în Jess
- Definirea faptelor și a regulilor în Jess
- Rezolvarea conflictelor
- Executia codului Jess. Comenzi de bază
- Organizarea regulilor în module