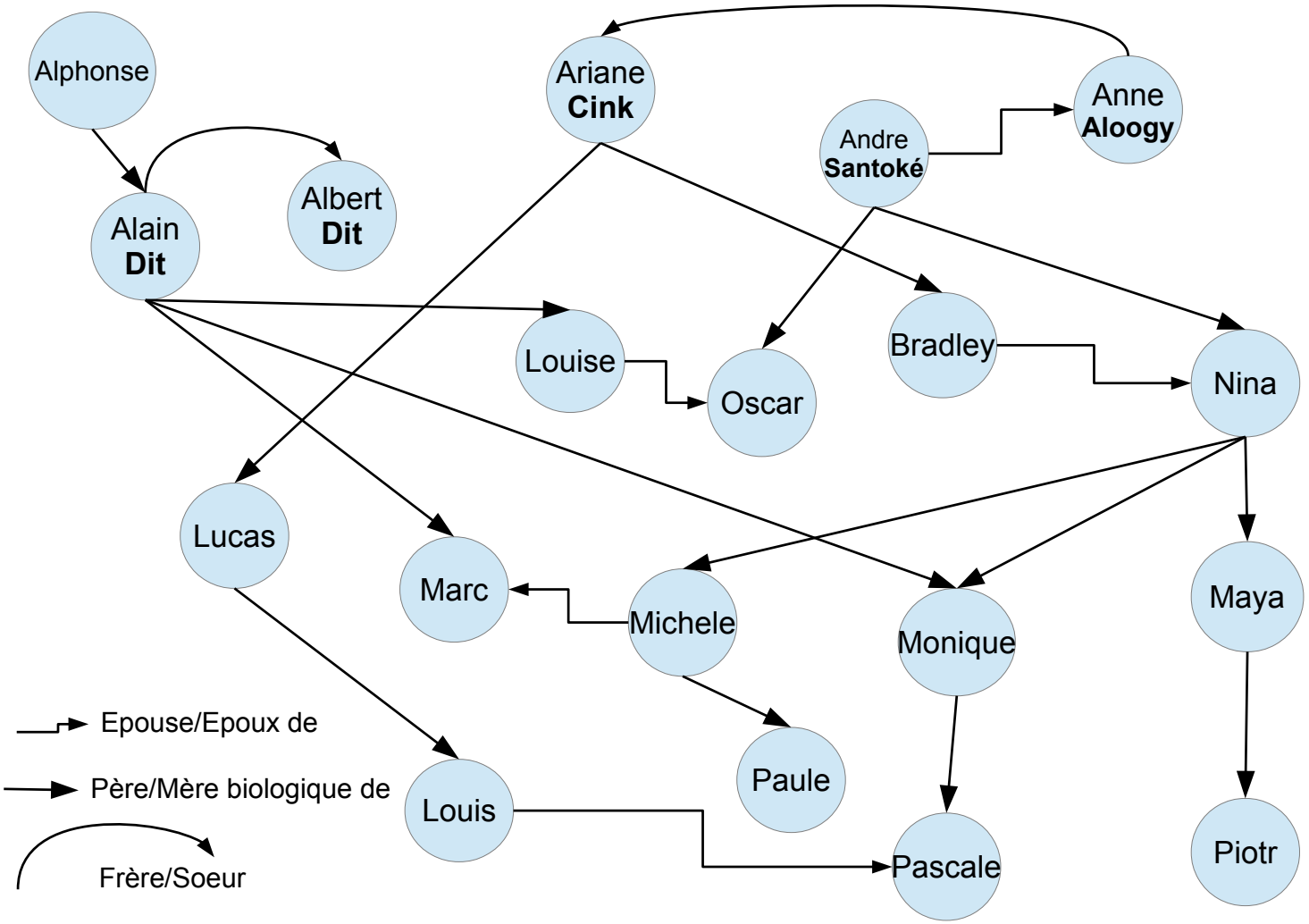
Université de Strasbourg

TDTP Systèmes experts – CLIPS

Pierre Gançarski
Université de Strasbourg

Graphe de parenté



Initialisation

- Q1 : Donner plusieurs types de faits possibles sur les nœuds et entre les nœuds du graphe
  - Exemple :
    - pereDe : (pereDe Alain Monique)
    - …

Initialisation

- Q1 : Donner plusieurs types de faits possibles sur les nœuds et entre les nœuds du graphe
  - Exemple :
    - pereDe , mereDe
    - nom
    - famille
    - APourFrere , APourSoeur
    - epouxDe, epouseDe

Initialisation

- Q2 : Donner un ensemble de faits (le plus petit possible) permettant de représenter ce graphe
  - Exemple :
    - (mereDe Maya Piotr)
    - …

Initialisation

- Q2 : Donner l'ensemble de faits initiaux permettant de représenter ce graphe
  - Exemple :
    - (mereDe Maya Piotr)
    - …

→ Voir fichier Corrigé Q1 (LIENS.clp)

Règles

```
(defrule <id_regle>
  (operateur
          <condition1>
             ...
         <conditionN>
   )
=>
   (action 1)
    <(action2>
    ...
   <(actionP)>
)
```

```
(defrule R1
 (and
      (pereDe ?x ?y)
      (pereDe ?x ?z)
      (genreDe ?z masculin)
  )
=>
 (assert (frereDe ?z ?y))
)
```

Attention au parenthèsage !!!
assert : ajoute le fait à la base de fait
S'il n'y a qu'un *and*, on peut l'omettre : on met juste la liste des conditions
Il ne peut pas y avoir de *test* ou de condition dans les actions , ni de *OU*

Question 2

- Q2 : Ecrire la ou les règles :
  - Q2.1 : qui crée(nt) pour chaque fait (epouxDe X Y) le fait "réciproque" (epouse Y X).

```
( defrule R2.1_EpouseDe
      (epouxDe ..... .....)
 =>
     (assert (epouseDe .... ....))
 )
```

Question 2

- Q2 : Ecrire la ou les règles :
  - Q2.1 : qui crée(nt) pour chaque fait (epouxDe X Y) le fait "réciproque" (epouse Y X).

```
( defrule R2.1_EpouseDe
      (epouxDe ..... .....)
 =>
     (assert (epouseDe .... ....))
 )
```

Question 2

- Q2 : Ecrire *la* règle :
  - Q2.3 : qui crée pour chaque nom P apparaissant dans le graphe, le fait (nom P)

```
( defrule R2.3_creerNom
   (or

   )

   =>
        (assert (nom ....))
   )
```

Question 2

- Q2 : Ecrire *la* règle :
  - Q2.3 : qui crée pour chaque nom P apparaissant dans le graphe, le fait (nom P)

```
( defrule R2.3_creerNom
   (or

   )

   =>
        (assert (nom ....))
   )
```

→ Fichier TP_corrige_Q2.1_2.3.clp

Question 2

- Q2 : Ecrire *la* règle :
  - Q2.4 : qui crée(nt), en utilisant la liste des noms, pour chaque personne P un fait (genre P Feminin) ou (genre P Masculin)

```
( defrule R2.3 _GenreMasculin (nom ?n)
      (not (genre ?n ....)
      ( listeNomsMasculins  . . . . )
=>
   (assert (genre .... ))
)
```

Question 2

- Q2 : Ecrire *la* règle :
  - Q2.4 : qui crée(nt), en utilisant la liste des noms, pour chaque personne P un fait (genre P Feminin) ou (genre P Masculin)

Parcours d'une liste : $?

(listeNomsMasculins $? ?c $?) : parcourt toute la liste en affectant

- le début de la liste à une variable « non stockée »
- un élément dans la variable ?c
- le début de la liste à une variable « non stockée »

Exemple :

Comment prendre deux éléments consécutifs de L : .....

Comment récupérer une liste L sans son premier élément : .....

Question 2

- Q2 : Ecrire *la* règle :
  - Q2.4 : qui crée(nt), en utilisant la liste des noms, pour chaque personne P un fait (genre P Feminin) ou (genre P Masculin)

Modification d'une liste : on utilise *retract* puis *assert*

Comment amputer une liste *Maliste* de son premier élément

```
(defrule R
    ?f1 <- (Maliste   ?x $   $?fin)
=>
   (retract ?f1)
   (assert (Maliste  $?fin)
)
```

Question 2

- Q2 : Ecrire *la* règle :
  - Q2.4 : qui crée(nt), en utilisant la liste des noms, pour chaque personne P un fait (genre P Feminin) ou (genre P Masculin)

```
( defrule R2.3 _GenreMasculin (nom ?n)
      (not (genre ?n ....)
      ( listeNomsMasculins  . . . . )
=>
   (assert (genre .... ))
)
```

→ Fichier TP_corrige_Q2.4_2.2.clp

Fait structuré

- Fait structuré → Template

```
(deftemplate membre
        (slot nom (type STRING)  (default ''NoName '')) ;; Personne
        (slot genre (default nil) (allowed-values Masculin Feminin));; Son genre
        (slot nomFamille (default nil))        ;; Son nom de famille
        (multislot pereDe)                     ;; Liste de ses enfants
        (multislot mereDe)                     ;; Liste de ses enfants
        (slot epouxDe (default nil))           ;; Nom de son épouse
        (slot epouseDe (default nil))          ;; Nom de son époux
        (multislot APourFrere)     ;; Liste des personnes dont elle est frère
        (multislot APourSoeur)     ;; Liste des personnes dont elle est soeur
)
```

Fait structuré

- Utilisation

```
(defrule R3.2_GenererMembre
  ?f <- (nom ?n)
=>
  (assert (membre (nom ?n) ))   )
```

- Modification

```
(defrule R3.3_CompleterNomFamille
  ?f <- (membre (nom ?n) (nomFamille nil))
=>
  (modify ?f (nomFamille ?fam))
```

- Q3 : Compléter ces deux règles

Fait structuré