

Systemes de production

Module MIA – Méthodes pour l' IA
CLIPS

Exemple de moteur d'inférence: **CLIPS**

1ère version 1986

Software Technology Branch (NASA)

- 1. Règles de production (algorithme RETE)**
- 2. Fonctions génériques**
- 3. Programmation « orientée objets »**

**Logiciel gratuit, fonctionne sous PC windows, mac,
Linux...**

syntaxe

- **Syntaxe règles:**

```
(defrule nom_règle « commentaires optionnel »  
  (motif_1)  
  (motif_2)  
  ...  
=>  
  (action_1)  
  (action_2)  
  ...)
```

- **Initialisation base de faits:**

```
(deffacts nomBDF fait1 fait2 fait3 ... faitn)
```

- **Variables:** ?<nom de variable>

- **Motif: condition**

- **Action: introduction de fonctions**

```
(assert x y ...), (retract x y ...), (printout t <chaine>) ...
```

Faits et commandes de l'interpréteur

- **Fait**

(`<nom_prédicat> <param1> <param2> <param3> ...`)

Ex. (`situation_financiere Jean Bonne`)

- **Commandes principales de l'interpréteur**

Insérer des faits dans la base : (`assert fait1 fait2 ... faitN`)

Retirer des faits de la base :

(`retract <référenceFait1> <référenceFait2> ...`
`<référenceFaitN>`)

Effacer la base de faits et la base de règles : (`clear`)

Effacer la base de faits et insertion de (`initial-fact`) : (`reset`)

Lancer l'exécution de l'algorithme RETE : (`run`)

Regarder pas à pas l'insertion et la suppression des faits : (`watch facts`)

Regarder pas à pas l'activation des règles : (`watch rules`)

Désactiver la commande watch : (`unwatch facts`), (`unwatch rules`), etc.

Condition

- **Motif**

1. (`<nom_prédicat> param1 param2 param3 ...>`)

2. **Test**

3. **Negation**

4. **Affectation**

N.B. les paramètres peuvent être des structures et contenir des variables

- **Test: prédicats (`and`, `or`, `eq`) et évaluateur (`=`)**

`(test (or ...))`, `(test (and ...))`, `(test (eq ...))`,
`(test (eq ?x (= 4 ?y)))`

- **Négation: `not motif`**

- **Affectation:**

`?f <- (<nom_prédicat> param1 param2 param3 ...>)`

La variable `?f` reçoit l'adresse du fait indiqué

Quelques facilités

- **Structures de contrôles**

```
(if <condition> then <action1> <action2> ...  
                    else <action1'> <action2'>  
...)
```

```
(while <condition> <action1> <action2> ...  
      <actionN> )
```

- **Initialisation de la base de faits**

1. Utiliser (deffacts <f1> <f2> <f3> ...)

2. Initialiser avec (reset)

cela insère (initial_fact) dans la base de faits; il
reste à ajouter une règle du type

```
(defrule « règle d'initialisation »  
  (initial_fact)
```

Autre fonctions

- **Donne l'ensemble des règles**

`(list-defrules)`

- **Liste l'ensemble des faits**

`(facts)`

- **Imprime une règle**

`(ppdefrule <nom de règle>)`

- **Lier une variable à une valeur**

`(bind ?<nom_var> <valeur>)`

Motif et mémoire de travail

- **Un fait appartient à la mémoire de travail**
(crise-financiere),
(valeur-refuge immobilier)
- **Un fait au moins de la mémoire de travail « filtre » vers le motif:**

Motif: (situation-financière ?i ?x)

Fait: (situation-financière Jean bonne), (situation-financière Martin mauvaise)...

Exemples 1

I
P
6

C
N
R
S

Logique des propositions

(defrule crise

(crise-financiere)

=>

(assert (immobilier-valeur-refuge)))

(defrule richesse

(immobilier-valeur-refuge)

(possede-appartement)

=>

(assert (est-riche)))

(defrule initialisation

(initial-fact)

=>

(assert (crise-financiere)

(possede-appartement)))

Exemples 2

I
P
6
C
N
R
S

Logique des propositions étendue: <attribut valeur>

```
(defrule crise
  (crise-financiere oui)
  =>
  (assert (valeur-refuge immobilier)))

(defrule richesse
  (valeur-refuge immobilier)
  (possede appartement)

  =>
  (assert (situation-financiere riche)))

(defrule initialisation
  (initial-fact)
  =>
  (assert (crise-financiere oui)
          (possede appartement)))
```

Exemples 3

I
P
6

C
N
R
S

Logique des prédicats du premier ordre

```
(defrule crise
  (crise-financiere oui)
  =>
  (assert (valeur-refuge immobilier)))

(defrule richesse
  (valeur-refuge immobilier)
  (possede ?i appartement)

  =>
  (assert (situation-financiere ?i riche)))

(defrule initialisation
  (initial-fact)

  =>
  (assert (crise-financiere oui)
    (possede Jean appartement)))
```

Programmation en CLIPS


Mécanique céleste

Comparaison des conceptions

- Aristotéliennes
- Coperniciennes
- Galiléennes

Aristote 1

```
(defrule initialisation
  (initial-fact)
=>
  (assert
    (planete Mercure)
    (planete Mars)
    (planete Venus)
    (planete Jupiter)
    (planete Saturne)
    (astre Soleil)
    (astre Lune)
    (immobile Terre)))
```

```
(defrule mouvement_astre
  (astre ?x)
=>
  (assert (mouvement_propre ?x
                             circulaire
                             uniforme
                             naturel)))
```

Aristote 2

mouvement

```
(defrule mouvement_planete
  (planete ?x)
=>
  (assert (mouvement_propre ?x
                             circulaire
                             non_uniforme
                             naturel)))
```

Aristote 3

observables

```
(defrule impression_observable
  (mouvement_propre ?x ?c ?u ?n)
=>
  (printout t
    "observable: le mouvement propre de "
    ?x " est "
    ?c ", "
    ?u " et «
    ?n
    crlf) )
```

```
(defrule corps_celestes  
  (or (planete ?x)  
       (lune ?x)  
       (astre ?x))
```

=>

```
(assert (corps_celeste ?x)))
```

Copernic 1

```
(defrule initialisation  
  (initial-fact)
```

=>

```
(assert
```

```
  (planete Mercure)
```

```
  (planete Mars)
```

```
  (planete Venus)
```

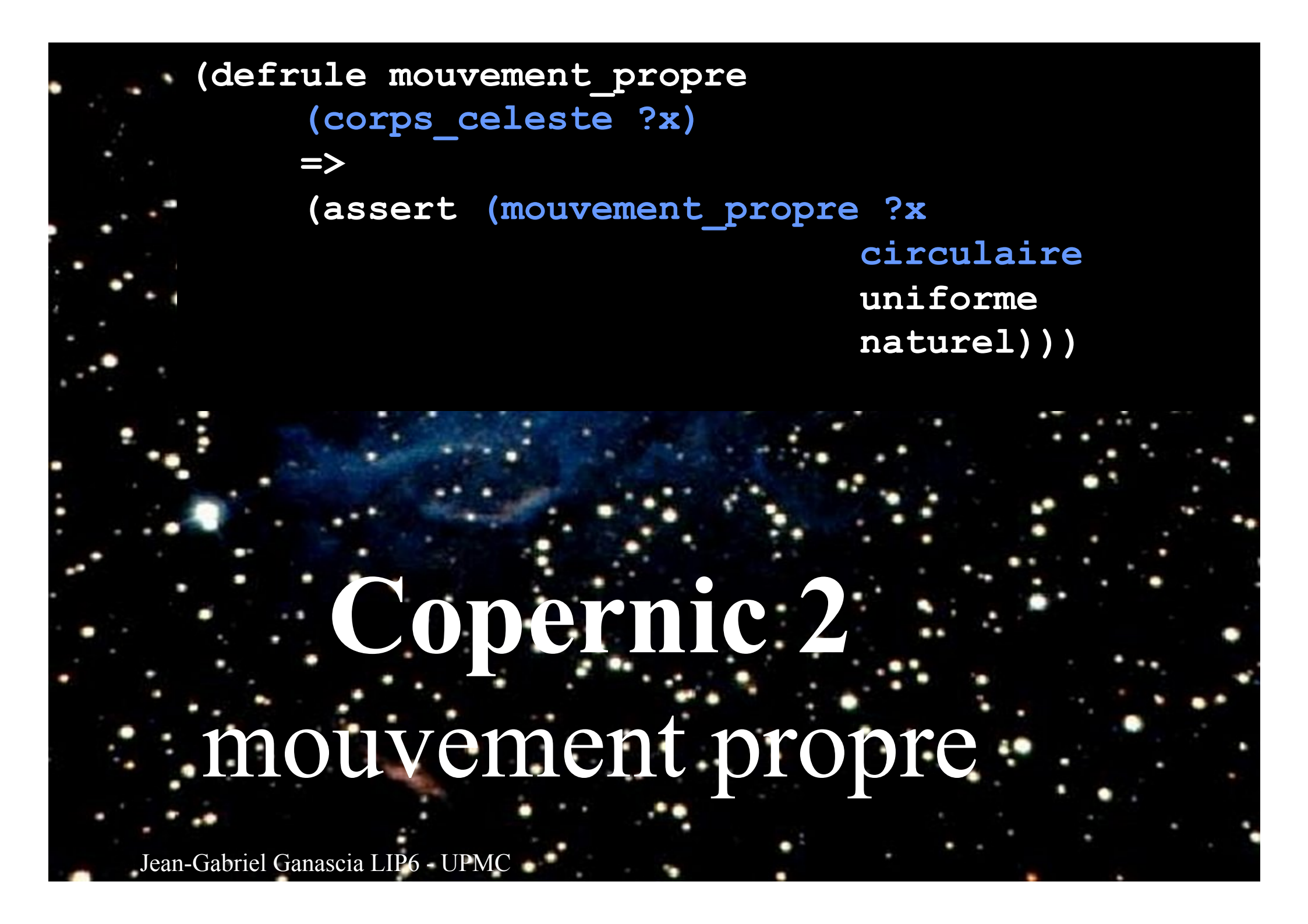
```
  (planete Jupiter)
```

```
  (planete Saturne)
```

```
  (astre Soleil)
```

```
  (lune Lune)
```

```
  (planete Terre)))
```

```
(defrule mouvement_propre
  (corps_celeste ?x)
  =>
  (assert (mouvement_propre ?x
                                circulaire
                                uniforme
                                naturel)))
```

Copernic 2

mouvement propre

```
(defrule centre_planete  
  (or (planete ?x) (astre ?x))  
  =>  
  (assert (centre_mouvement ?x Soleil)))
```

Copernic 2

centre

```
(defrule centre_lune  
  (lune ?x)  
  =>  
  (assert (centre_mouvement ?x Terre)))
```

```
(defrule immobilite_propre  
  (centre_mouvement ?x ?x)  
  =>  
  (assert (immobile_propre ?x)))
```

```
(defrule mouvement_apparent_simple
  (mouvement_propre ?x ?c ?u ?n)
  (centre_mouvement ?x Terre)
=>
  (assert (mouvement_apparent ?x ?c ?u ?n)))
```

```
(defrule mouvement_apparent_simple_bis
  (immobile_propre ?x)
  (centre_mouvement Terre ?x)
  (mouvement_propre Terre circulaire
    uniforme naturel)
=>
  (assert (mouvement_apparent ?x circulaire
    uniforme naturel)))
```

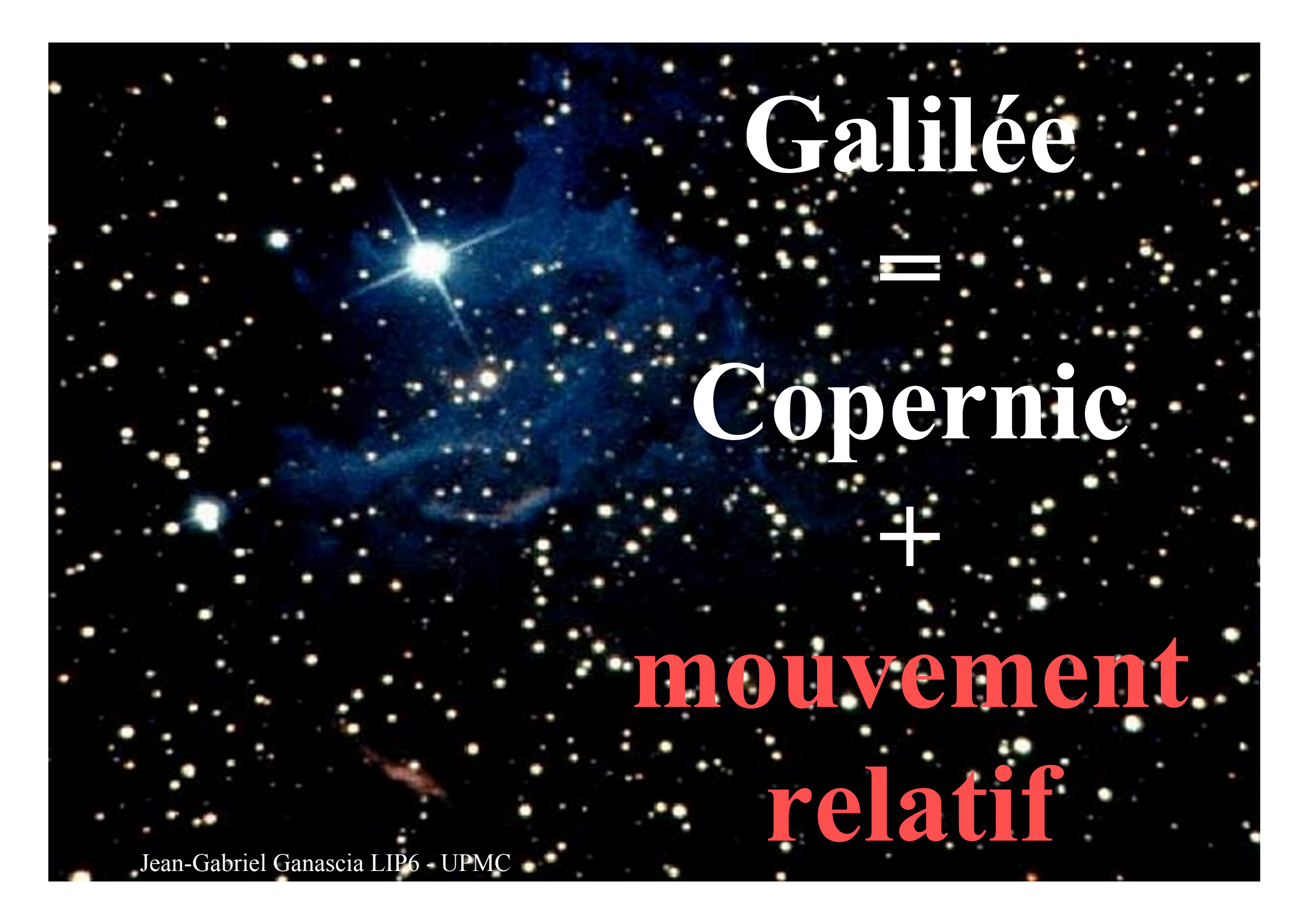
Copernic 2

mouvement apparent simple

```
(defrule mouvement_apparent_composite
  (mouvement_propre ?x circulaire
    uniforme naturel)
  (mouvement_propre Terre circulaire
    uniforme naturel)
  (centre_mouvement ?x ?S)
  (centre_mouvement Terre ?S)
  (test (neq ?x Terre))
  (test (neq ?x ?S))
  (test (neq ?S Terre))
=>
  (assert (mouvement_apparent ?x circulaire
    non_uniforme naturel)))
```

Copernic 2

mouvement apparent composite



Galilée
=
Copernic
+
**mouvement
relatif**

```
(defrule mouvement_apparent_simple
  (mouvement_propre ?x ?c ?u ?n)
  (centre_mouvement ?x Terre)
=>
  (assert (mouvement_apparent ?x ?c ?u ?n)))
```

```
(defrule mouvement_apparent_simple_bis
  (immobile_propre ?x)
  (centre_mouvement Terre ?x)
  (mouvement_propre Terre circulaire
    uniforme naturel)
=>
  (assert (mouvement_apparent ?x circulaire
    uniforme naturel)))
```

Copernic 2

mouvement apparent simple

```
(defrule mouvement_relatif_simple
  (mouvement_propre ?x ?c ?u ?n)
  (centre_mouvement ?x ?S) (test (neq ?x ?S))
=>
  (assert (mouvement_relatif ?x ?S ?c ?u ?n)))
```

```
(defrule mouvement_relatif_simple_bis
  (immobile_propre ?x)
  (centre_mouvement ?s ?x)
  (mouvement_propre ?s circulaire
                        uniforme naturel)
  (not (immobile_propre ?s))
=>
  (assert (mouvement_relatif ?x ?s circulaire
                              uniforme naturel)))
```

Galilée 2

mouvement relatif simple

```
(defrule mouvement_apparent_composite
  (mouvement_propre ?x circulaire
                     uniforme naturel)
  (mouvement_propre Terre circulaire
                     uniforme naturel)
  (centre_mouvement ?x ?S)
  (centre_mouvement Terre ?S)
  (test (neq ?x Terre))
  (test (neq ?x ?S))
  (test (neq ?S Terre))

=>
  (assert (mouvement_apparent ?x circulaire
                              non_uniforme naturel)))
```

Copernic 2

mouvement_apparent_composite


```
(defrule mouvement_relatif_composite
  (mouvement_propre ?x circulaire
    uniforme naturel)
  (mouvement_propre ?y circulaire
    uniforme naturel)
  (centre_mouvement ?x ?S)
  (centre_mouvement ?y ?S)
  (test (neq ?x ?y))
  (test (neq ?x ?S))
  (not (immobile_propre ?x))
  (not (immoblie_propre ?y))
=>
  (assert (mouvement_relatif ?x ?y circulaire
    non_uniforme naturel)))
```

Galilée 2

mouvement apparent composite



```
(defrule mouvement_apparent
  (mouvement_relatif ?x Terre ?c ?u ?n)
=>
  (assert (mouvement_apparent ?x ?c ?u ?n)))
```

Galilée 2

mouvement apparent

```
(defrule frere_ou_soeur
  (parent ?x ?y)
  (parent ?z ?y)
  (test (neq ?x ?z))
  =>
  (assert (frere_ou_soeur ?x ?z)))
```

```
(defrule oncle_ou_tante
  (parent ?x ?y)
  (frere_ou_soeur ?y ?z)
  =>
  (assert (oncle_ou_tante ?x ?z)))
```

```
(defrule cousin
  (oncle_ou_tante ?x ?y)
  (parent ?z ?y)
  =>
  (assert (cousin ?x ?z)))
```

Exemple 4: présence d'un conflit

```
(defrule initialisation
  (initial-fact)
  =>
  (assert (parent Joseph Sarah)
          (parent Sarah Frances)
          (parent Mabel Frances)
          (parent Hebe Mabel)))
```

Modification de la stratégie

Choix de l'instance à déclencher dans l'« agenda des tâches »

- **Stratégie par défaut: « profondeur »**
 - (get-strategy)
- **Modification de la stratégie:**
 - (set-strategy ...) : depth, breadth, simplicity, complexity, random, lex, mea

Stratégies et priorités

- **Stratégies: choix de l'instance déclenchée**
 - Modification de la stratégie: (set-strategy <stratégie>)
 - Connaissance de la stratégie: (get-strategy)
 - Stratégies: depth, breadth, simplicity, complexity, random, lex, mea
- **Priorités: affectation d'un ordre sur les règles**
 - Définition de la « salience » des règles
 - Salience: nombre compris en -10000 et +10000, par défaut 0
 - Modification dynamique de salience

Stratégies

Changement de stratégie

- **En profondeur:** agenda = pile
- **En largeur:** agenda = file
- **Aléatoire**
- **Simplicité:** choix règle en fonction simplicité prémisse
- **Complexité:** choix règle en fonction complexité prémisse
- **Lex:** stratégie de récence (*l'instance de règle qui couvre le fait le plus récent d'abord*)
- **Mea** stratégie de récence (*en tenant compte de l'ordre des prémisses*)

```
(defrule frere_ou_soeur
  (parent ?x ?y)
  (parent ?z ?y)
  (test (neq ?x ?z))
  =>
  (assert (frere_ou_soeur ?x ?z)))

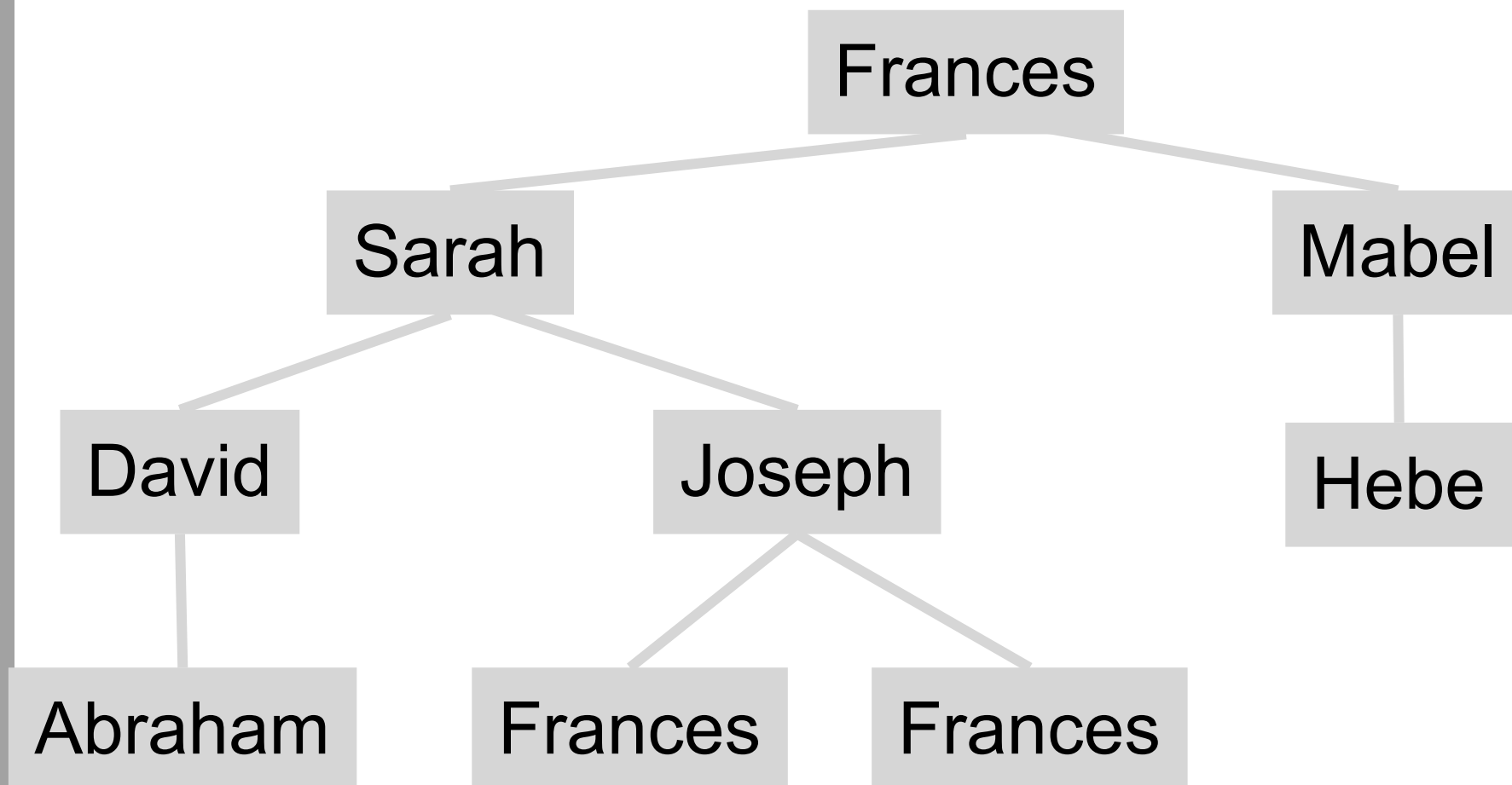
(defrule oncle_ou_tante
  (parent ?x ?y)
  (frere_ou_soeur ?y ?z)
  =>
  (assert (oncle_ou_tante ?x ?z)))

(defrule cousin
  (oncle_ou_tante ?x ?y)
  (parent ?z ?y)
  =>
  (assert (cousin ?x ?z)))
```

Mini exemple 5: présence d'un conflit

```
(defrule initialisation
  (initial-fact)
  =>
  (assert (parent Joseph Sarah)
          (parent Sarah Frances)
          (parent Mabel Frances)
          (parent Hebe Mabel)
          (parent Abraham David)
          (parent Nathanael Joseph)
          (parent Raphael Joseph)))
```

Arbre généalogique



I Un modèle psychologique

I

P

6

C

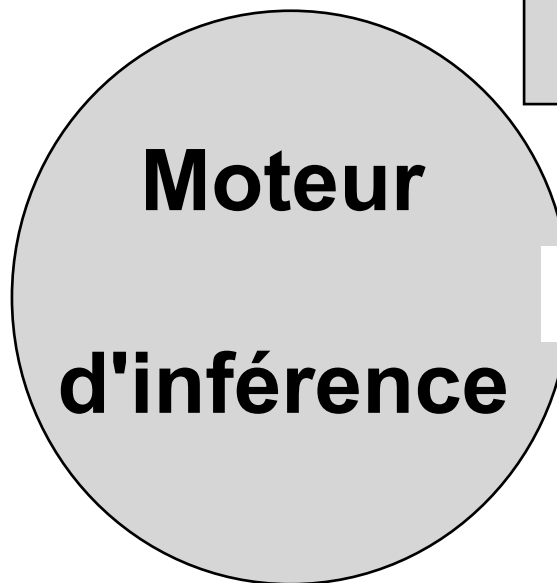
N

R

S

Mémoire à long terme

Base de connaissances:
règles de production



Mécanisme de raisonnement

Etat initial

Etat courant

Base de faits

But

Mémoire à court terme