# De la pratique industrielle à la logique mathématique

*Un exemple de certification de logiciel*

Solange Coupet-Grimal

Laboratoire d'Informatique Fondamentale
de Marseille

# Systèmes critiques

## Vies humaines en jeu – Gros enjeux économiques

- Transports

  RER (Meteor, ligne B), avions, fusées (Ariane)

- Matériel médical

  Laser, chirurgie à distance, traitement par rayons (Thérac 25)

- Banques

  Cartes à puce, porte-monnaie électronique, traitement informatisé des salaires

# Systèmes critiques

- Systèmes complexes

  Parallèles, communicants, exécutions infinies

- Exigence de fiabilité maximale

  Standard international : *Critères Communs (CC) d'évaluation de la sécurité des systèmes d'information* (1994)

- Evaluation Assurance Level

  Définition de 7 niveaux d'évaluation (EAL1-EAL7)

  EAL7 exige l'emploi de méthodes formelles

# Méthodes formelles

- Par opposition aux tests: coûteux et non exhaustifs
- Spécification formelle

  Modélisation dans des langages logiques ad hoc dont la sémantique est définie mathématiquement
- Preuve formelle

  Démonstration mathématique dans une logique ad hoc que la spécification satisfait les propriétés requises
- Vérification de la preuve

  Par des systèmes automatisés d'aide à la preuve

  Réflexion sur la preuve (méta-mathématiques)

# Plan de l'exposé

- Un exemple de logiciel critique

  Système concurrent dont l'exécution est infinie

- Sa modélisation mathématique

- Preuve formelle de sûreté

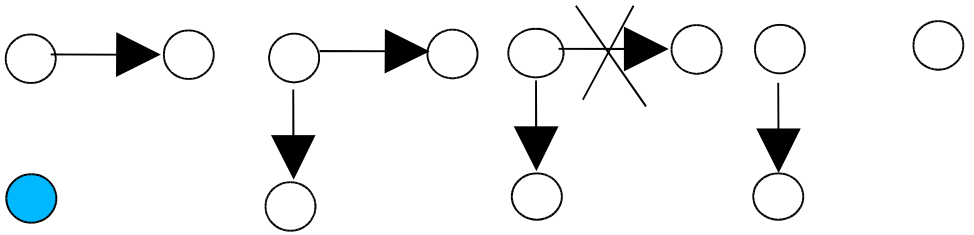
  Outils logiques

- La vérification des preuves

  Avec un assistant de preuves (logiciel)

# Un système embarqué sur cartes à puce: *récupérateur de mémoire*

- SmartCard, nouvelle génération de cartes à puce
  - Multi-applications
  - Evolutive
  - Langage JavaCard
- Allocation dynamique de mémoire



- Récupération de mémoire

# Un système embarqué sur cartes à puces: *récupérateur de mémoire*

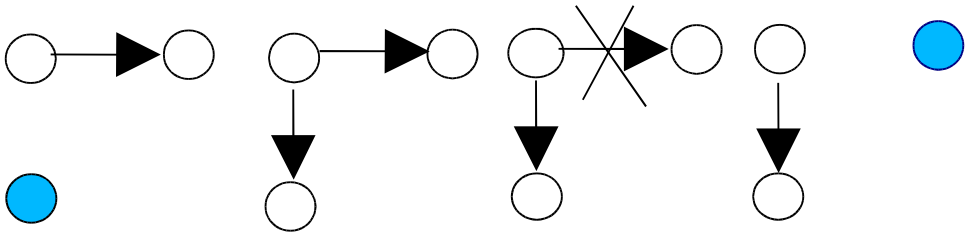
- SmartCard, nouvelle génération de cartes à puce

  Multi-applications

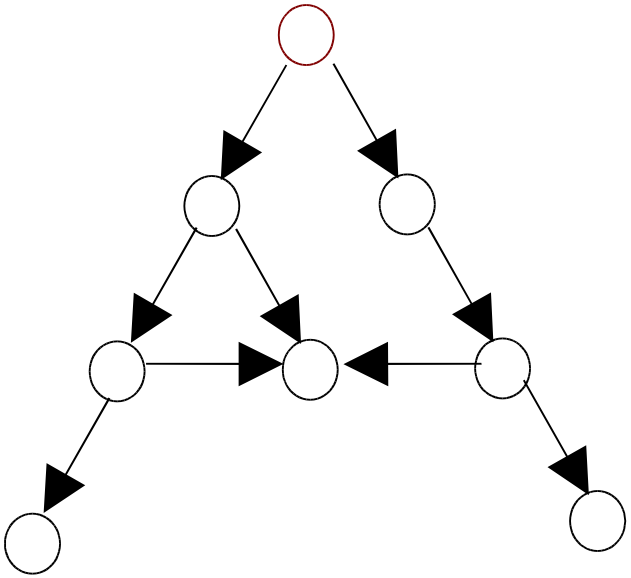
  Evolutive

  Langage JavaCard

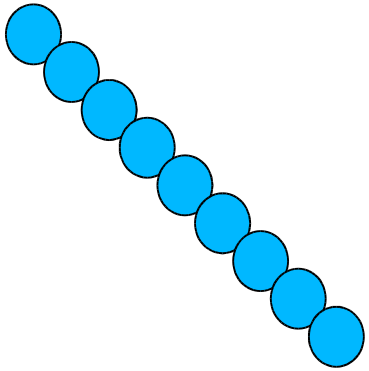
- Allocation dynamique de mémoire



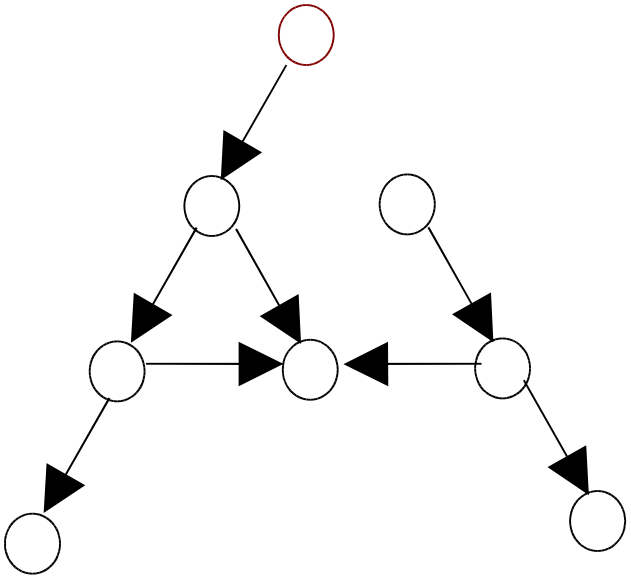
- Récupération de mémoire (*GC : Garbage collector*)

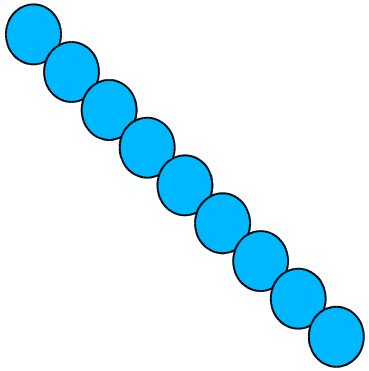


Le tas des cellules occupées



Les cellules libres



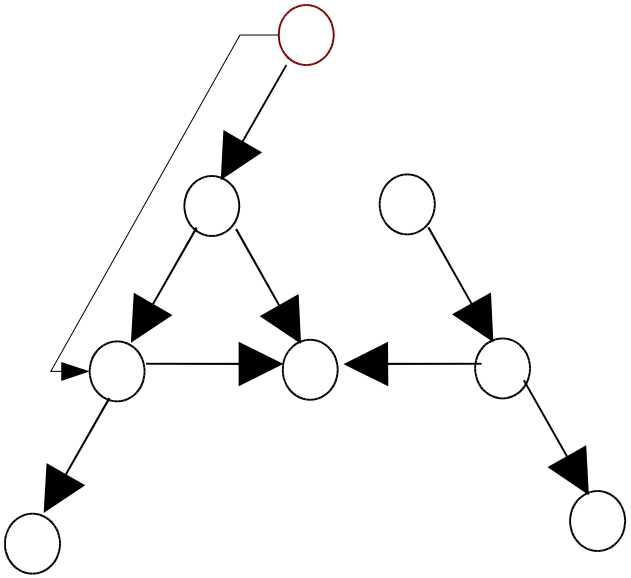


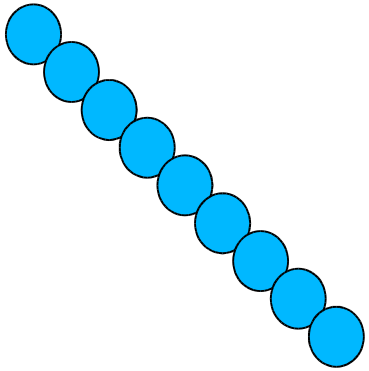
Le tas des cellules occupées

Les cellules libres

# Les actions du programme utilisateur (Mutateur)

*Suppression d'un arc*



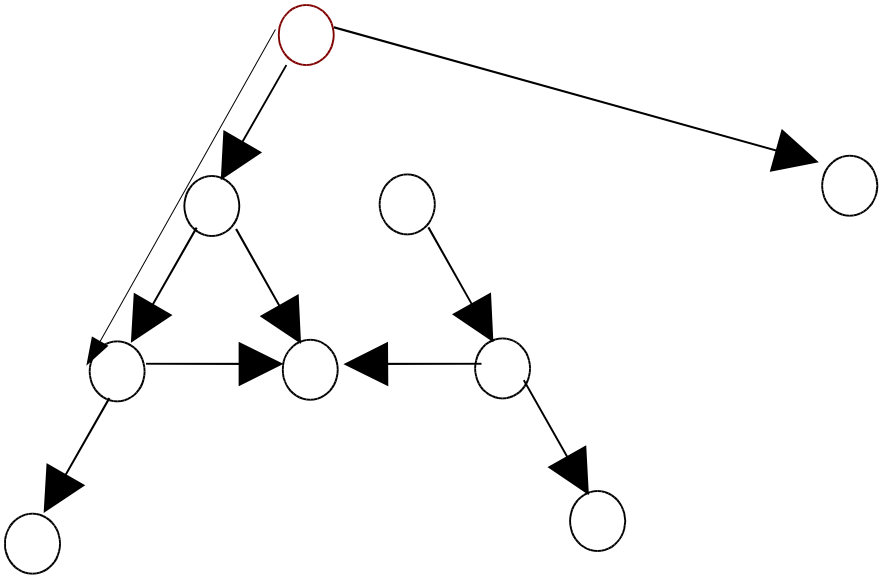


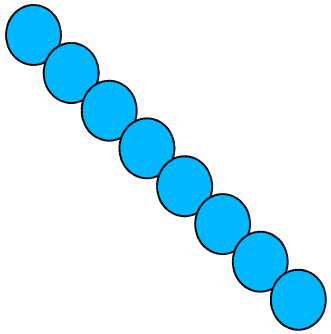
Le tas des cellules occupées

Les cellules libres

# Les actions du programme utilisateur (Mutateur)

*Rajout d'un arc entre 2 noeuds accessibles*



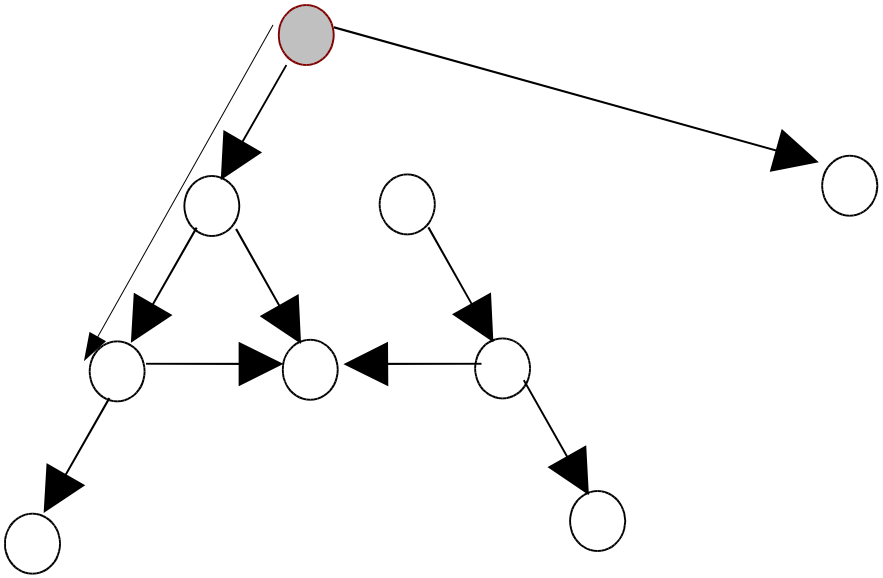


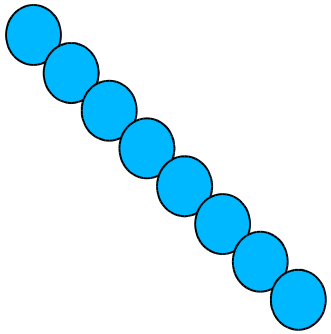
Le tas des cellules occupées

Les cellules libres

# Les actions du programme utilisateur (Mutateur)

*Allocation d'une nouvelle cellule*



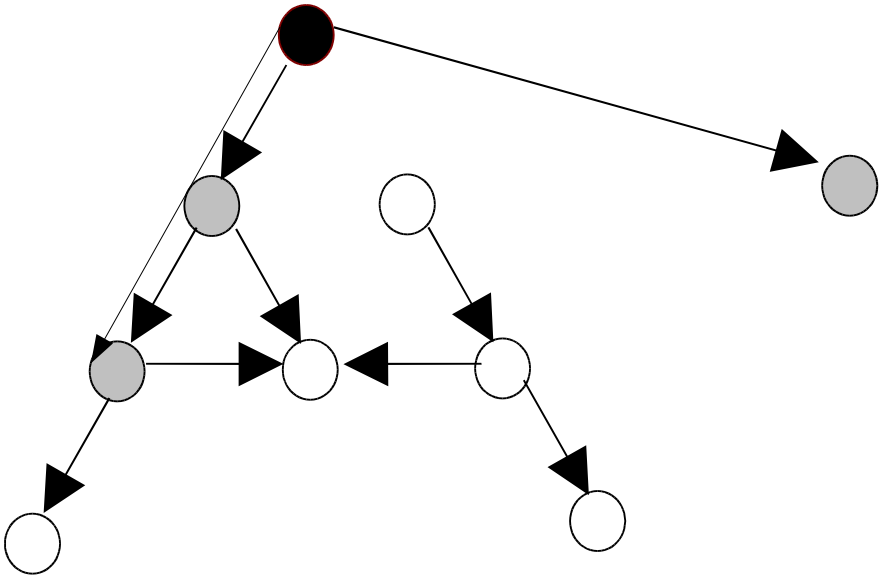


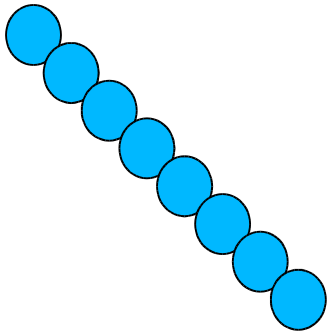
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Marquage*



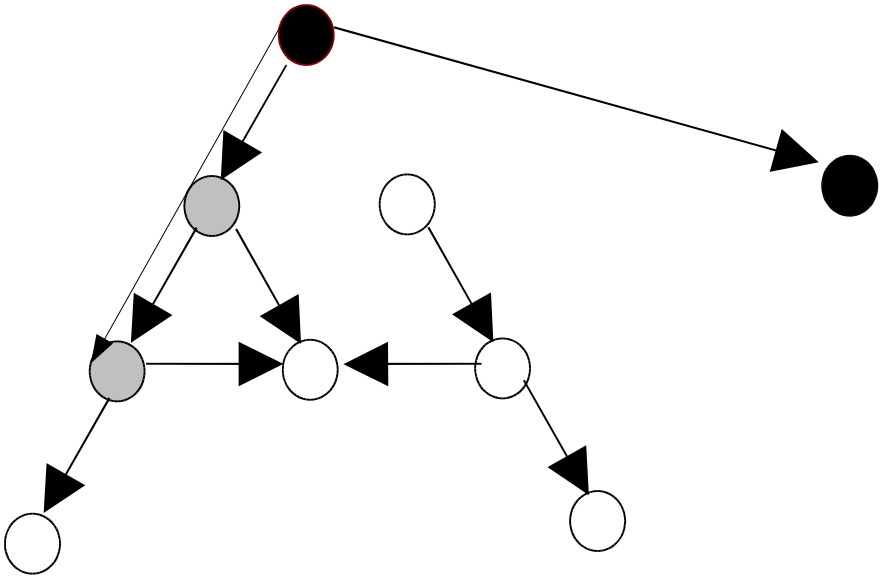


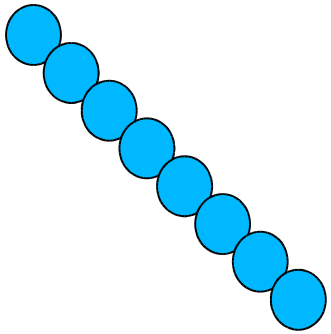
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Marquage*



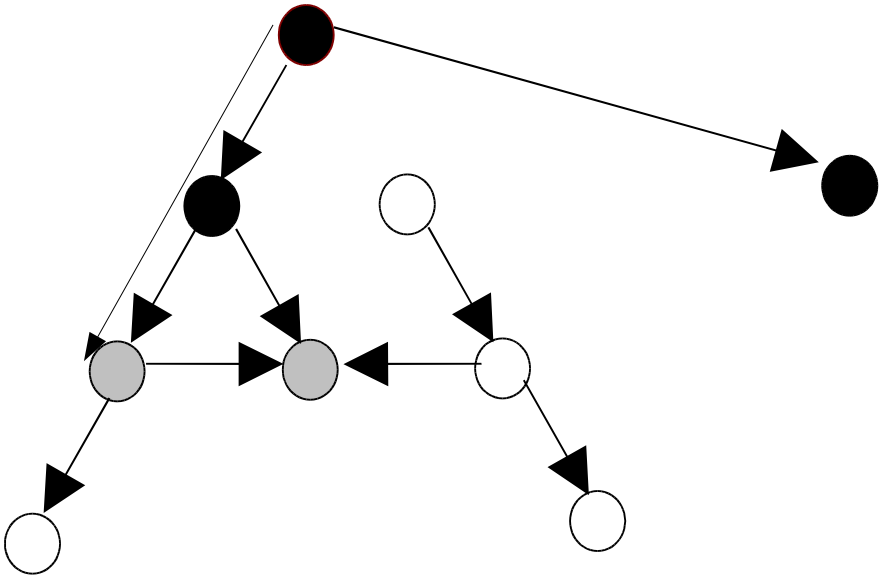


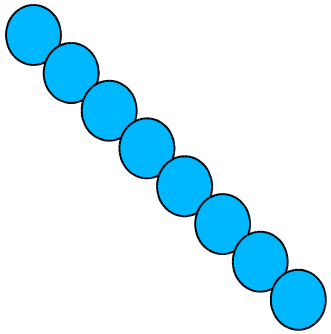
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Marquage*



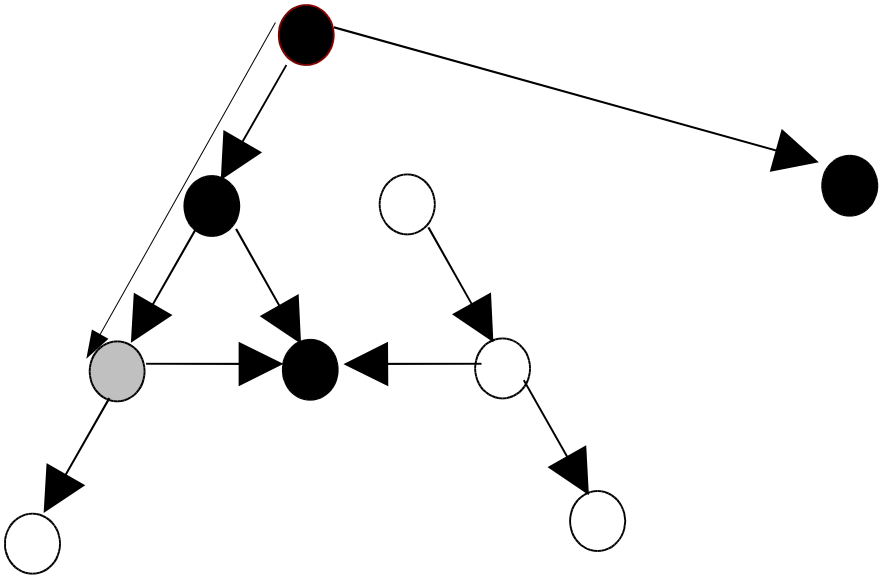


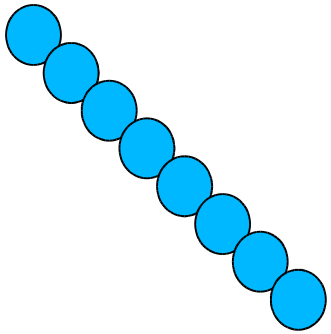
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Marquage*



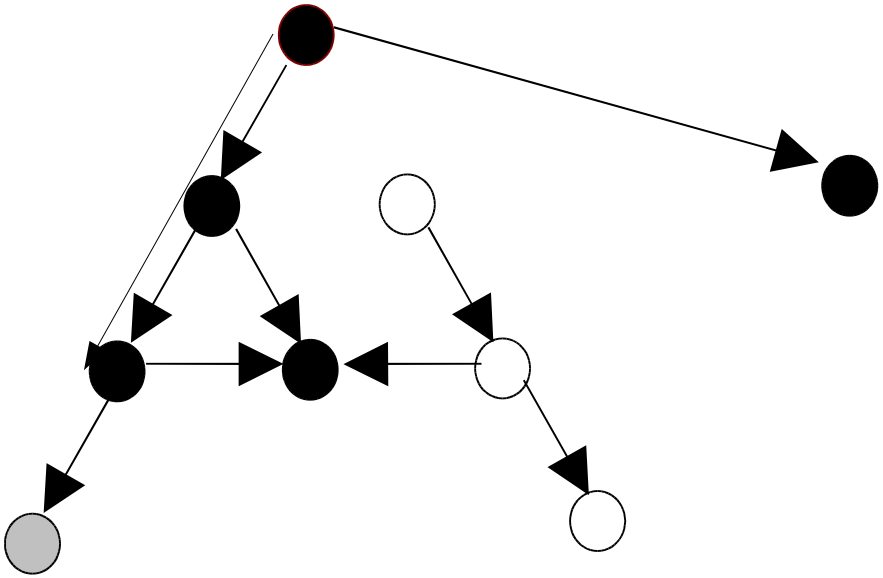


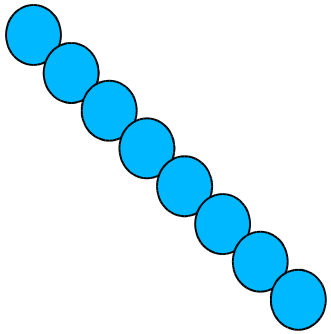
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Marquage*



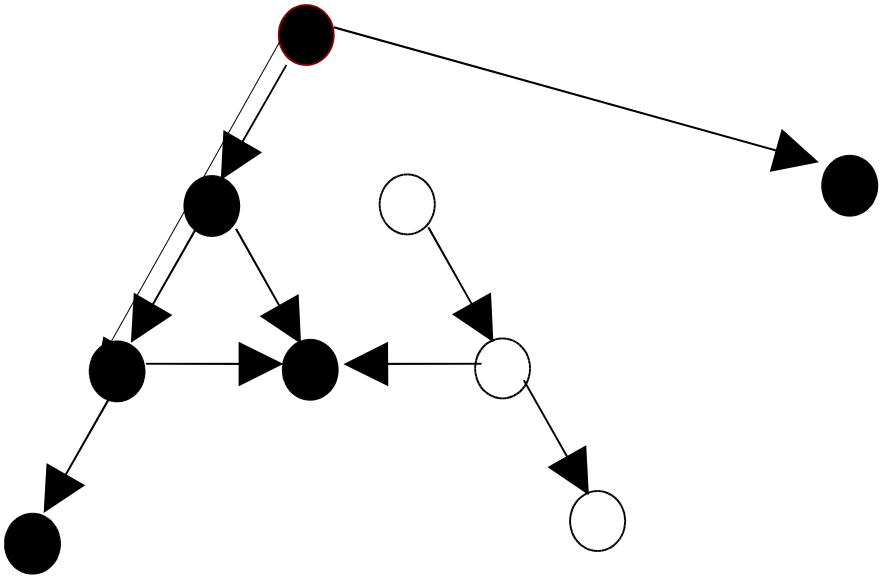


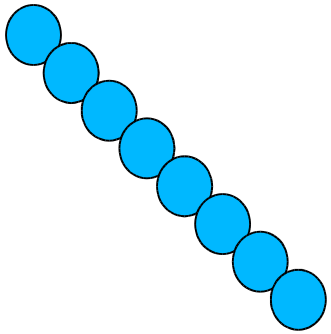
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Marquage*



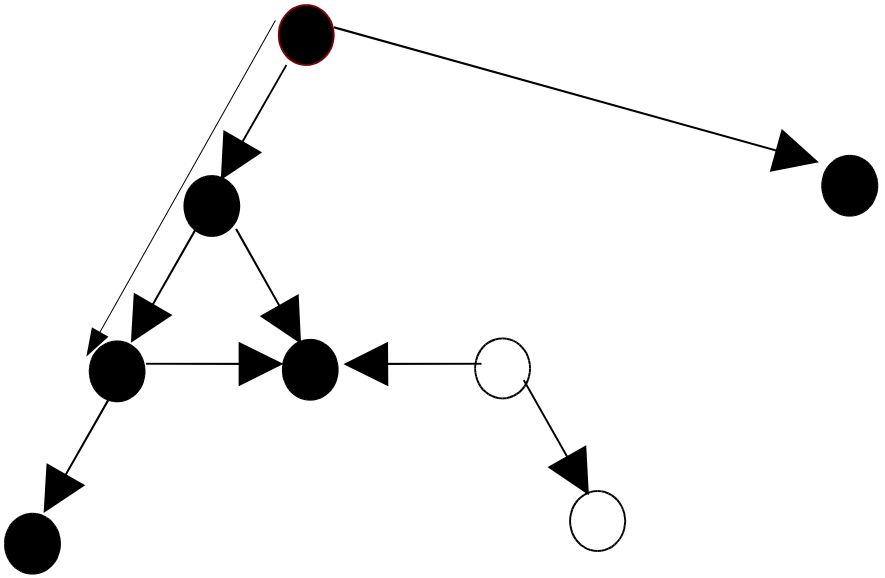


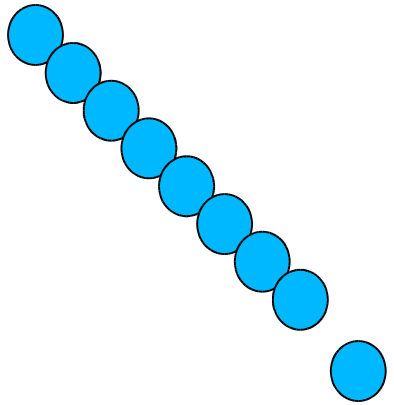
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Marquage*



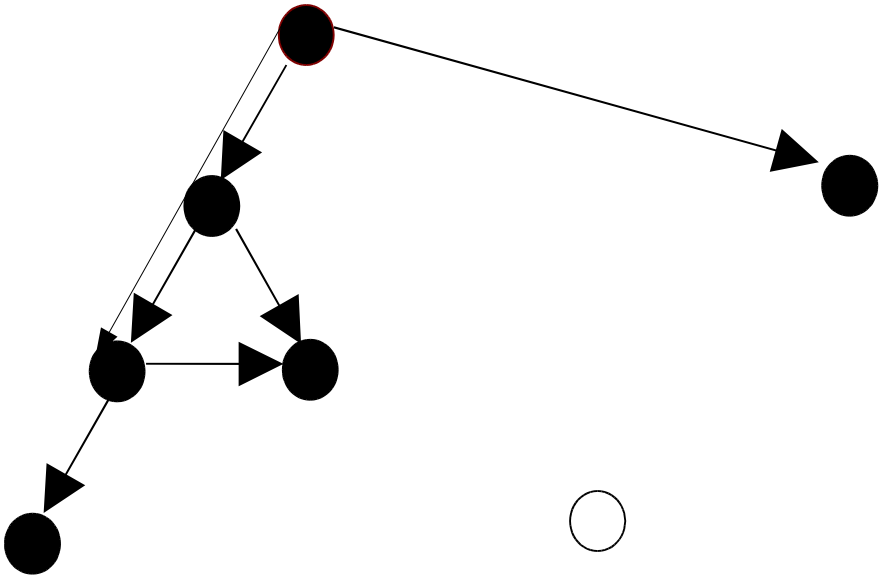


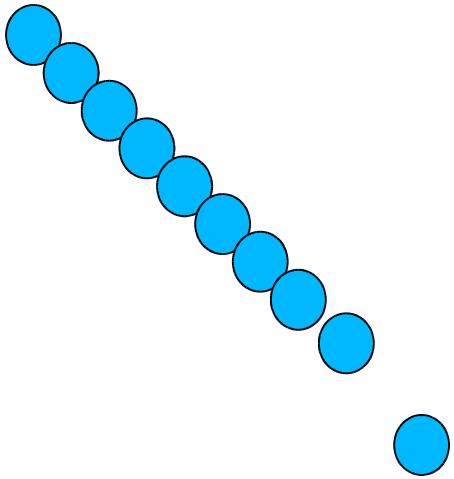
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Balayage*



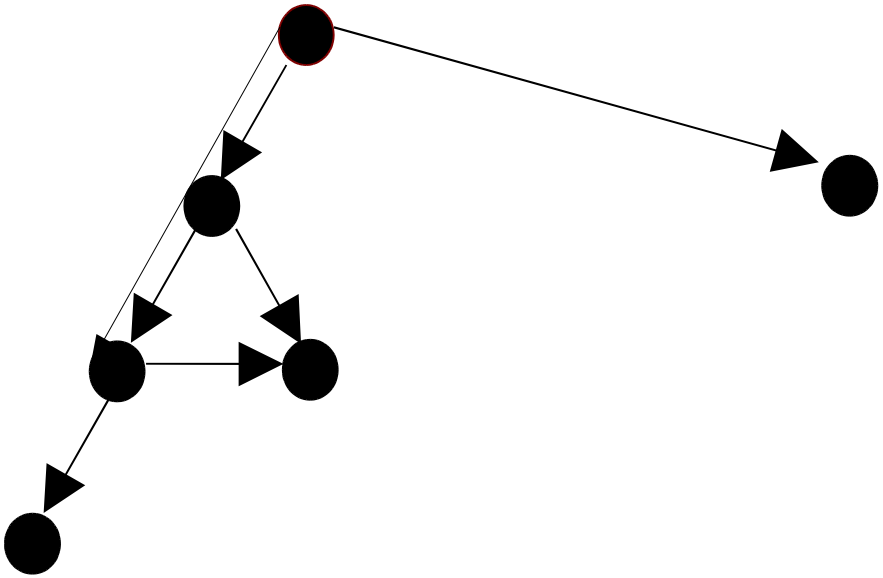


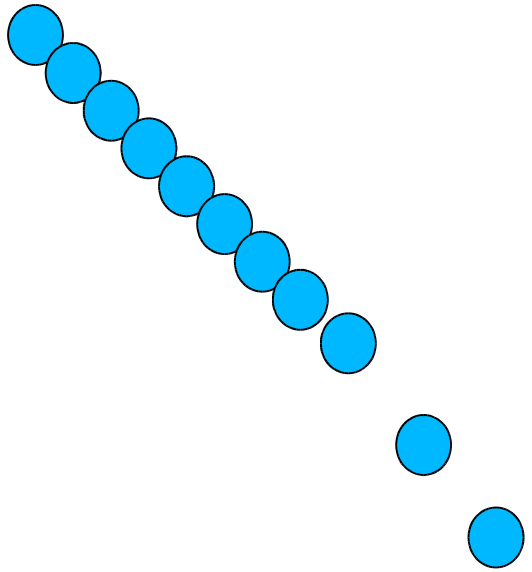
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Balayage*



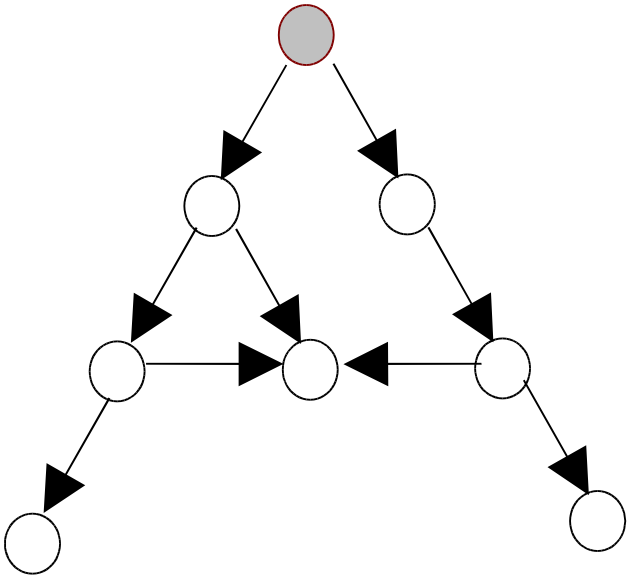


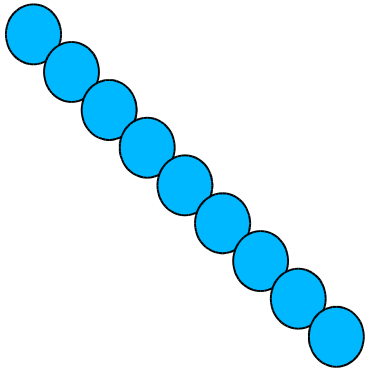
Le tas des cellules occupées

Les cellules libres

# Les actions du récupérateur (GC)

*Balayage*

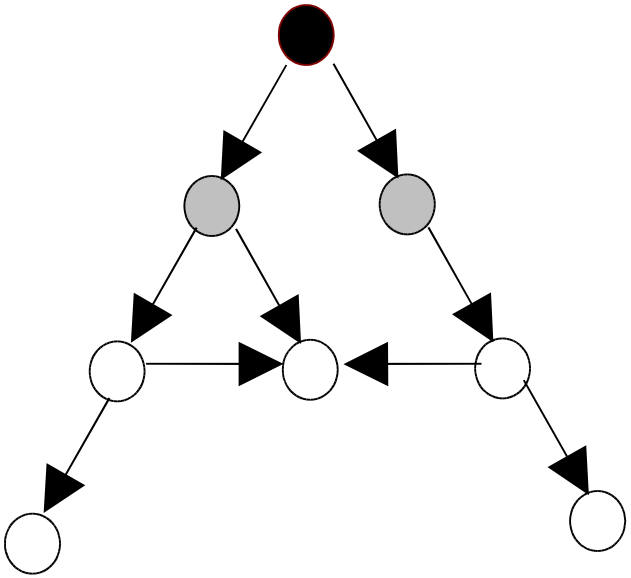


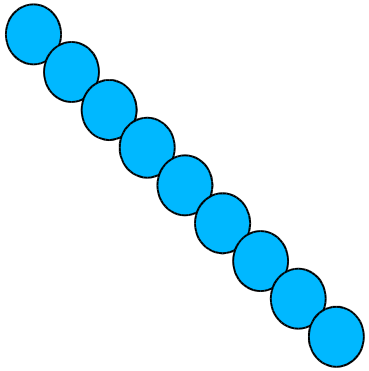


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

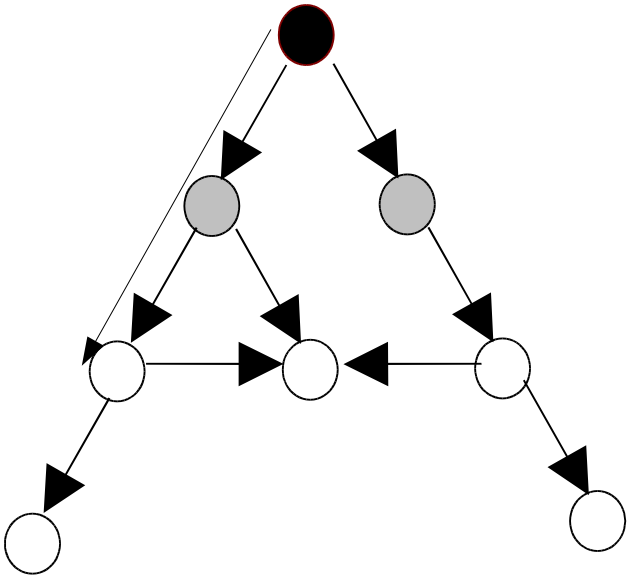


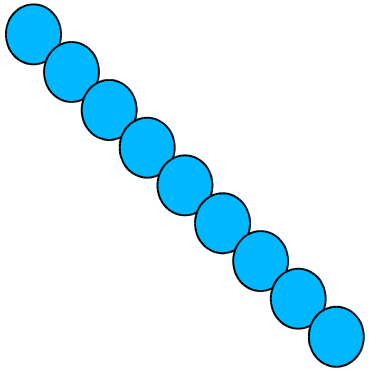


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

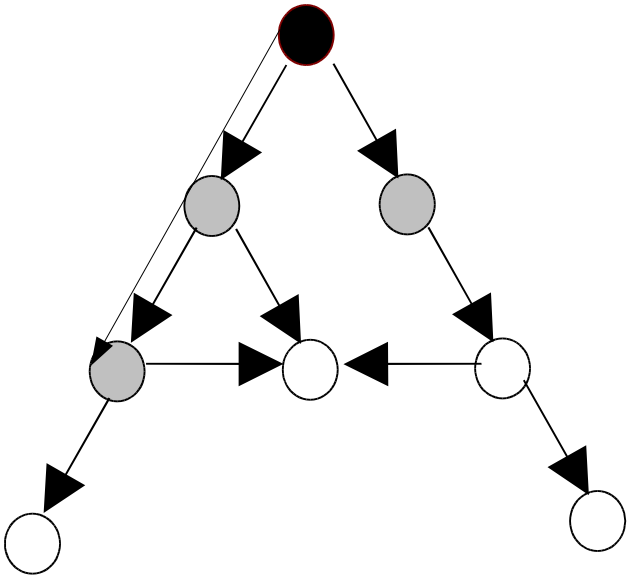


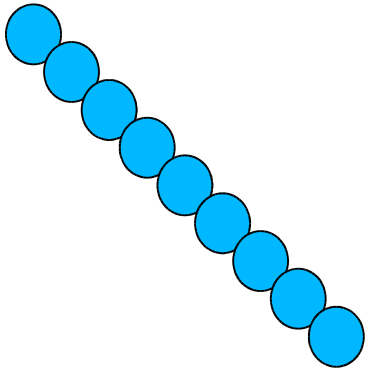


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

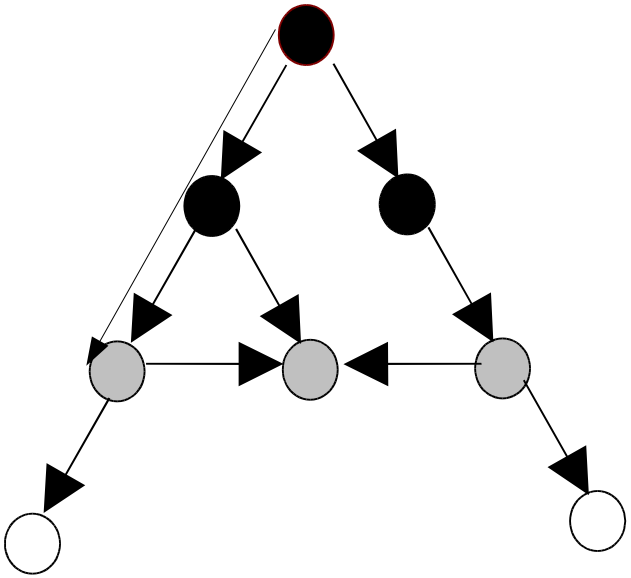


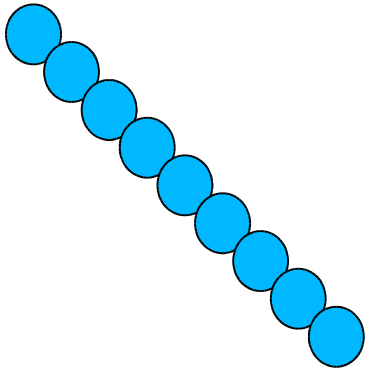


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

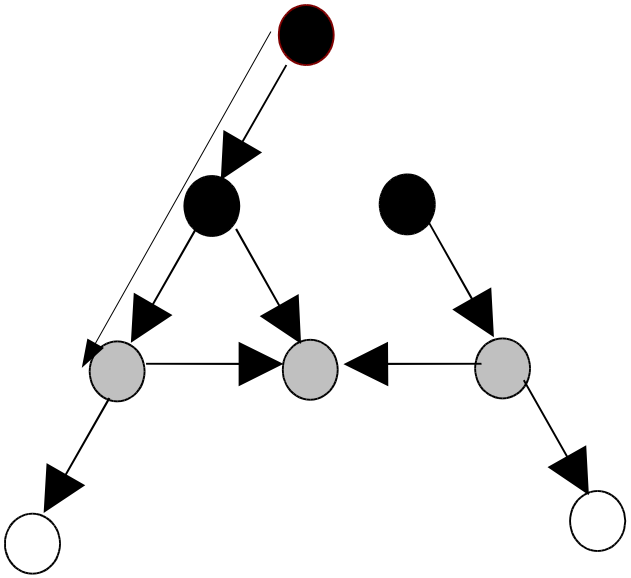


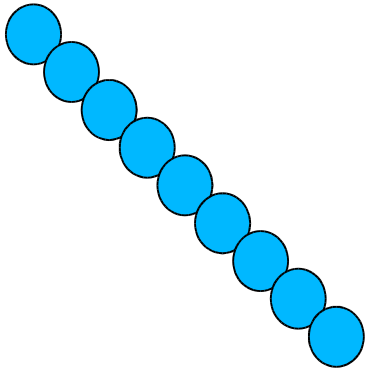


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

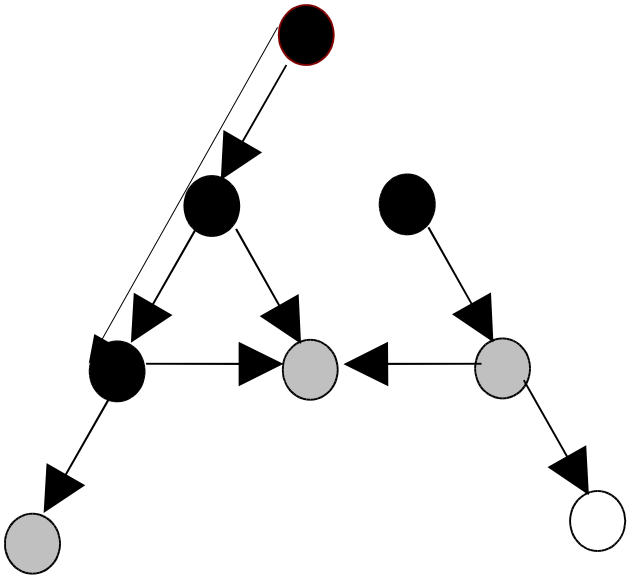


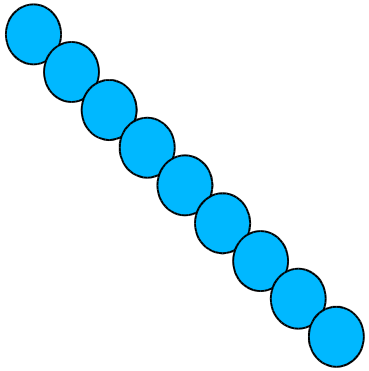


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

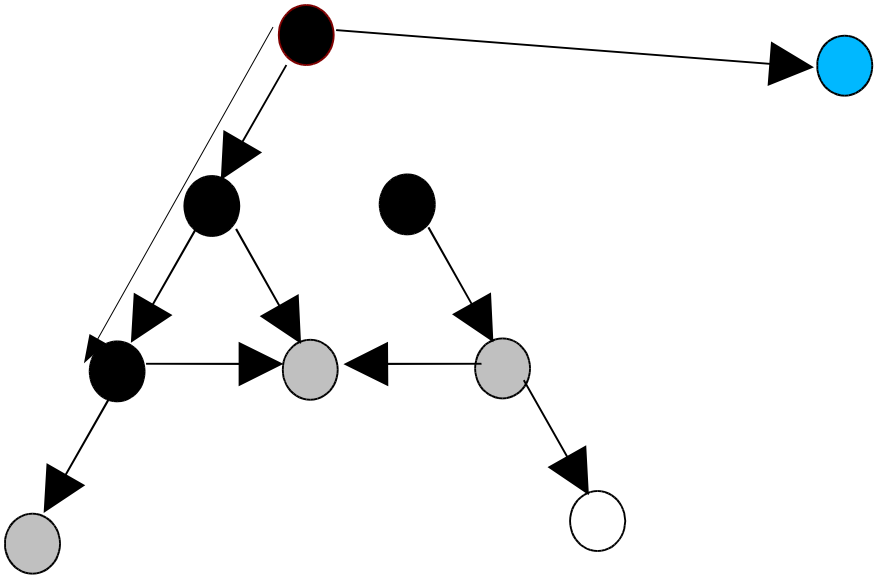


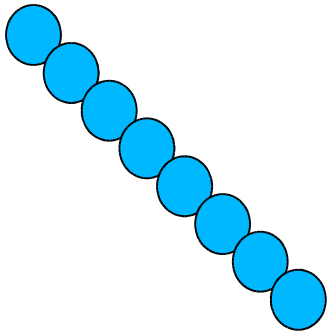


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

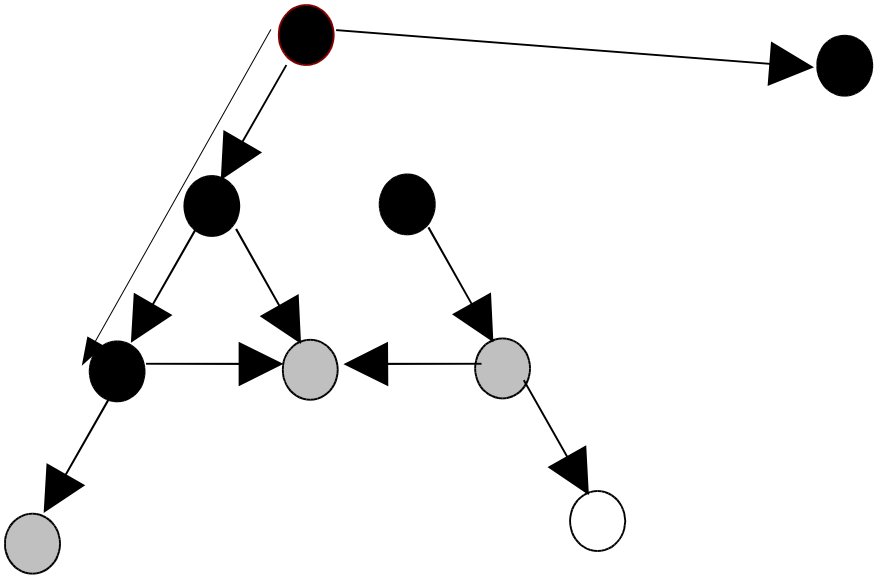


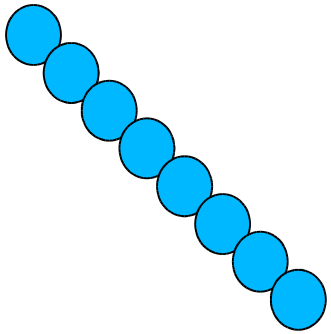


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

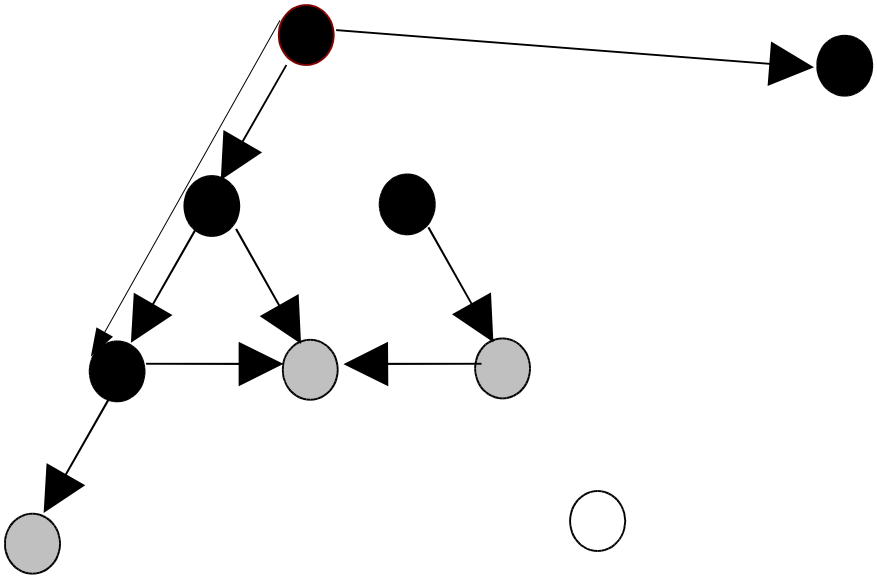


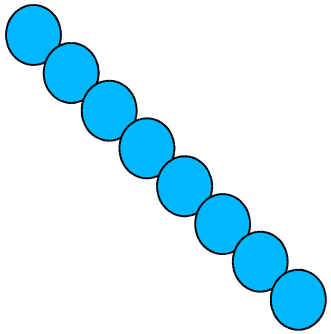


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

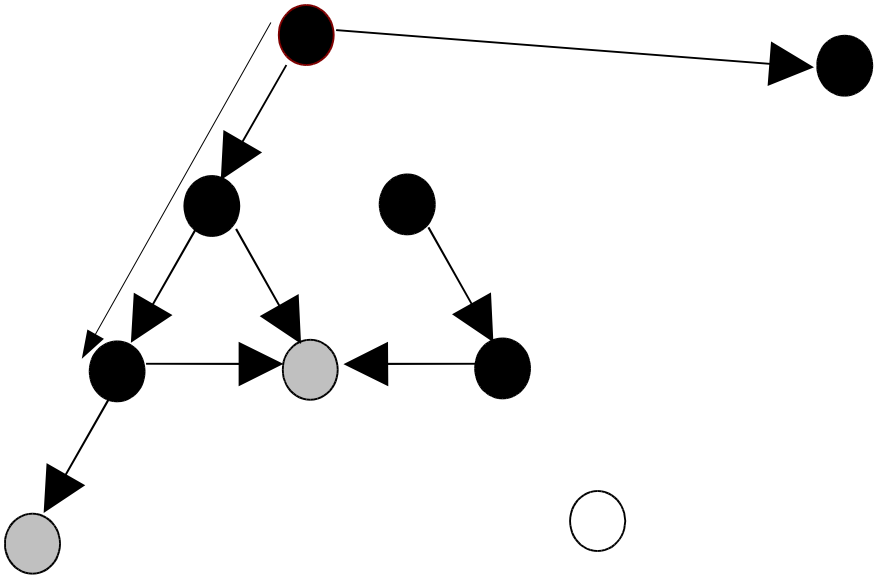


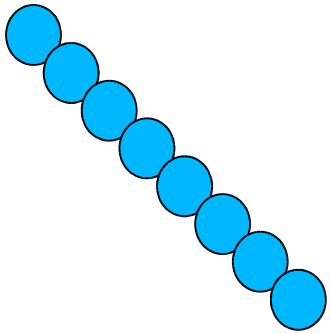


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

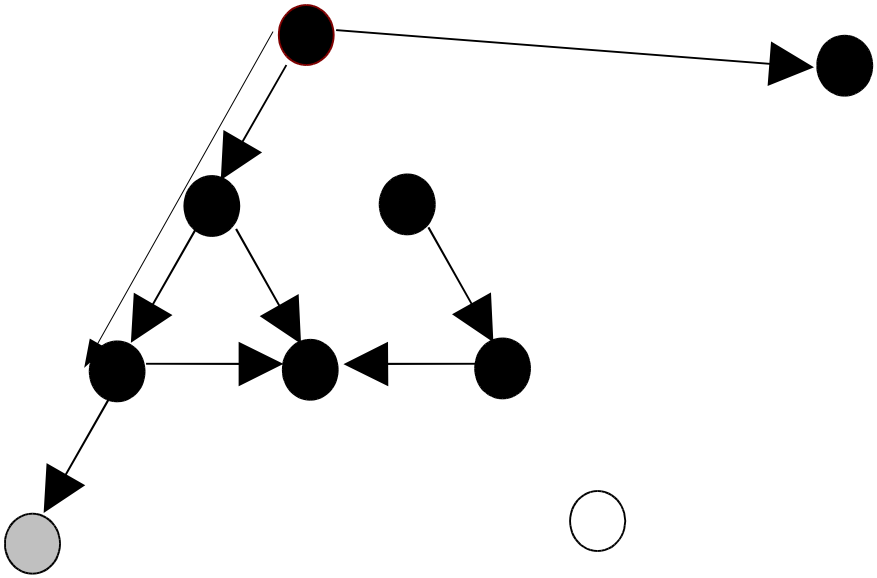


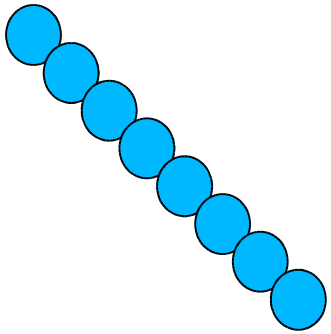


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

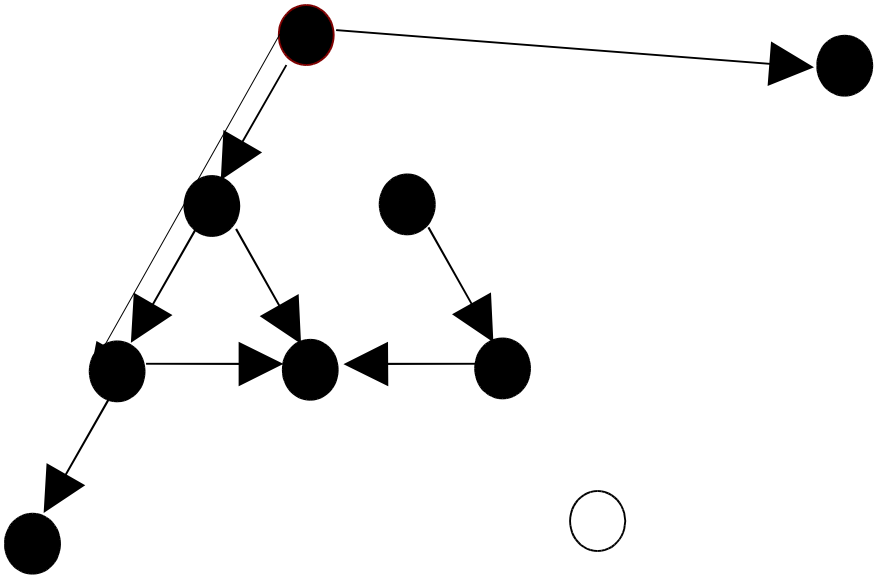


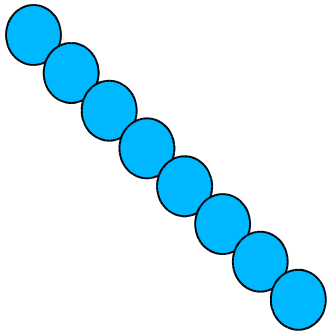


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

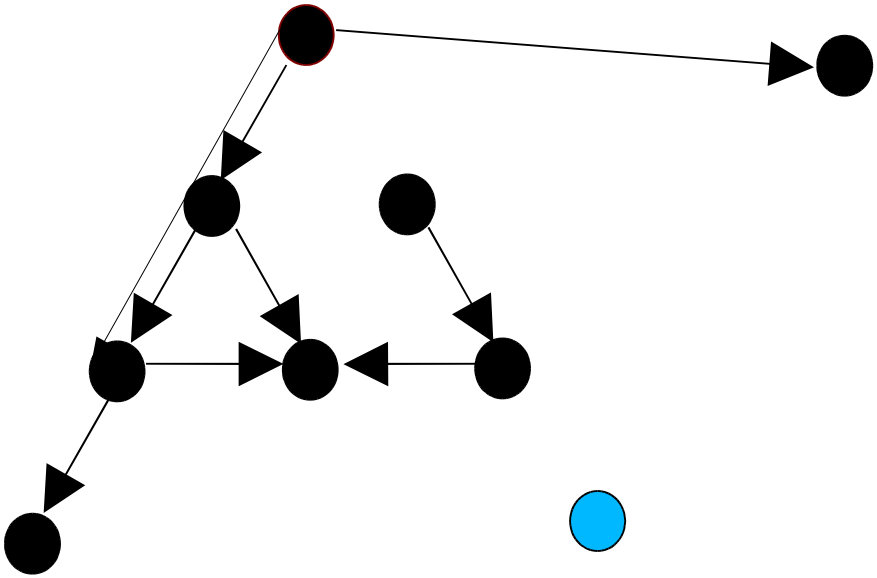


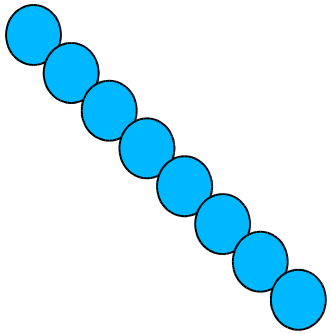


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes

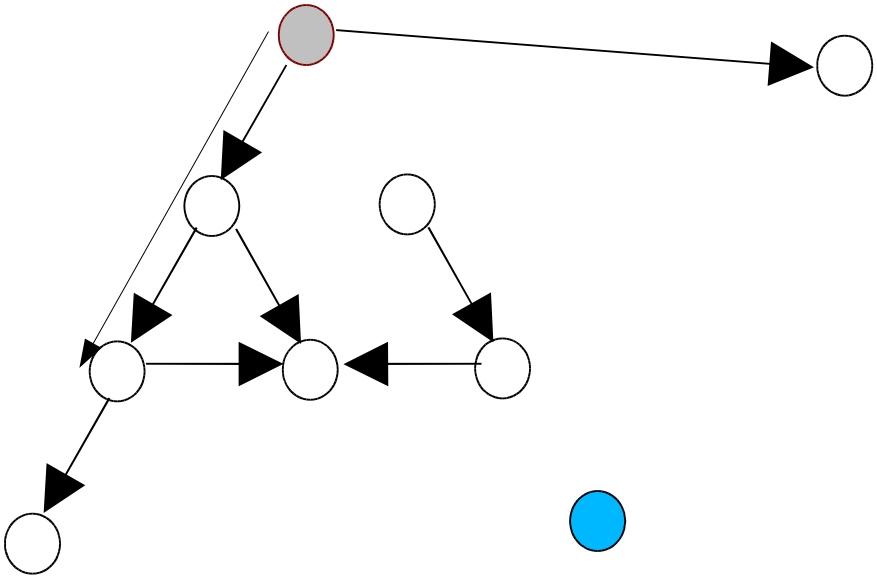


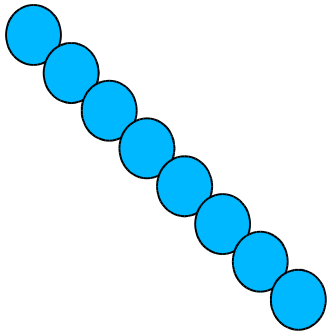


Le tas des cellules occupées

Les cellules libres

## Entrelacement des actions des 2 programmes





Le tas des cellules occupées

Les cellules libres

# Entrelacement des actions des 2 programmes

*Ca continue indéfiniment!*

# Modélisation

Cellules = un ensemble quelconque

Racine = un élément donné de Cellules

Couleurs = {Blanc, Gris, Noir, Libre}

Tas : une relation binaire t sur les cellules

$$t(c_1, c_2) \in \{vrai, faux\}$$

Marquage : une fonction de Cellules →Couleurs

Contrôle = {Marquage, Balayage, Mutateur}

Etat du système : s = (t, m, ctl)

t : Tas, m: Marquage, ctl : Contrôle

# Modélisation (suite)

- **Etat initial**

  Pas d'arc, racine: noire, autres cellules: libres

  ctl = *Mutateur*

- **Transitions**

  Relations entre 2 états correspondant aux 9 actions

  a : *supprimer un arc*

$s \xrightarrow{a} s'$

$s = (t, m, \text{Mutateur})$  $s' = (t', m, \text{Mutateur})$

$\exists (c_1, c_2)$  $t(c_1, c_2) = \text{vrai}$  *et*  $t'(c_1, c_2) = \text{faux}$

t et t' identiques ailleurs

# Modélisation (suite)

## Système de transitions infinies

### Exécutions

- Suites infinies d'états $\sigma = (\sigma_0, \sigma_1, ..., \sigma_n, ...)$
- $\forall i$, $\sigma_{i+1}$ se déduit de $\sigma_i$ par une des 9 transitions
- $\sigma_0$ est l'état initial

# Logique temporelle linéaire

P, Q : prédicats sur les suites infinies d'états

$\sigma = (\sigma_0, \sigma_1, ..., \sigma_n, ...)$ $\quad \sigma_{|t} = (\sigma_t, \sigma_{t+1}, ..., \sigma_n, ...)$

## Opérateurs temporels

$(\Box P\ \sigma) \equiv \forall\ i\ (P\ \sigma_{|i})$ $\quad (\Diamond P\ \sigma) \equiv \exists\ i\ (P\ \sigma_{|i})$

$(\Box\Diamond P\ \sigma) \equiv$ P est satisfaite infiniment souvent sur $\sigma$

$(P\ \mathcal{U} Q\ \sigma) \equiv \exists\ i\ (\forall j<i\ (P\ \sigma_{|j})\ ) \wedge (Q\ \sigma_{|i})$

# Logique temporelle linéaire

- Sûreté : Une cellule libérée est *toujours* inaccessible

# Logique temporelle linéaire

- Sûreté : Une cellule libérée est *toujours* inaccessible
- Vivacité : Toute cellule inaccessible sera *finalement* libérée

# Logique temporelle linéaire

- Sûreté : Une cellule libérée est *toujours* inaccessible
- Vivacité : Toute cellule inaccessible sera *finalement* libérée

Hypothèses d'équité

- Equité faible : une action *toujours* exécutable *finira* par être exécutée
- Equité forte : une action exécutable *infiniment souvent* *finira* par être exécutée

# Logique temporelle linéaire

- Sûreté : Une cellule libérée est *toujours* inaccessible
- Vivacité : Toute cellule inaccessible sera *finalement* libérée

Hypothèses d'équité

# Sûreté

Invariant: *toujours* satisfait par les exécutions

- La racine est grise ou noire
- Si le contrôle est *balayage*, pas de cellules grises
- Pas d'arc d'une cellule noire vers une blanche
- Si un noeud est accessible, il n'est pas libre
- S'il n'y a pas de cellules grises, toute cellule accessible est noire

*vrai sur l'état initial et conservé par chacune des 9 transitions*

# Vivacité

- Preuves par induction
- Mesure sur les états dans un ordre bien fondé

  *Pas de suites infinies strictement décroissantes*
- Sous certaines hypothèses, la mesure décroît.

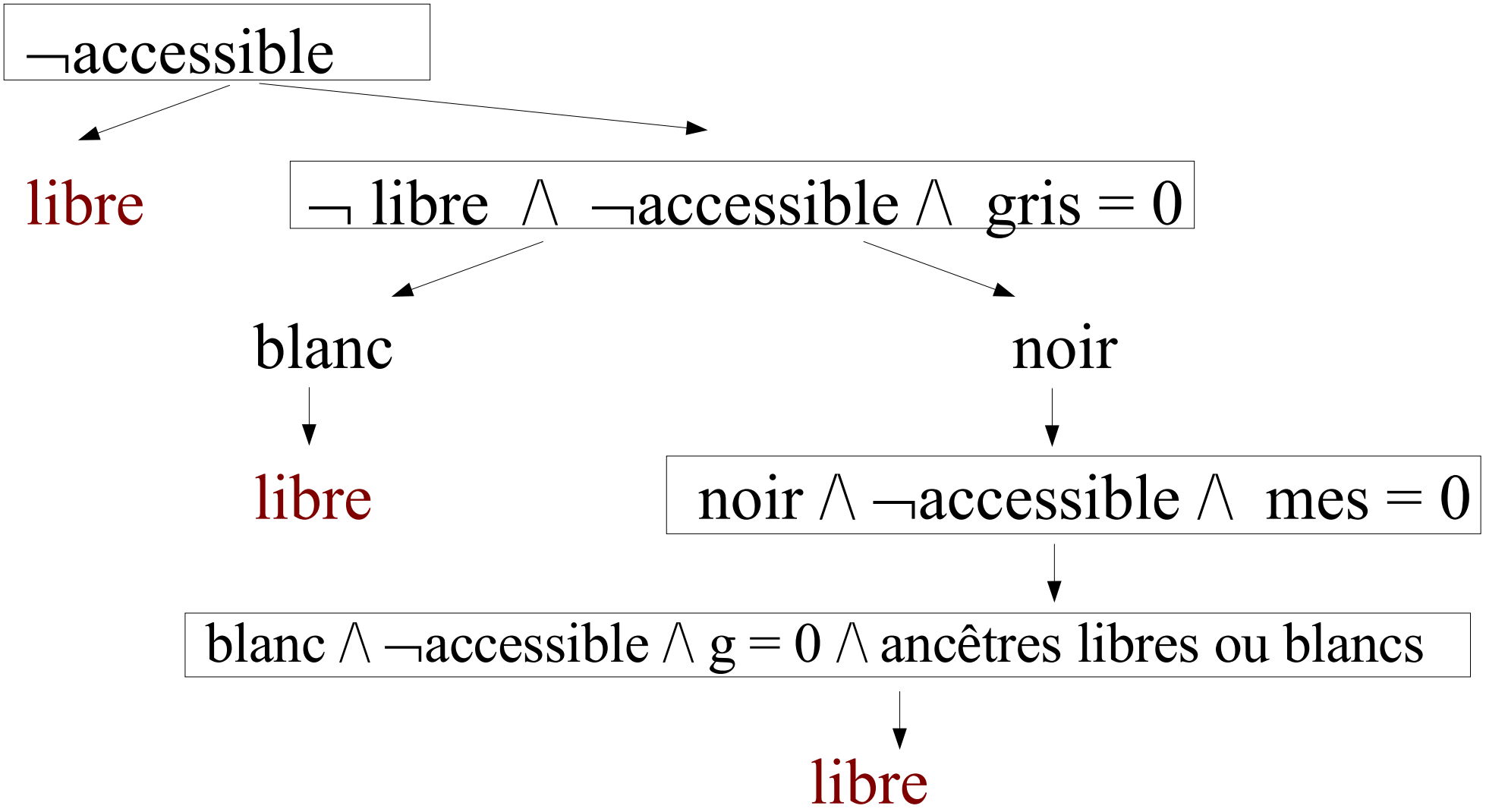
# Vivacité

Les cellules blanches, grises et noires sont *toujours* en nombre fini

mesure := #cellules blanches + #cellules grises

Sur toute exécution faiblement équitable, *infiniment souvent* la mesure est nulle

# Vivacité


¬accessible
libre
¬ libre /\ ¬accessible /\ gris = 0
blanc
noir
libre
noir /\ ¬accessible /\ mes = 0
blanc /\ ¬accessible /\ g = 0 /\ ancêtres libres ou blancs
libre

# Vivacité

- $(A\ n\ \sigma) \rightarrow ((A\ n)\ \mathcal{U}\ (B\ n)\ \sigma)$
- $(C\ n\ \sigma) \rightarrow ((C\ n)\ \mathcal{U}\ (D\ n)\ \sigma)$
- $(C\ n\ \sigma) \rightarrow (\Diamond(D\ n)\ \sigma)$
- $(E\ n\ \sigma) \rightarrow ((E\ n)\ \mathcal{U}\ (G\ n)\ \sigma)$
- $(F\ n\ \sigma) \rightarrow ((F\ n)\ \mathcal{U}\ (G\ n)\ \sigma)$
- $(G\ n\ \sigma) \rightarrow ((G\ n)\ \mathcal{U}\ (H\ n)\ \sigma)$
- $(K\ n\ \sigma) \rightarrow ((K\ n)\ \mathcal{U}\ (K'\ n)\ \sigma)$
- $(K\ n\ \sigma) \rightarrow ((K\ n)\ \mathcal{U}\ (C\ n)\ \sigma)$
- $(K\ n\ \sigma) \rightarrow (\Diamond(D\ n)\ \sigma)$

# Ces preuves *papier-crayon* assurent-elles la fiabilité du programme?

Longues et fastidieuses

De très nombreux cas à examiner

Pièges logiques

Subtilités ...

⇓

*Risque d'erreurs !*

# De la correction des programmes à celle des preuves

Première approche : *automatisation des preuves*

Limitations

- *Problèmes décidables uniquement*
- *Explosion combinatoire*

# De la correction des programmes à celle des preuves

## Deuxième approche

Construction interactive de la preuve avec un logiciel

Vérification automatique de sa correction

Assistants de preuves : *Coq (INRIA)*

## La preuve comme objet d'étude

*Interaction Logique – Informatique*

Sa représentation formelle (codage)

Sa vérification

Quelle différence entre une preuve et un programme?

# *Théorie des types*
## *« Proofs as programs » (Curry-Howard)*

$a : A$

| | |
|---|---|
| A ensemble | A proposition |
| a élément de A | a preuve de A |

$f : A \to B$

| | |
|---|---|
| f fonction (algorithme) | f preuve |
| de A dans B | de l'implication $A \to B$ |

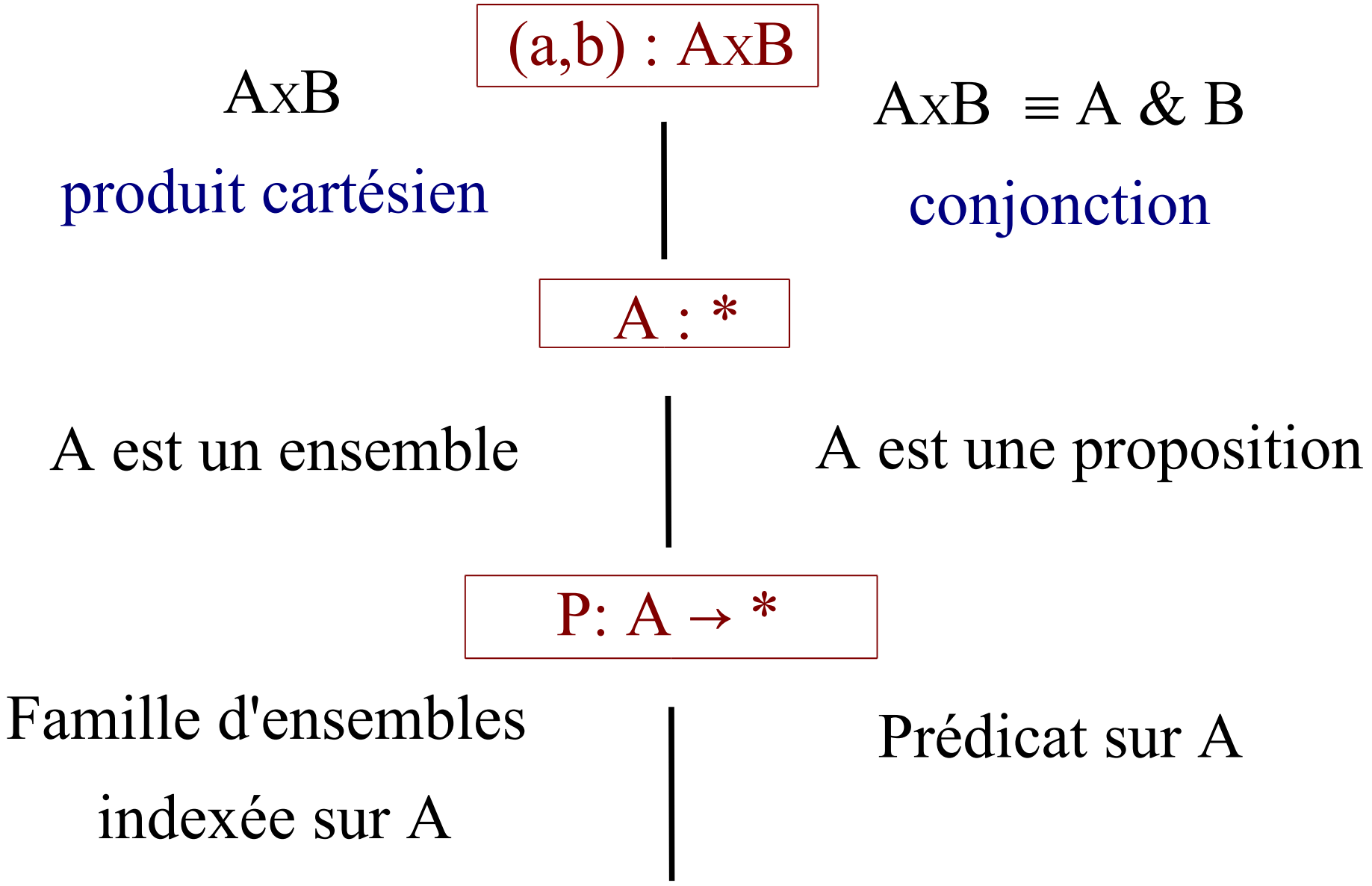
$$\frac{f : A \to B \quad a : A}{f(a) : B}$$

| | |
|---|---|
| Application | Modus ponens |

Une proposition = { Ses preuves effectives }

*f preuve correcte de P si le terme f est de type P (décidable)*

# *Théorie des types*
## *« Proofs as programs » (Curry-Howard)*



# *Théorie des types*
## *« Proofs as programs » (Curry-Howard)*

$$\frac{x : A \vdash h : P(x)}{(x{:}A \hookrightarrow h) : (\forall x{:}A)\ P(x)}$$

Produit d'ensembles

$\Pi x{:}A.(P\ x)$

Proposition universellement quantifiée

Polymorphisme: types ↳fonctions

$$(A : * \hookrightarrow Id_A) : (\forall\ A{:}*\ )(A \rightarrow A)$$

Fonctions des types dans les types

$$\& \equiv A, B{:}* \hookrightarrow A \times B$$

# Théorie des types

## *Le calcul des constructions*

- Expressif, uniforme, puissant, logique d'ordre supérieur, induction, co-induction
- Intuitionniste, constructive, pas de tiers exclus

---

$$H : (\forall x{:}A)\ (\exists y{:}B)\ P(x, y)$$

*Construire pour tout x un témoin y tel que P(x, y)*

$H = (f : A \rightarrow B,\ pr)$ $\qquad$ $pr : (x{:}A)\ P(x, f(x))$

# Axiomatisation des ensembles finis

$(\text{cardinal } \emptyset\ 0)$

$a_0 \notin Q \rightarrow (\text{cardinal } Q\ n) \rightarrow (\text{cardinal } Q \cup \{a_0\}\ (n+1))$

# Induction, Co-induction

- Inductive list : Set :=

  nil : list |

  cons: A → list → list.

- CoInductive list : Set :=

  nil : list |

  cons : A → list → list.

- CoInductive stream : Set :=

  cons :

  state → stream → stream.

# Axiomatisation de LTL (1)

$(\Box P\ \sigma) \equiv \forall\ i\ \ (P\ \sigma_{|i})$

CoInductive $\Box P$ : stream $\rightarrow$ Prop:=

C_always : $(\forall \sigma$:stream)

$(P\ \sigma) \rightarrow (\Box P\ (tl\ \sigma)) \rightarrow (\Box P\ \sigma)$

$(\Diamond P\ \sigma) \equiv \exists\ i\ \ (P\ \sigma_{|i})$

Inductive $\Diamond P$ : stream $\rightarrow$ Prop:=

ev_h: $(\forall \sigma$:stream) $(P\ \sigma) \rightarrow (\Diamond P\ \sigma)$ |

ev_t : $(\forall \sigma$:stream) $(\Diamond P\ (tl\ \sigma)) \rightarrow (\Diamond P\ \ \sigma)$

# $X : (\forall \sigma{:}\mathrm{stream})(\Box P\ \sigma) \rightarrow (\Box\Box P\ \sigma)$

X à construire

---

soit $\sigma = (\mathrm{cons}\ \sigma_0\ \sigma')$ et $h : (\Box P\ \sigma)$

$p : (P\ \sigma)$ $\quad$ $h' : (\Box P\ \sigma')$

---

But $(\Box\Box P\ \sigma)$

$(\Box P\ \sigma)$ $\quad$ $(\Box\Box P\ \sigma')$

Hypothèse h

on utilise $(\Box P\ \sigma') \rightarrow (\Box\Box P\ \sigma')$

i.e. on applique récursivement

X à σ' et h' ! (pt fixe)

---

*fix X. λσ λh (C_always* $\sigma_0$ *σ' h (X σ ' h'))*

# Conclusion

- Quelques chiffres sur le récupérateur de mémoire
  - 600 lemmes
  - 100 définitions
  - 10 000 lignes de code
- Purement constructif

  1 axiome: décidabilité de l'égalité sur les cellules
- On a trouvé des erreurs!

# Conclusion

- Historique
  - Dijkstra et al. 1978 (GC 3 couleurs, preuve papier crayon)
  - Ben Ari (1984) (GC 2 couleurs, preuve papier crayon)
  - Russinoff (1992) (Preuve “mécanisée” de l'algorithme de Ben Ari)
- Vérification formelle de GC
  - Gontier (96), Havelund /Shankar (97), Jackson (98), Goguen/ Broosky/ Burstall (99), Verma/Goubault-Larrecq et al (00), Moreau/ Duprat (01).
- Logiques temporelles dans des assistants de preuves
  - Russinoff (92) (Boyer-Moore),
  - Crégut/Heyd (96) et Stehr (98) (Unity en Coq),
  - Miculan (01) et Sprenger (98) (μ-calcul en Coq)
  - Luna (CTL en Coq)