Analyse de forme de programmes Java

R. Guider
(Projet CROAP)

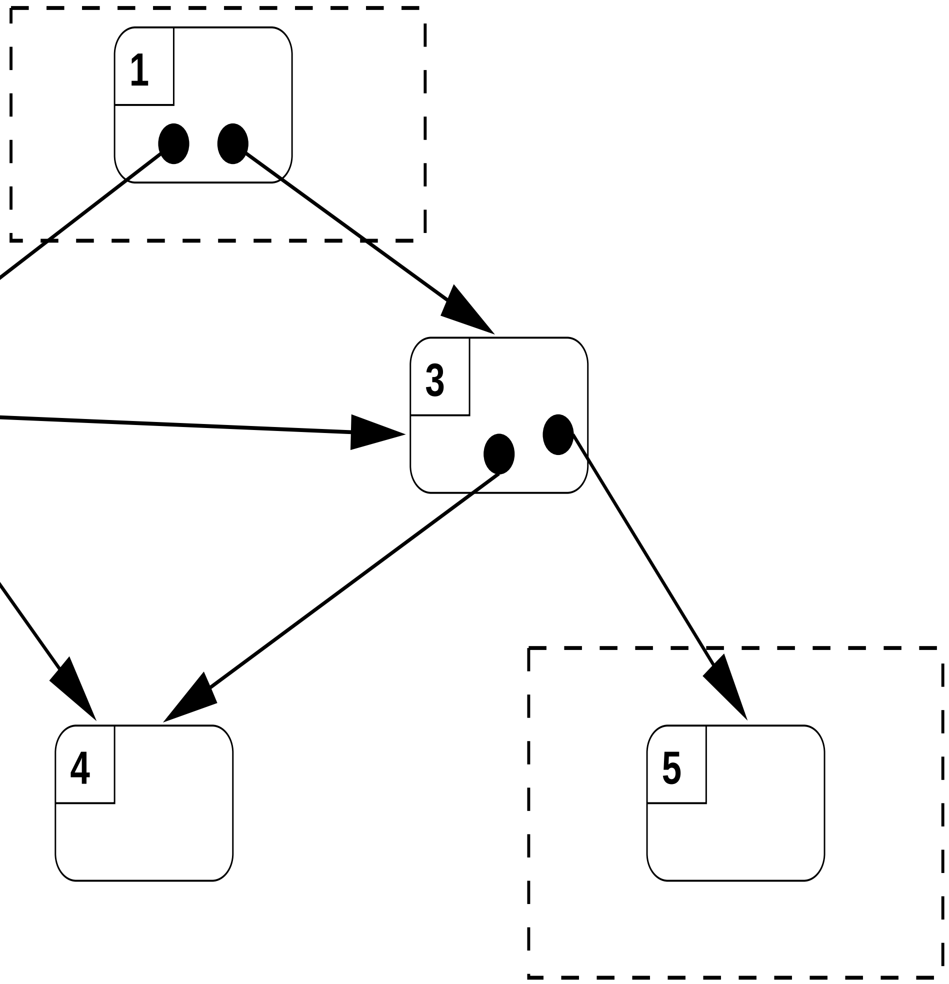
INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93
06902 Sophia-Antipolis Cedex

E-mail: Romain.Guider@sophia.inria.fr

Objectifs

- Découvrir automatiquement des propriétes des graphes d'objets.
- Utiliser ces propriétés pour aider à la parallélisation de code séquentiel

Attributs de forme



Domaines concrets

$$
\begin{array}{lcl}
Val & = & Int \cup Bool \cup Oid \\
Env & = & Id \to Val \\
Obj & = & Oid \times Type \times Env \times \\
Objs & = & \mathcal{P}(Obj)
\end{array}
$$

$$
\begin{array}{lcl}
Act & = & Insts \times Type \times Id \times Env \\
Stack & = & Act^{\star} \\
State & = & Stack \times Objs
\end{array}
$$

Sémantique *little-step* : relation de transition sur $State$.

Sous ensemble de Java Traite

- `if, while, affectation,création d'objets`
- `instruction d'échapement`
- `throw, try-catch`
- `appel de méthode`
- On ne traite pas les tableaux ni les attributs/méthodes de classe.
- On restreint la forme des affectations.

Domaines abstraits

$$
\begin{array}{lcl}
Env_a & = & Id \to Val_a \\
Obj_a & = & Oid_a \times Env_a \times Type \times Bool \\
Objects_a & = & \mathcal{P}(Obj_a) \\
Act_a & = & Insts \times Type \times Id \times Tok \times Bool \times Env_a \times \mathcal{P}(Retp) \\
State_a & = & Act_a \times Objects_a
\end{array}
$$

L'invariant abstrait est dans $\mathcal{P}(State_a)$

Abstraction des piles d'appels

$\hat{\alpha}_{Stack}$:

$$\boxed{\begin{array}{l}\langle i, t, mid, tok, e, v\rangle \\ \langle i_r, t_r, mid_r, tok_r, \_, \_\rangle \\ \vdots \\ \langle i, t, mid, tok, e', v'\rangle \\ \vdots\end{array}} \rightarrow \langle i, t, mid, tok, true, \alpha(e), \{(i_r, t_r, mid_r)\}\rangle$$

$$\boxed{\alpha(S) = \bigvee_{(s,objs)\in S} \langle \hat{\alpha}_{Stack}(s), \alpha_s(\{objs\})\rangle}$$

Domaines abstraits : les graphes d'objets

$$
\begin{aligned}
Obj_a &= Oid_a \times Env_a \times Type \times Bool \\
Objects_a &= \mathcal{P}(Obj_a)
\end{aligned}
$$

Identifiant d'objets abstraits(Références abstraites)

On se donne le domaine Oid_a(**fini**) et une fonction

$$token \ : \ State \times Oid \rightarrow Oid_a$$

On sait calculer le token d'un objet abstrait sur une création d'objet. Deux

exemples :

- $token(o)$ rend la classe de l'objet concret o.
- $token(o)$ rend l'ensemble des variables qui réfèrent o

Abstraction des graphes d'objets

$$\boxed{e \sqsubseteq_e e' \text{ ssi } \forall id \in Id, e[id] \sqsubseteq_{val} e^{'}[id]}$$

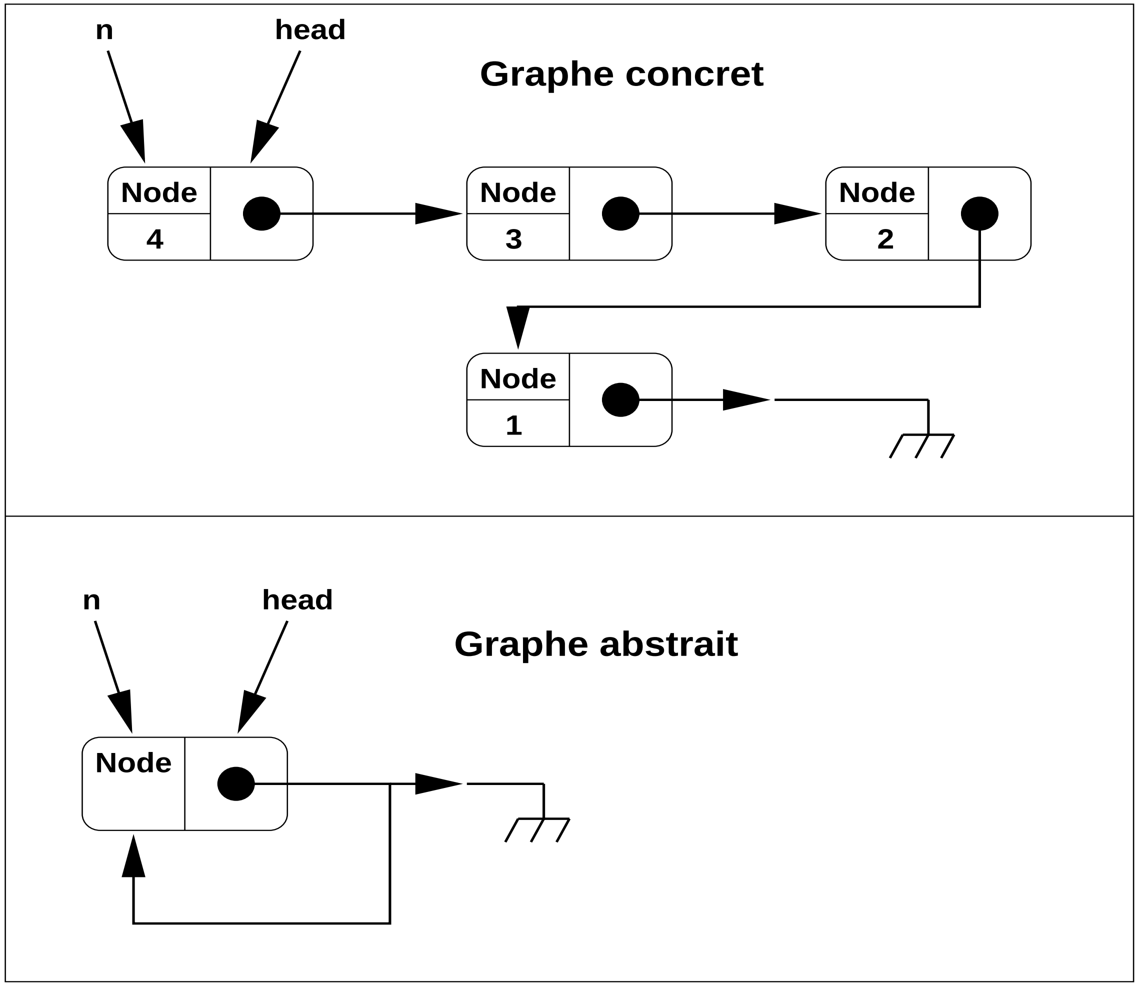
$$\boxed{\begin{array}{c} h \sqsubseteq_O h' \text{ ssi} \\ \forall \langle r, e, t, b, \rangle \in h, \exists \langle r, e^{'}, t, b^{'} \rangle \in h' \text{ tel que} \left\{ \begin{array}{l} e \sqsubseteq_e e' \\ b \Rightarrow b^{'} \end{array} \right. \end{array}}$$

$$\boxed{\alpha_S^O(O) = \bigvee_{objs \in O,\ \langle o,t,e,tok \rangle \in objs} \{\langle token(S,o), t, \alpha_e(e), tok, iss(o, objs)\rangle\}}$$

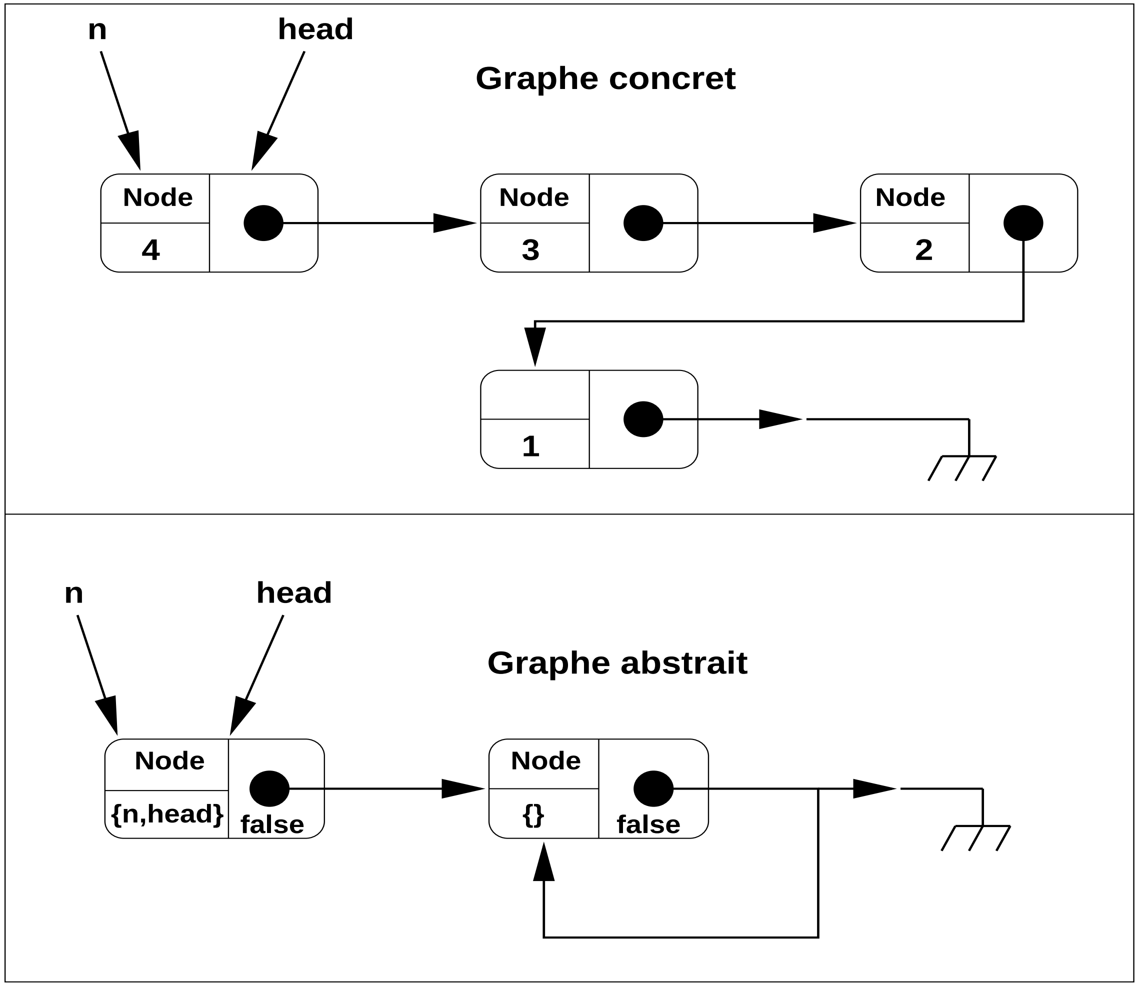
Exemple : Liste chainée

```
class ListInsert {
  public static void main(String[] args) {
    Node head,n;
    int i,j;
    head = new Node();
    i = 2;
    while (i < 5) {
      n = new Node();
      n.next = head;   head = n;
     i = i + 1;
    }
     n = head.next;   aux = n.next;
     head.next = aux;  n.next = head;
     head = n;
  }
}
class Node {
 Node next;
}
```

Abstraction des graphes d'objets : utilisation des classes



Abstraction des graphes d'objets : utilisation variables



Relation de transition abstraite

Relation de transition :

$$\boxed{State_a \to \mathcal{P}(State_a)}$$

Exemple :

```
if:
  cu, states
    |- aact(if(expr, Sttrue, Stfalse),...),objects
        ->
        astates[
          astate(aact(Sttrue,...), objects),
          astate(aact(Stfalse,...), objects)]
```

Affectations abstraites

Les formes d'affectations qu'on traite :

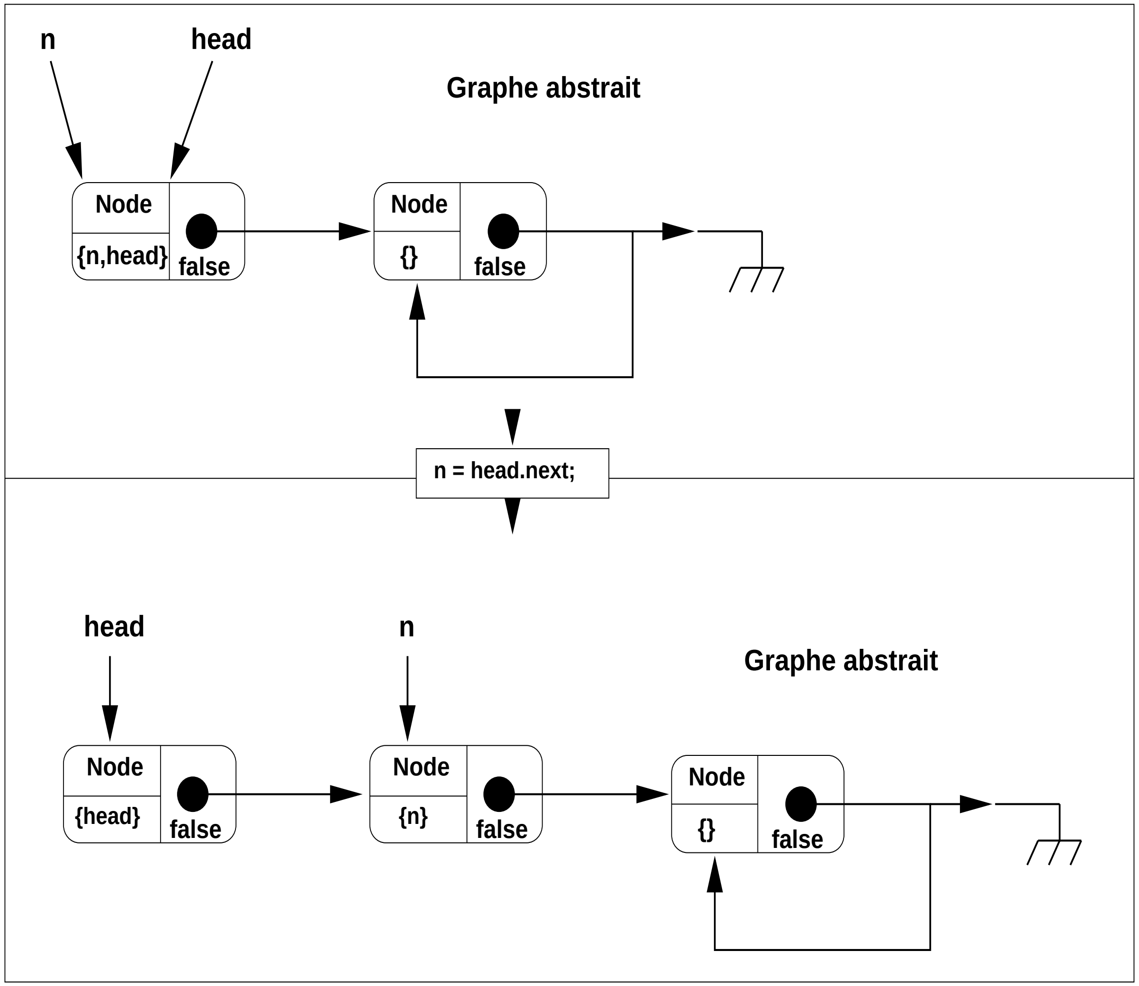
`x.sel=y`   `x=y.sel`   `x=y`

On peut toujours mettre un programme sous cette forme

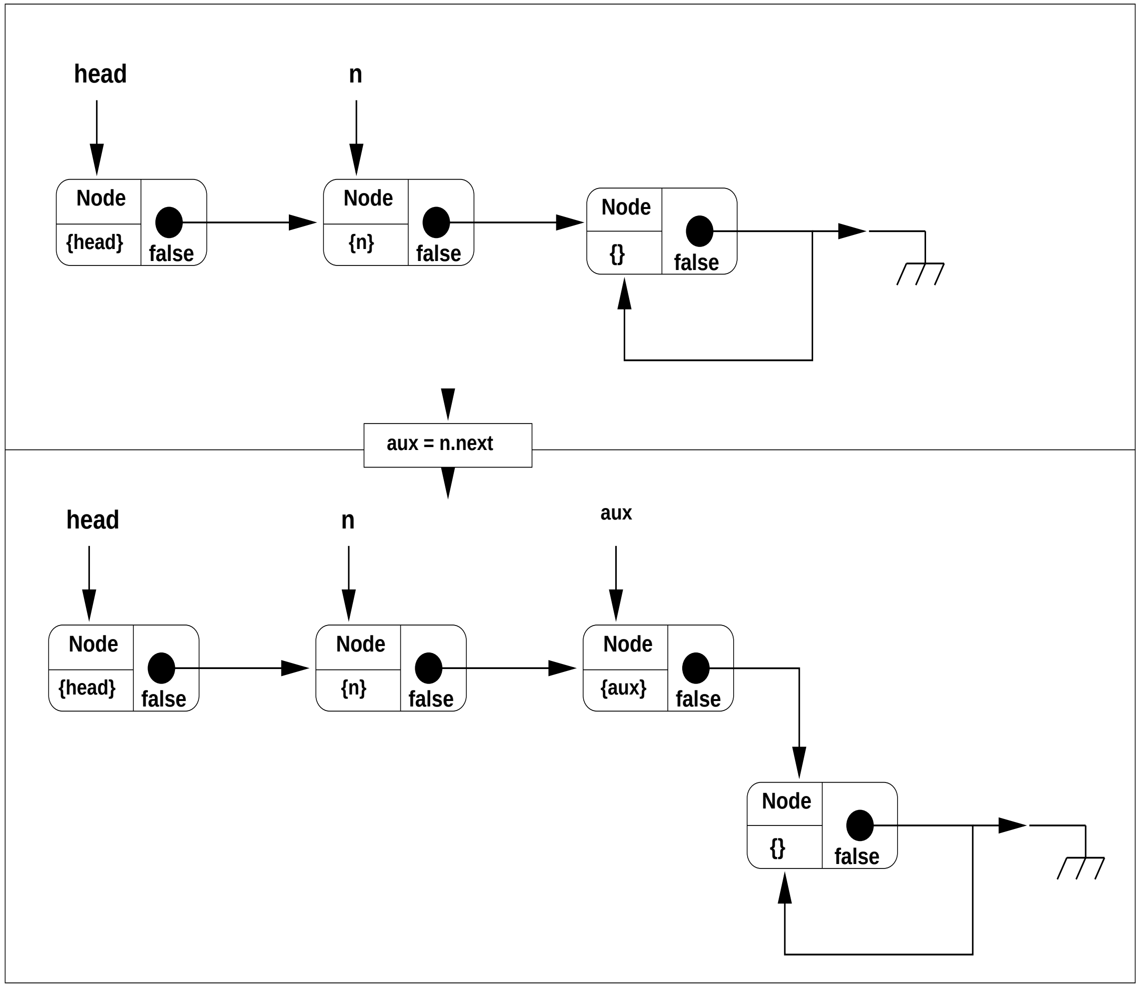
Augmentation de taille : fonction linéaire de la taille initiale.

Utilisation de **piles d'opérandes** : élimination des restrictions.

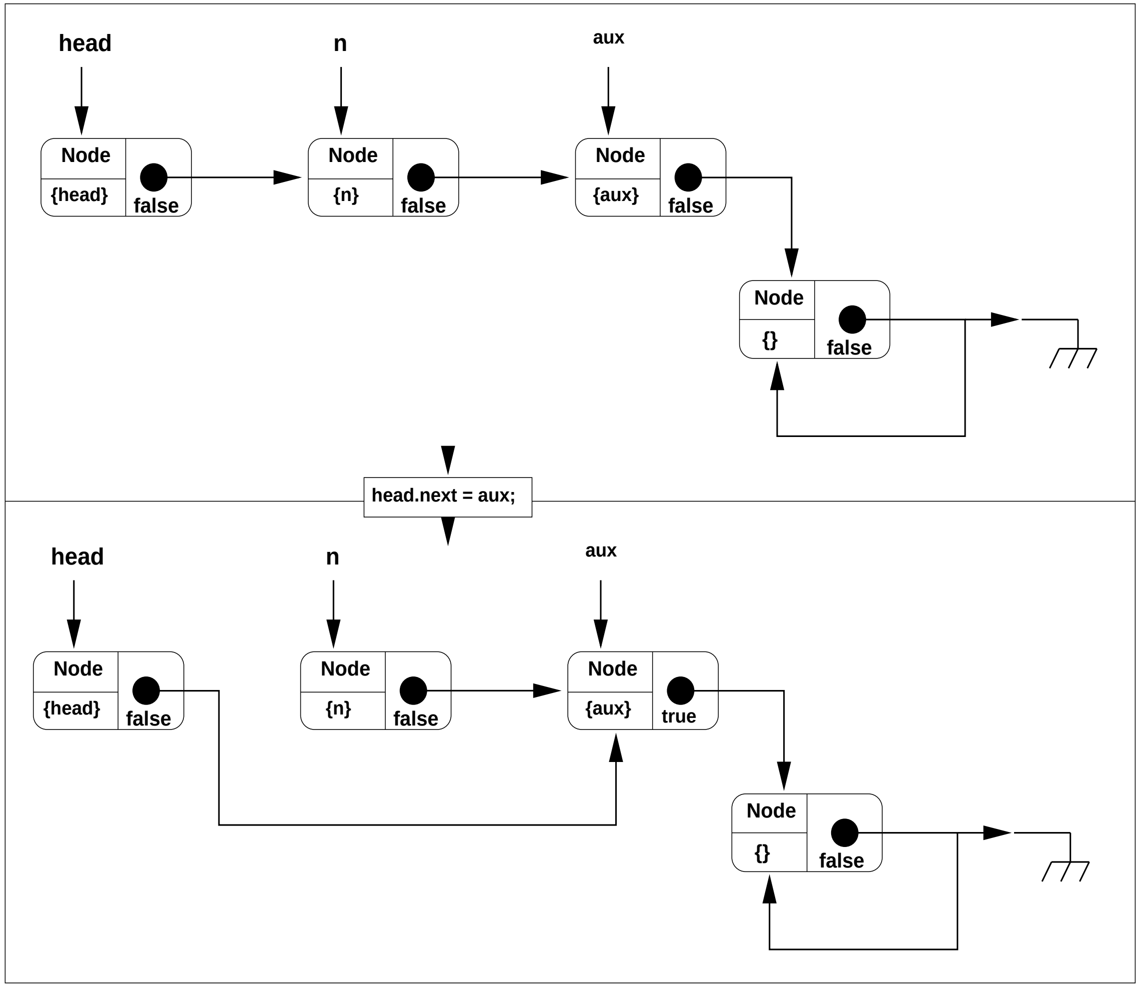
n = head.next


n
head
Graphe abstrait
Node
{n,head}
false
Node
{}
false
n = head.next;
head
n
Graphe abstrait
Node
{head}
false
Node
{n}
false
Node
{}
false

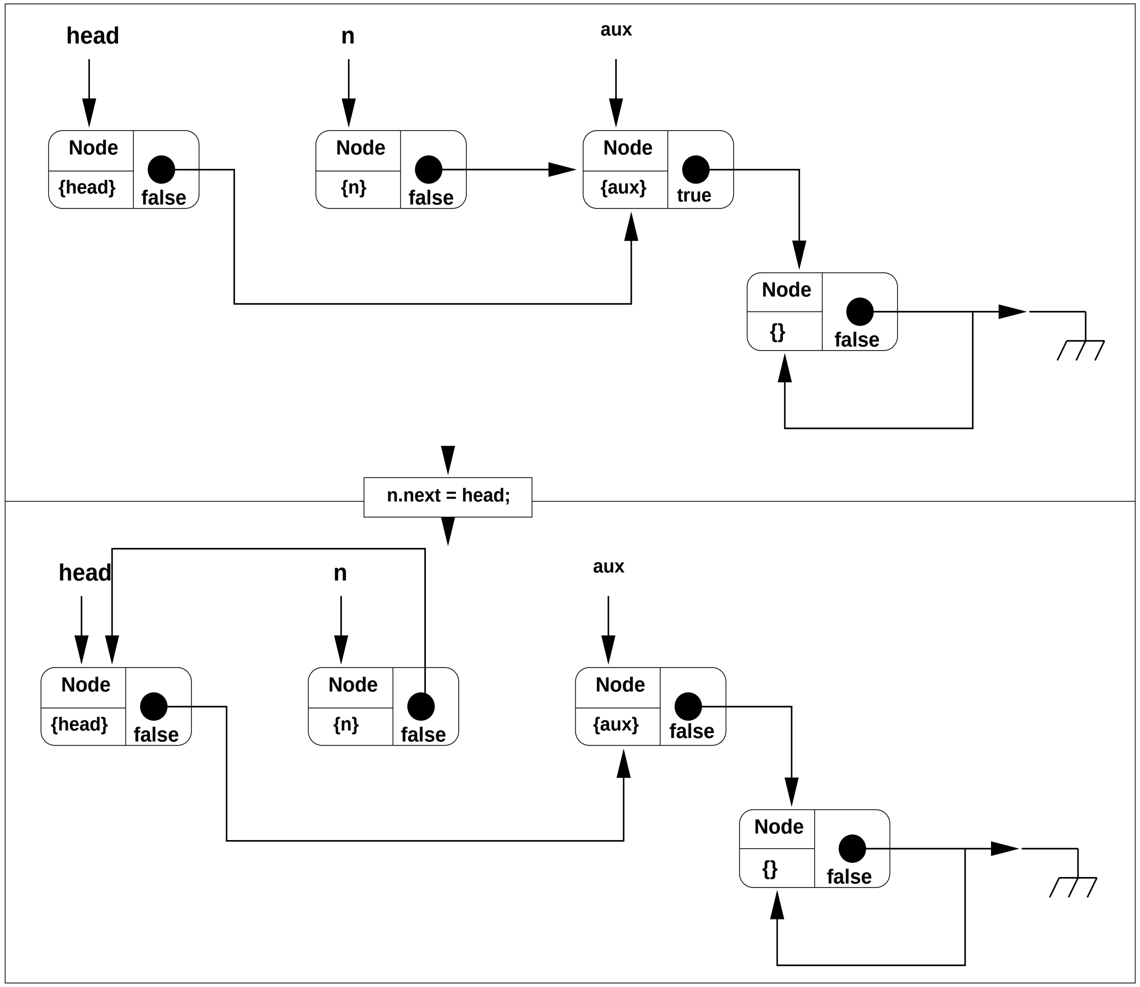
aux = n.next


head
n
Node
{head}
false
Node
{n}
false
Node
{}
false
aux = n.next
head
n
aux
Node
{head}
false
Node
{n}
false
Node
{aux}
false
Node
{}
false

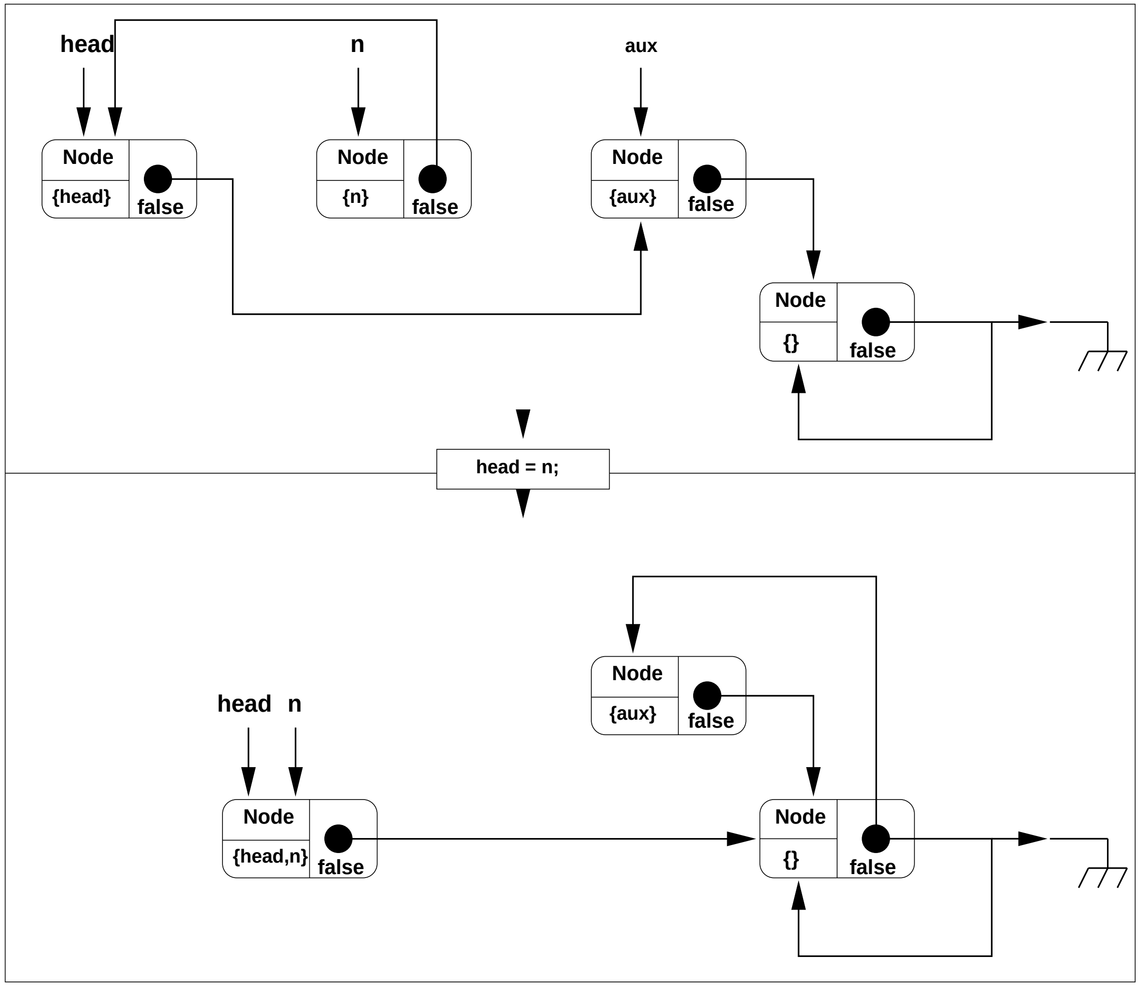
head.next = aux



n.next = head;


head
n
aux
Node
{head}
false
Node
{n}
false
Node
{aux}
true
Node
{}
false
n.next = head;
head
n
aux
Node
{head}
false
Node
{n}
false
Node
{aux}
false
Node
{}
false

head = n;



head
n
aux
Node
{head}
false
Node
{n}
false
Node
{aux}
false
Node
{}
false
head = n;
head n
Node
{aux}
false
Node
{head,n}
false
Node
{}
false

Résultats

Positifs :

- Liste chainées (insertion)
- Arbres binaires de recherche (insertion)

Par contre :

- toute structure **non-arborescente** est mal représentée.
- les arbres binaires ne résistent pas à la suppréssion.

Applications envisageables

- outils d'aide à la parallélisation
- compilation : *compile-time garbage-collection*, ...
- divers outils d'aide à la programmation.

Travaux connexes

- Analyses de forme des structures dynamiques.(Sagiv et al. 96)
- Analyse dataflow de programmes à objets(Chambers et al.97).
- Interprétation abstraite de Pascal (Bourdoncle)