Calculs Locaux
Exemples de calculs d'arbres recouvrants

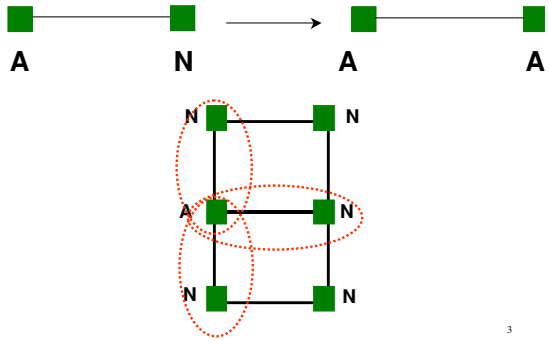
1

Introduction

- Etat d'un process codé par une étiquette
- (selon son état, les états de ses voisins) une étape de calcul est effectué
- Changement d'état: « réétiquetage » ou « relabeling »
- Utilisation des règles de réétiqutage
- Buts :
  - Abstraction
  - simplification
  - facilité de preuves
  - indépendance de tout modèle
  - implémentation unifiée

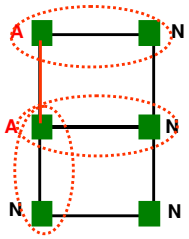
2

Exemple



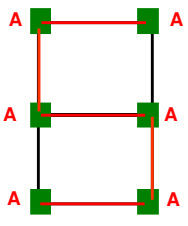
3

Etape de calcul



4

un arbre recouvrant



5

Codage d'algorithmes distribués

- Un sommet étiqueté A (Actif), et tous les autres sont étiquetés N (neutre)
- Règle:
  - Si un sommet x étiqueté A possède un voisin y étiqueté N
  - Alors x est étiqueté A et l'arête (x,y) est marquée
- Résultats : arbre recouvrant
- plusieurs règles peuvent être appliquées en //

6

Preuves :

- Terminaison : noethérien
- Correction : invariants

- Cet algorithme est distribué ne *détecte pas localement la terminaison globale*.
- D'autres systèmes de réécriture permettent de capturer cette propriété.

7

Définitions : systèmes de réétiquetage de graphes

- Un système de réétiquetage (GRS) est un triplet R=(L,I,P) où L est un alphabet, I sous ensemble de L, P est un ensemble de règles de réétiquetage. Chaque règle est de la forme (Gr, λ_r) (Gr, λ'_r), notée aussi (Gr,λ_r,λ'_r)
- Une étape de réétiquetage est une application d'une règle
- un graphe est (G,λ) est R-irréductible si aucune règle de R n'est applicable
- Système de réétiquetage avec priorités

R=(L,I,P,>) est un système de réétiquetage avec priorités si (L,I,P) est un SR et > est un ordre partiel défini sur l'ensemble P (relation de priorité). Une règle prioritaire s'applique en 1er en cas de conflit.

8

- Contexte interdit: Un contexte d'un graphe (G,λ) est un sous graphe de (G,λ) à isomorphisme près.

Un règle de réétiquetage avec contexte interdit est une ($Gr,\lambda_r,\lambda'_r,F_r$) avec F_r : contexte interdit

Un système de réécriture avec contexte interdit est un système de réécriture ou chaque règle est avec contexte interdit

Une règle est appliquée sur un sous graphe s'il ne contient aucun de ses contextes interdits

9

Application à l'étude des algos distribués

- Ramener l'étude d'un algo distribué à l'étude d'un système de réétiquetage qui le code
- plus simple, facile à comprendre, à prouver etc.

1/ Terminaison : un algorithme distribué termine si le SR n'induit pas de chaîne de réécriture infinie (s'il est noethérien). Pour prouver qu'un système est noethérien, il suffit d'exhiber un ordre noethérien compatible avec le SR: trouver un ensemble partiellement ordonné (S,<) qui n'a pas de chaîne infinie décroissante

et une fonction f : GL S /

(G,λ) (G,λ') alors f(G,λ) > f(G,λ')

2/ Correction : invariants

3/ Complexité : nombre de réécritures

10

- Exemple (Arbre recouvrant avec SR1)
- SR1 = (L1,I1,P1)

Théorème:

1. SR1 est noethérien
2. Soit (G,λ) un graphe étiqueté dans I1, avec un seul sommet étiqueté A, et soit (G,λ') un graphe SR1-irréductible. Le sous-graphe induit par les arêtes étiquetées 1 est un arbre de G.
3. La longueur d'une séquence est au plus n -1 (avec n le nombre de sommets de G)

11

- SR1 est noethérien
  - Preuve

    f : GL1 N

    (G, λ) |VN|

    L'ordre > sur N est compatible avec le système de réécrirture SR1

- Invariants:
  - P1 : Toute arête incidente à un sommet étiqueté N est étiquetée 0
  - P2 : Tout sommet étiqueté A (à part la racine) est incident à au moins une arête étiquetée 1
  - P3 : Le sous graphe induit par les arêtes étiquetées 1 est un arbre

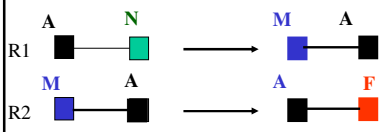
Preuve : par récurrence sur les chaînes de réécriture

12

- Exercice:
  - Que donne ce système si on a plusieurs sommets initialement étiquetés A ?

13

Calcul d'un arbre recouvrant séquentiel avec détection de la teminaison (système SR2)



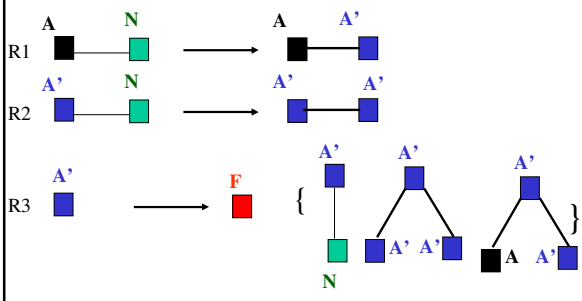
R1 > R2

14

- Théorème : SR2 calcule un arbre recouvrant. Il détecte localement la terminaison globale.
- Preuve:
  - SR2 est noethérien
  - SR2 est correct (le calcul est un arbre recouvrant)
    - P1: Toutes les arêtes incidentes à N sont étiquetées 0
    - P2: Tout sommet étiqueté A, F ou M est incident à au moins une arête étiquetée 1
    - P3: le sous graphe induit par les arêtes étiquetées 1 est un arbre
    - P4: il y a exactement un seul sommet étiqueté A
    - P5: le sous graphe induit par les sommets étiquetés A et M et les arêtes étiquetées 1 est un chemin ayant une extrémité étiquetée A
    - P6: tout sommet étiqueté F n'a aucun voisin étiqueté N

15

Distribué avec terminaison (SR3)



16

- Théorème: le système SR3 calcule un arbre recouvrant avec détection locale de la terminaison globale

Preuve:

- SR3 est noethérien
- le résultat est un arbre recouvrant
- détection locale de la terminaison globale: A entourée par des F

17

- P1: Toutes les arêtes incidentes à N sont étiquetées 0
- P2: tout sommet étiqueté A, A', ou F est incident à au moins une arête étiquetée 1
- P3: Le sous graphe induit par les arêtes étiquetées 1 est un arbre
- P4: Il y a exactement un sommet étiqueté A
- P5: le sous graphe induit par les sommets étiquetés A et A' est les arêtes étiquetées 1 est un arbre
- P6: Tout sommet étiqueté F n'a aucun voisin étiqueté N

18

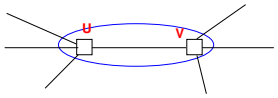
Calculs locaux

- généralisation des SR (réécriture de boules)
- conditions:
  - C1 : ne modifie pas le graphe sous-jacent. Seul l'étiquetage est modifié.
  - C2 : ils sont locaux, chaque étape ne concerne qu'un sous graphe de taille fixe dans le graphe sous-jacent
  - C3 : les réétiquetages sont localement contrôlés (ou engendrés); l'application d'une règle ne dépend que du contexte local du sous graphe

19

Implémentation de calculs locaux.

- Implémentation par des procédures
- Trois types de calculs locaux:
- computations:
  1. Rendez vous : Etiquettes d'un sous graphe k2 sont modifiés selon des règles qui dépendent de ces étiquettes.



20

Implémentation (Procédure Rendezvous).

Chaque sommet exécute les actions suivantes:

un sommet v choisit au hasard un de ses voisins c(v);

v envoie 1 à c(v);

v envoie 0 à tous les sommets différents de c(v);

v reçoit les messages de tous ses voisins.

(Il y a un rendez-vous entre v et c(v) si v reçoit 1 de c(v). Une étape du calcul peut avoir lieu*)*

21

2. Calcul Local 1 (LC1): Pour une boule de rayon 1 (une étoile), l'étiquette du centre de la boule est modifiée selon une règle qui dépend de celle des sommets de la boule.
3. Calcul Local 2 (LC2): Comme pour LC1sauf que tous chaque sommet de la boule modifie son étiquette.

Implémentation de LC1(Election locale probabiliste):

Le sommet v tire au hasard un entier rand(v);

v envoie rand(v) à tous ses voisins;

v reçoit les entiers de ses voisins;

(le sommet v est élu dans B(v,1) si rand(v) est strictement plus grand que tous les entiers reçus par v; dans ce cas un pas de calcul LC1 peut être effectué sur B(v,1) *)*

22

Implémentation de LC2(Election locale probabliste):

Le sommet v tire au hasard un entier rand(v);

v envoie rand(v) à tous ses voisins;

v reçoit les entiers de ses voisins;

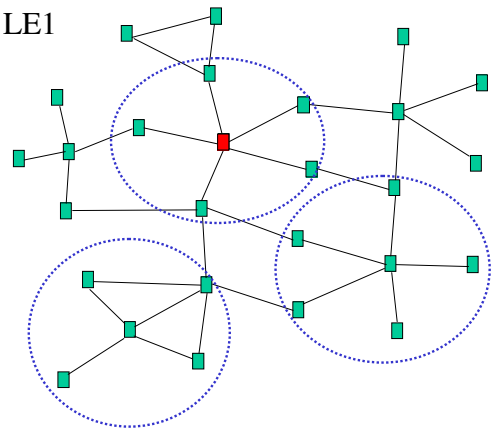
v envoie le max des entiers reçus à chacun de ces voisins;

v reçoit les entiers de ses voisins.

(le sommet v est élu dans B(v,2) si rand(v) est strictement plus grand que tous les entiers reçus par v; dans ce cas un pas de calcul LC2 peut être effectué sur B(v,1) *)*

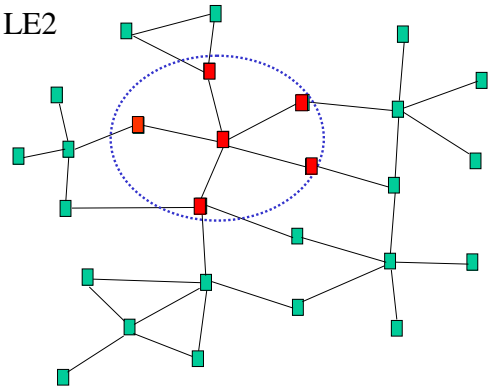
23

LE1



24

LE2



25

Un méta-algorithme

Chaque processeur exécute

While (run){

\\ Synchronisation (rendezvous, LE1, LE2);

\\ Echange d'étiquettes, attributs,

\\ calculs;

\\ Mise à jour des étiquettes, attributs, états;

\\ Arrêt de la Synchronisation;

}

26

Exemple (Calcul d'un arbre recouvrant – Rendez-vous)

```
while (run) {
  neighbour = rendezVous();
 sendTo(neighbour,myLabel);
  neighbourLabel=receiveFrom(neighbour);
  if (myLabel == 'N') && (neighbourLabel == 'A'){
      myLabel = 'A';
      edge[neighbour]=1
      }
  breakSynchro();
}
```

27

Application à la visualisation de l'exécution d'un algorithme distribué

- Implémentation à la visualisation de tous les évènements.
- Bibliothèque de primitives.
- Les propriétés du système de réécriture sont préservées: : consistance, ordre des évènements, terminaison, résultats.

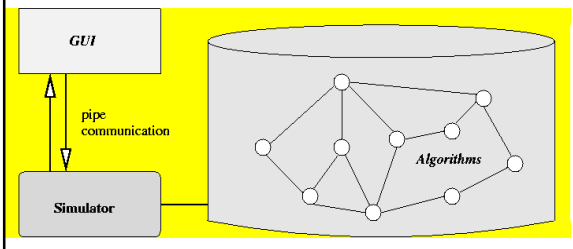
28

ViSiDiA

- Un environnement pour simuler, visualiser et expérimenter les algorithmes distribués.
- Un processeur est implémenté par un thread Java.
- Une bibliothèque de primitives de haut niveau est disponible pour le programmeur.
- Une interface graphique (contrôlable par l'utilisateur) permet à l'utilisateur de
  1. Construire d'éditer un réseau (avec GML export/import).
  2. visualiser l'exécution d'un algorithme ou de l'expérimenter.

29

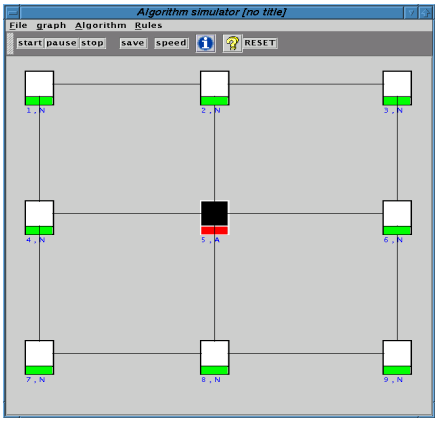
Architecture



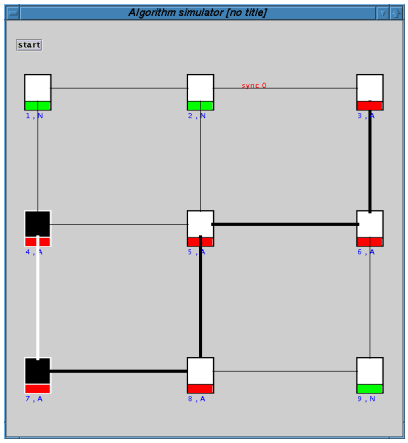
•Modulaire, orienté-objet, portable, interactif, limites (celles de la java)..

30

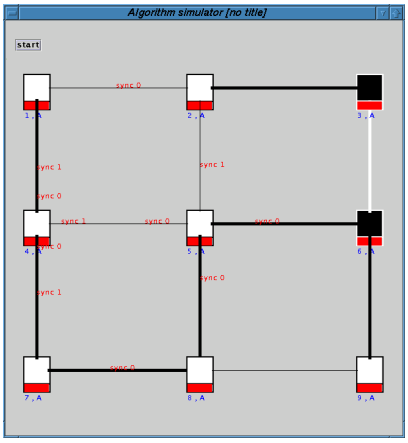
p a n n i g t r e e)



31



32



33

A/ Calculs d'arbres recouvrants (anonyme)

Arbre recouvrant (sans détection locale de la terminaison globale)

1. Implémentation avec LC1

```
while (run) {
   synchro=starSynchro1();
   if( synchro==starCenter) {
        for (int door=0;door<arity;door++) {
          neighbourState[door]=receiveFrom(door);
          if (neighbourState[door]=="A")
            neighbourA=door;    }
      if ((myState=="N") && (neighbourA != Null)) {
          myState="A";
          neighbourLink[neighbourA]=true; }
       breakSynchro(); }
   else
     if (synchro == starLeaf)
        sendTo(center,myState); }
```

34

2. Implémentation avec LC2

```
while (run) {
   synchro=starSynchro2();
   if( synchro==starCenter ){
      for (int door=0;door<arity;door++) {
          neighbourState[door]=receiveFrom(door);
          if ((myState=="A") && (neighbourState[door]=="N")) {
              neighbourLink[door]=true;
              sendTo(door,change);
          }  }
      breakSynchro();
   }  else
     if (synchro == starLeaf) {
        sendTo(center,myState);
        answer=receiveFrom(center);
        if (answer == change)
          myState="A";
   }
```

35

Arbre recouvrant (séquentiel avec détection de la terminaison)

- Système avec priotité.
- Connaissance du contexte pour vérifier les priorités.

  ➡ Implémentation avec LC1 ou LC2

- Principe: (avec LC2)
  - Si l'étiquette du centre est N alors
    - S'il y a un voisin étiqueté A alors appliquer R1
  - Si l'étiquette du centre est M alors
    - S'il y a un voisin étiqueté A, on ne peut pas appliquer R2

    (le sommet étiqueté A peut avoir un voisin N ailleurs)
  - Si l'étiquette du sommet est A alors
    - S'il y a n voisin N, appliquer R1
    - S'il y a un voisin M, appliquer R2
    - Si tous les voisins sont F …c'est fini !!! (démo)/

36

Arbre recouvrant (distribué avec détection de la terminaison)

- Système avec contexte interdit
- Implémentation avec LC1 ou LC2
- Principe:
  - Application des règles R1 et R2
  - Pour la règle R3:
    - Si le centre est étiqueté A', collecter toutes les étiquettes des voisins et vérifier le contexte. Si aucun contexte interdit, appliquer R3.
  - La détection de la terminaison : étiquette du centre est A, et tous les voisins sont étiquetés F.

37

Arbre recouvrant (réseaux avec identités)

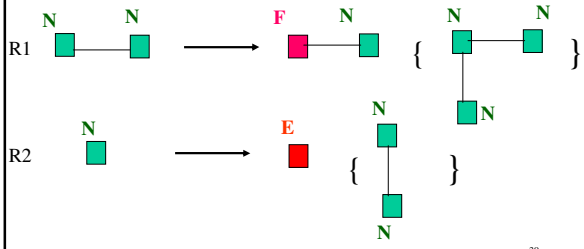
- On considère des graphes avec identités. On suppose que les étiquettes sont initialiées par les identités. Le principe de l'algorithme est de réécrire un sommet par le numéro d'un voisin plus grand.
- Ecrire le système de réécriture

38

B/ Election

- Choisir un processeur unique (qui sera utilisé pour un calcul ultérieur).

B.1/ Election dans un arbre.



39

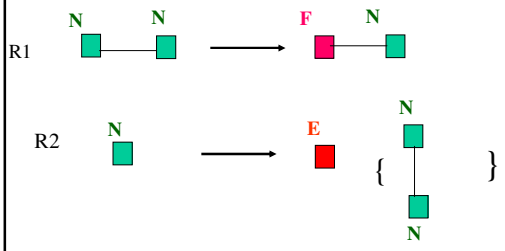
Implémentation de l'élection dans un arbre

```
while(run){
   synchro=starSynchro1();
   if( synchro==starCenter ){
        int n_Count=0;

        for (int door=0;door<arity;door++) {
            neighbourState[door]=receiveFrom(door);
            if (neighbourState[door]=="N")
                n_Count++;
        }
          if ((myState="N") && (n_Count ==1))
            changeState(F);
        else
          if ((myState=="N") && (n_Count==0)) {
            changeState(E);
            run=false;
          }
        breakSynchro();
   }
   else {
       if (synchro == starLeaf) {
          sendTo(center,myState);
       }
   }
}
/* demo */
```

40

B.2/ Election dans un graphe complet



41

Implémentation de l'élection dans un graphe complet

```
while(run){
   neighbour=rendezVous();
   sendTo(neighbour,myState);
   neighbourLabel=receiveFrom(neighbour);
    if ((myState == "N") && (noNeighbourN))
      myState ="E"
    if ((myState == "N") && (neighbourLabel== "N")) {
          myState="F";
   }
}
```

42

C/ Algorithmes de détection de la terminaison

Déf: tous les processeurs ont terminé et aucun message n'est en transit.

Terminaison explicite: Il existe au moins un processeur qui détecte la terminaison globale.

I/ Algorithme de Dijkstra-Scholten (diffusion)

- Maintien d'un arbre dynamique de processeurs actifs.
- Chaque processeur a deux états: passif, actif, et un compteur (du nombre de fils activés).
- Initialement : un processeur actif (racine), tous les autres sont passifs.

43

Suite Dijkstra-Scholten

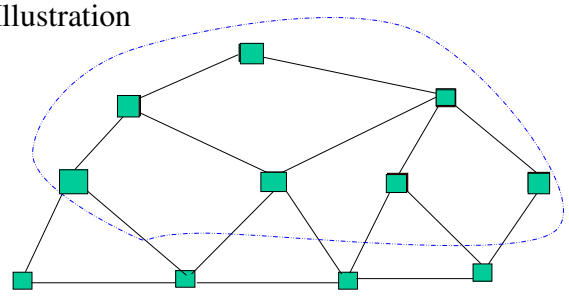
- Principe:
  - Chaque processeur actif peut « activer » ses voisins passifs (qui deviennent ses fils s'ils sont activés pour la première fois)
  - Le passage d'un état actif a passif est spontané.
  - Un processeur passif et qui n'a aucun descendant actif

(feuille de l'arbre) envoie un acquittement à son père dans l'arbre.

- Terminaison: racine est passif et ne possède aucun descendant.

44

Illustration



Terminé !!!!

45

Un système de réécriture décrivant Dijkstra-Scholten

- Chaque sommet : une étiquette (X,Y,sc)

X : marquage {A,A'}; Y : actif ou passif {Ac,Pa}

R1 (X,Ac,sc) — Pa → (X,Ac,sc+1) — Ac — (A',Ac,sc) ; (N,Pa,0) — X∈{A,A'}

R2 (X,Ac,sc) → (X,Pa,sc)

R3 (X,Ac,sc) — Pa → (X,Ac,sc) — Pa — (Y,Ac,sc') ; (Y,Pa,sc') — sc'≠ 0

46

Suite Dijkstra Scholten

R4 (X,E,sc) — Ac → (X,E,sc-1) — Pa — (N,Pa,0) ; (A',Pa,0) — E ∈{Ac,Pa}

R5 (A,Pa,0) → Fin

47

Applications:

1/ Preuve de l'algorithme: Les invariants:

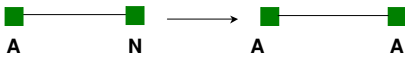
- sc+1 est le nombre d'arêtes étiquetées Ac incidentes au sommet étiqueté (X,Y,sc).
- Toutes les arêtes incidentes à (N,Pa,0) sont étiquetées Pa.
- Le sous graphe induit par les sommets étiquetés (X,Y,sc) est et les arêtes étiquetées Ac est connexe, et ne possède pas de cycle.

La démonstration de ces invariants se fait par récurrence sur les règles de réécriture.

48

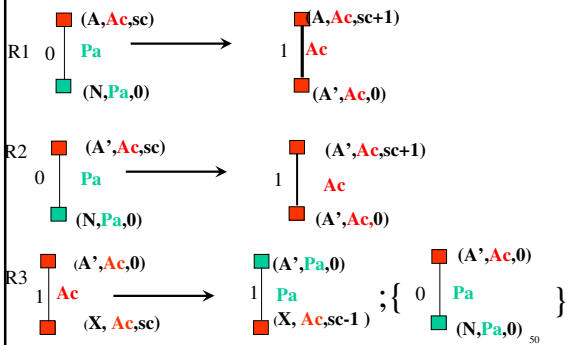
2/ Application à la détection de la terminaison d'un algorithme distribué

- Un système de réécriture décrivant un algorithme distribué ou le calcul est initialisé par un seul sommet.
- Pour le rendre détecter localement la terminaison, il suffit de « superposer » deux systèmes de réécritures.
- Exemple: le calcul d'un arbre recouvrant



49

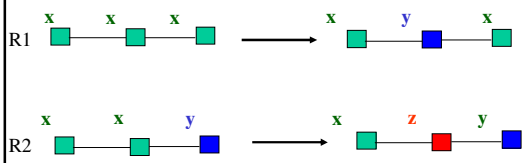
2.1/ Arbre recouvrant avec terminaison de Dijkstra-Scholten



50

2.2/ 3-coloration des sommets auto-stabilisante d'un anneau

- Anneau avec des couleurs sur les sommets (initalement: configuration arbitraire)
- But: 3-coloration correcte (pas de sommets adjacents avec les mêmes couleurs).

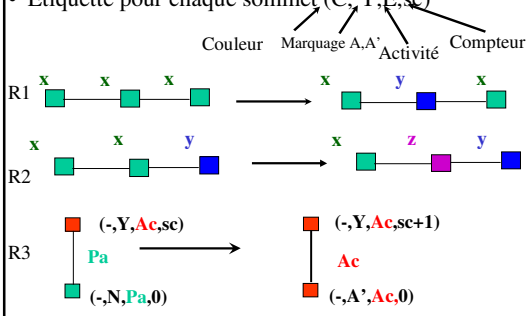


Démo.

51

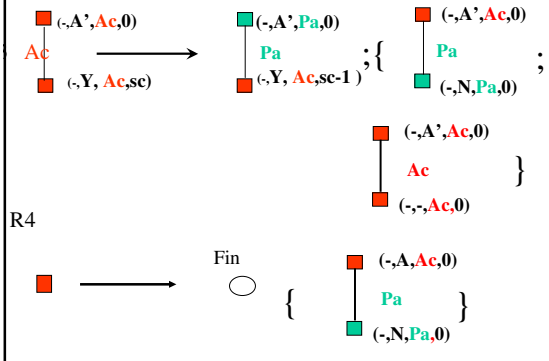
Terminaison par Dijkstra-Scholten

- Étiquette pour chaque sommet (C, Y,E,sc)



52

Suite 3-coloration-Dijkstra-scholten



53

2/ Algorithmes de Dijkstra-Feijen-Van Gasteren.

gr

54

Conclusion

- Approche uniforme pour implémenter les calculs locaux.
- Utilisation: validation, expérimentation, enseignement d'algorithmes distribués (comprendre les concepts)
- Autres outils:

  Lydian: (université de chalmers): outil permettant de simuler des réseaux et implémenter des algorithmes distribués (avec le choix du modèle).

  Daj: (université de Linz) une bibliothèque de fonctions Java pour l'algorithmique distribuée.

55