

Interprétation Abstraite

Paul Feautrier

`Paul.Feautrier@ens-lyon.fr.`

ENS Lyon

Plan

- Concepts de Base
- Exemple : signe des variables
- Le treillis des signes
- Interprétation abstraite
- Exemple : la propagation des constantes
- Calcul du point fixe
- Equations de flot
- Exécution symbolique

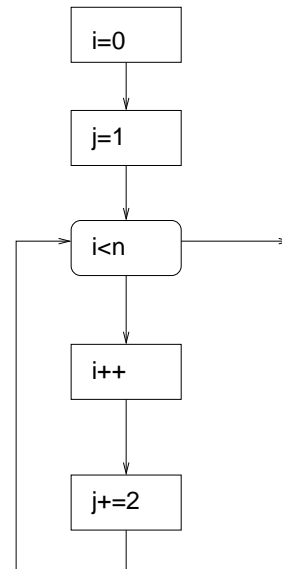
Bibliographie

- Les oeuvres de Patrick Cousot
- Muchnick et Jones : Program Flow Analysis
- Muchnick : Advanced Compiler Design and Implementation, Chap. 8

Concepts de base, I

- Graphe du flot de contrôle : Un graphe dont les sommets sont les instructions du programme. Il y a un arc entre S et T s'il est possible que T suive directement S dans une exécution du programme.

```
i=0 ;  
j=1 ;  
while(i<n) {  
    i++;  
    j+=2 ;  
}
```



- **Point de programme** : un arc du graphe de contrôle. Ce n'est que sur un arc (entre deux instructions) que l'on peut observer l'état de la mémoire.

Concepts de base, II

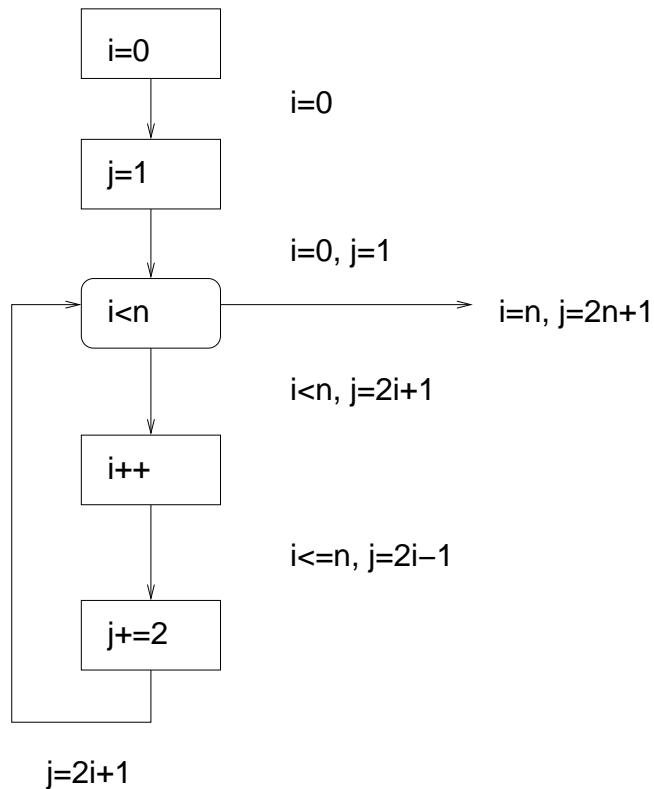
Propriétés. Tout prédicat pouvant être vérifié ou non dans un état de la mémoire.

Soit i , z , j , tab , x des variables du programme. Exemples de propriétés :

- x défini,
- $z == 12$,
- $i \geq 0$,
- $i == 2j + 1$,
- tab alloué.

On s'intéresse en général aux propriétés qui sont vraies chaque fois que le contrôle passe par un certain point de programme. On dit que la propriété est attachée au point de programme.

Exemple de propriétés

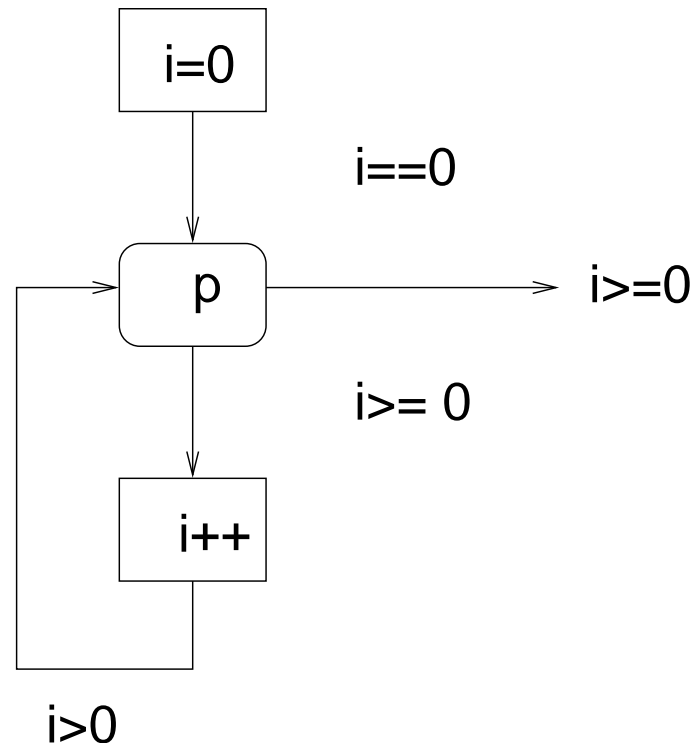


- Ici, les propriétés se démontrent par induction sur l'exécution du programme.
- La propriété $j = 2i + 1$ est vraie juste après les initialisations.
- Il suffit ensuite de suivre l'effet des affectations.
- Enfin, la propriété est vraie après le test, puisque celui-ci n'a pas d'effet de bord.
- Le problème est d'automatiser ce raisonnement.

Un exemple

- Déterminer le signe des variables d'un programme.

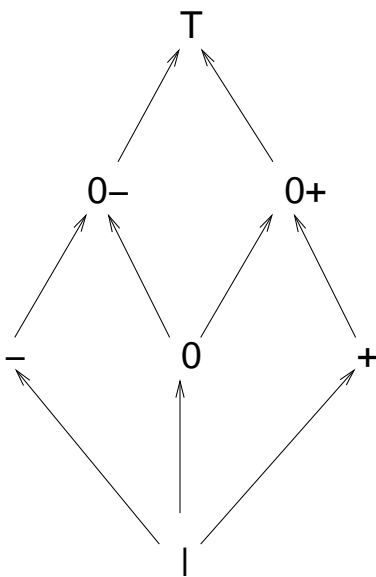
```
i=0;  
while(p) {  
    i++;  
}
```



- Comment automatiser la détermination des signes ?

Le treillis des signes

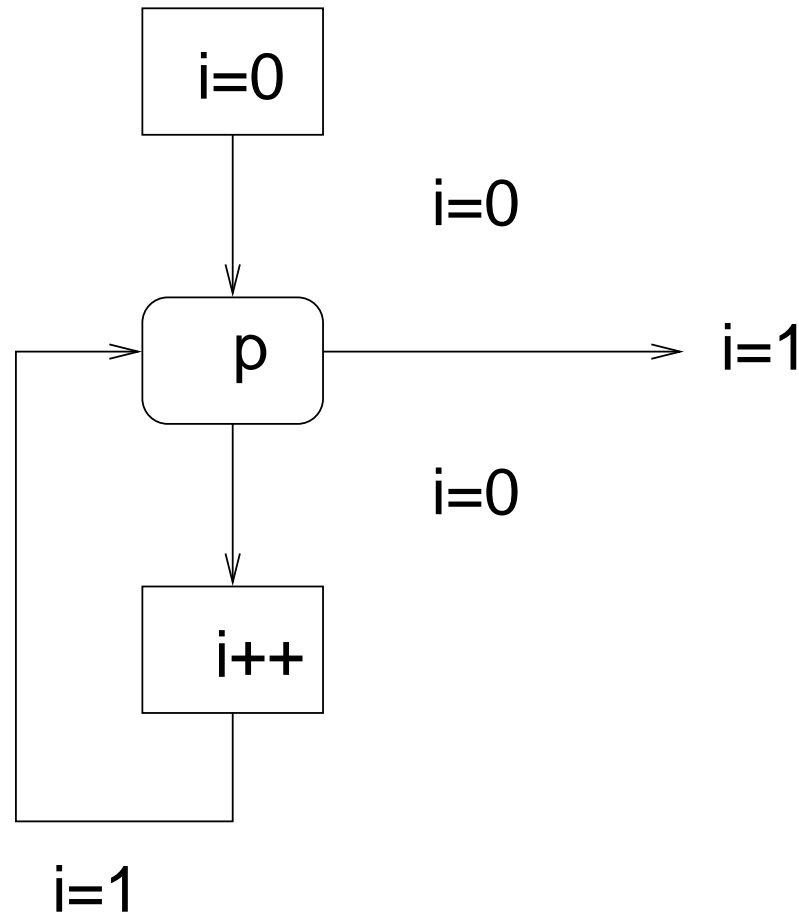
- On considère les propriétés $x < 0$, $x \leq 0$, $x = 0$, $x \geq 0$, $x > 0$, que l'on dénote $-$, $0-$, 0 , $0+$ et $+$.
- Chaque propriété est en fait un sous-ensemble de \mathbf{IR} . Ceux-ci forment un treillis.



- Le signe \perp est le point minimum du treillis. Il représente l'ensemble vide.
- De même, \top désigne \mathbf{IR} tout entier.
- Par exemple, la borne supérieure de 0 et de $+$ est $0+$, l'ensemble des nombres non-négatifs.

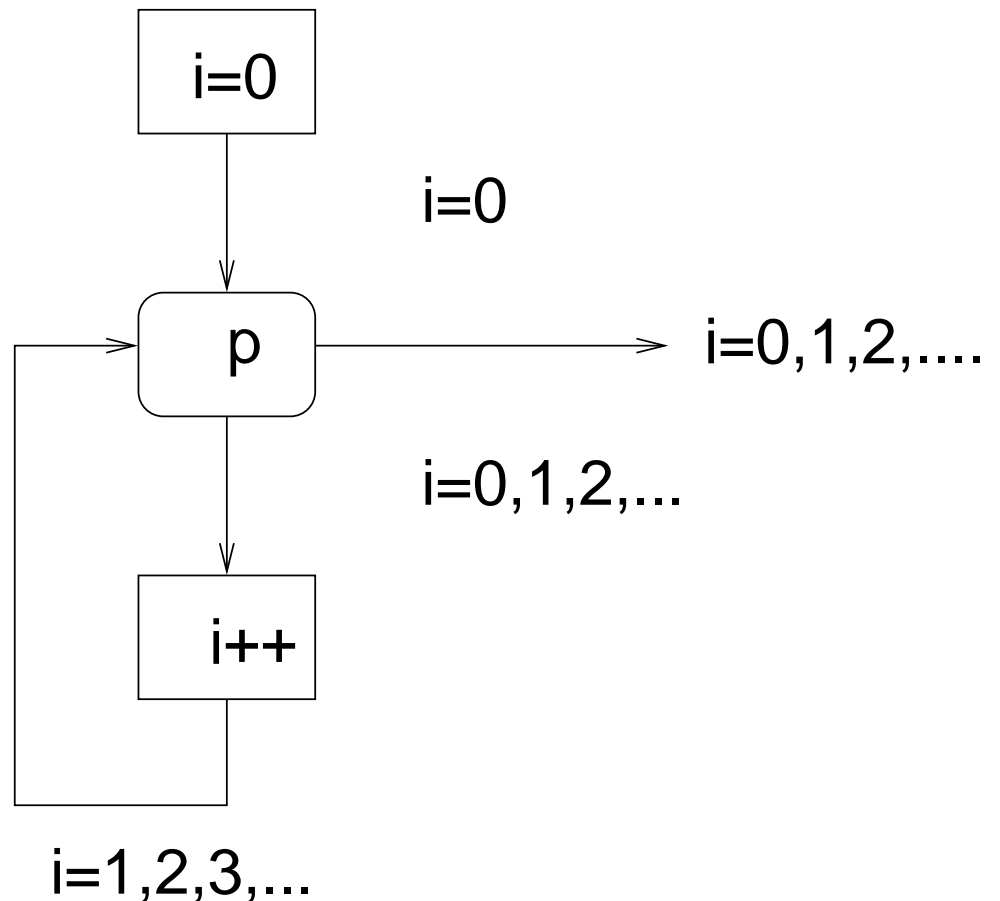
Interprétation, I

- Un interprète est un programme qui simule une exécution du programme source, par exemple à partir des informations contenues dans la RI, ou mieux à partir du graphe de contrôle.



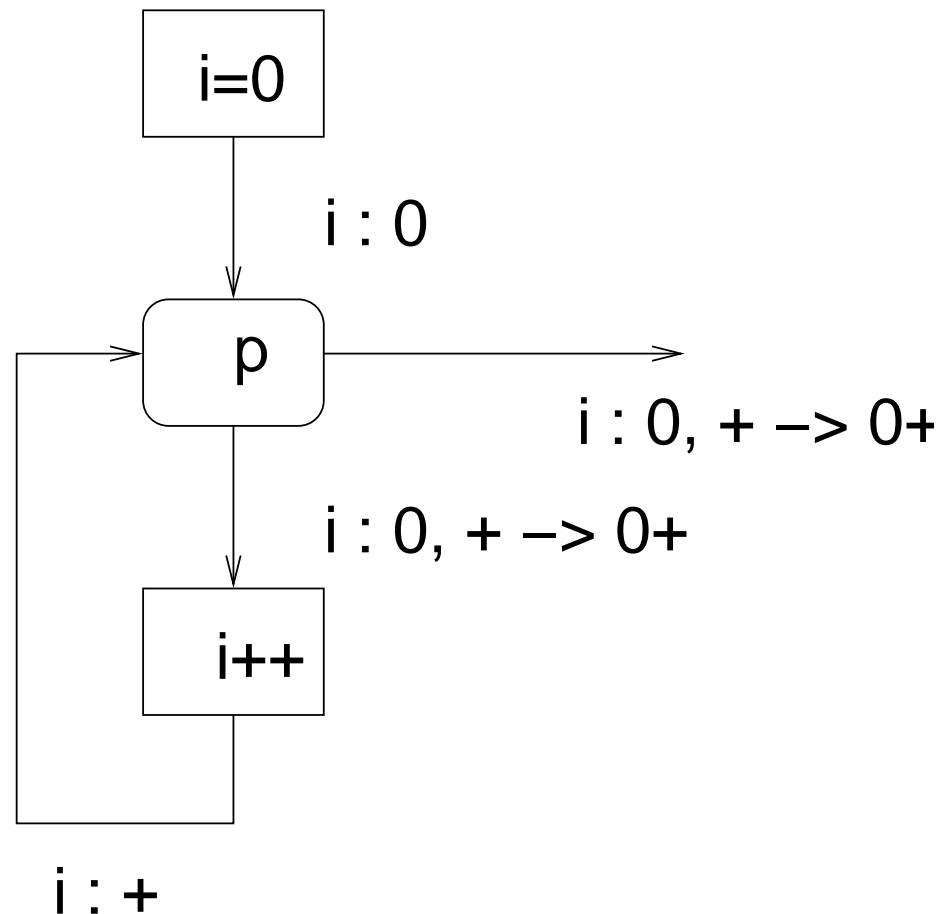
Interprétation, II

- Si l'on veut calculer des propriétés universelles, il faut conserver toutes les valeurs prises par les variables.
- En l'absence d'autres informations, il faut suivre toutes les branches du programme.

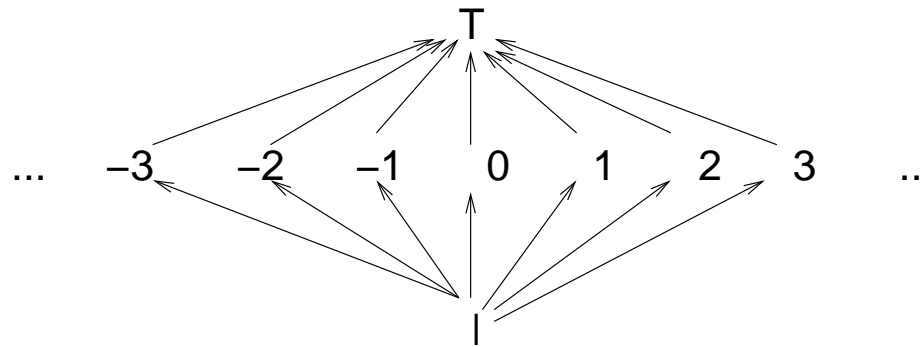


Interprétation abstraite

- On remplace les ensembles en extension par des ensembles en intention.
- On obtient un point fixe en trois itérations.



Propagation des constantes

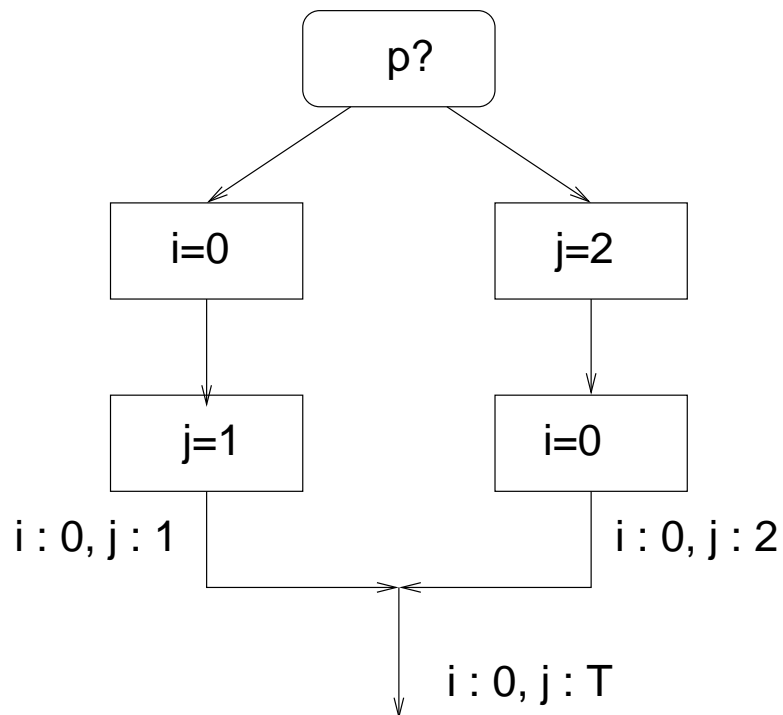


● Le treillis :

● Les règles de calcul :

- Si les arguments d'une fonction sont des constantes, effectuer le calcul, sauf si la fonction est trop complexe ou inconnue (fonction de bibliothèque).
- Si une variable peut prendre deux valeurs différentes, lui donner la valeur T.
- Si une fonction a un argument T, son résultat est T.

Traitement des tests



- Les valeurs des variables non affectées sont simplement transmises.
- En l'absence d'autres informations, on doit prendre indifféremment les deux branches du test.
- Après la jonction, une variable prend pour valeur l'union des valeurs avant la jonction, ou encore, la borne supérieure des deux valeurs dans le treillis.

Questions de convergence

- Quand doit on sortir d'une boucle ?
- On doit attendre que les valeurs abstraites des variables ne changent plus (point fixe).
- Pour être sûr d'atteindre un point fixe, il faut imposer des contraintes aux instructions abstraites et au treillis des valeurs abstraites.
- Les instructions abstraites doivent être des fonctions monotones croissantes :

$$x < y \Rightarrow f(x) < f(y)$$

au sens de l'ordre du treillis.

- Le treillis doit être de hauteur finie. L'existence d'un point fixe est alors garantie.
- Les deux exemples vus jusqu'ici ont les bonnes propriétés.
- Quel est le treillis à utiliser pour vérifier l'allocation des tableaux ?

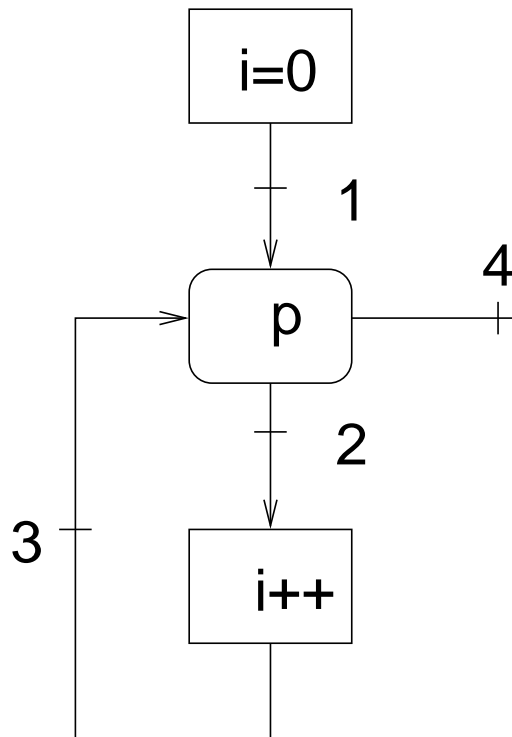
Equations de flot

- L'écriture d'un interprète abstrait est délicate, en particulier si l'on veut être sûr de ne pas oublier de chemin en présence de tests.
- On peut faire mieux en écrivant des « équations de flot » qui expriment les relations qui existent entre les valeurs des variables en un point de programme et les valeurs en ses prédécésseurs (ou ses successeurs).
- On résoud ensuite le système obtenu, soit par une méthode itérative, soit par une méthode directe.

Exemple

- On note i_k la valeur abstraite de i au point k .
- En un point de jonction, la valeur de i est l'union des valeurs en ses prédécesseurs :

$$i_2 = i_1 \cup i_3, \quad i_4 = i_1 \cup i_3$$



- Une instruction d'affectation détruit la valeur précédente de la variable affectée et lui donne une nouvelle valeur :

$$i_1 = 0, \quad i_3 = +$$

- La résolution itérative de ce système à partir de $i_1 = i_2 = i_3 = i_4 = \perp$ donne :

$$i_1 = 0, \quad i_2 = 0+, \quad i_3 = +, \quad i_4 = 0+$$

Construction des équations de flot

- En tout point du programme, on possède des informations sur toutes les variables du programme (du moins toutes les variables « intéressantes »).
- Après une convergence, les informations sont l'union des informations des branches convergentes : il y a en général perte d'information.
- Une instruction d'affectation détruit les informations relatives à la variable affectée et leur en substitue d'autres, obtenues par exécution abstraite de son membre gauche.
- Pour chaque opérateur du langage, il faut construire une table de Pythagore dans le treillis utilisé.

Exemple

La table de $+$ dans le treillis des signes.

	\perp	-	0-	0	0+	+
\perp	\perp	-	0-	0	0+	+
-	-	-	-	-	T	T
0-	0-	-	0-	0-	T	T
0	0-	-	0-	0	0+	+
0+	0+	T	T	0+	0+	+
+	+	T	T	+	+	+

Méthodes de résolution

- Principes : la résolution est itérative. Au départ, la valeur de chaque variable est \perp . A chaque itération, et en chaque point, on calcule la valeur de chaque variable comme fonctions de l'état à l'itération précédente (itération de Jacobi). On s'arrête quand plus aucune valeur ne change.
- Il existe toutes sortes d'optimisations :
 - Si à l'itération k le point i dépend du point j qui vient d'être mis à jour, on peut utiliser la valeur à l'itération k au lieu de la valeur à l'itération $k - 1$ (itération de Gauss-Seidel).
 - On peut éliminer les variables inutiles (par exemple les variables réelles si l'on ne s'intéresse qu'aux indices de tableaux) et les points inutiles (qui ne font que transmettre les valeurs du point précédent).
 - On peut organiser les itérations pour tenir compte de la structure du programme. Exemple : deux nids de boucles en séquence.

Bloc de base et DAG

- Définition : bloc de base : un ensemble de points de programme tel que tout point n'a qu'un seul prédécesseur (sauf le premier) et qu'un seul successeur (sauf le dernier).
- Il est impossible d'exécuter partiellement un bloc de base.
- Si l'on exécute la première instruction d'un bloc de base, alors on les exécute toutes jusqu'à la dernière.
- Définition DAG : une portion de programme dont le graphe de contrôle est acyclique et qui n'a qu'un point d'entrée et un point de sortie.
- Aussi bien dans un DAG que dans un bloc de base, il n'est pas besoin d'itérer pour appliquer l'interprétation abstraite.
- Il est fréquent que l'on traite un bloc de base comme une instruction unique, en regroupant les effets de ses composantes.

Exécution symbolique, I

- Comment trouver des relations entre variables (voir exemple I) ?
- On peut construire le treillis des relations (mais il faut se limiter au treillis des relations affines : $i = 2j, i + j \leq k$, etc.).
- Mais ce treillis n'est pas de hauteur finie ; il faut utiliser des accélérateurs de convergence (méthode de Cousot-Halbwachs) et l'algorithme devient complexe.
- On peut utiliser une méthode plus simple, l'exécution symbolique, qui s'applique bien dans les blocs de base, et qui peut être étendue aux boucles dans les cas favorables.

Exécution symbolique, II

- Le treillis utilisé est celui des expressions algébriques que l'on sait manipuler : par exemple, le treillis plat des expressions linéaires.
- Dans un bloc de base, on sait évaluer les valeurs des variables par simple propagation :

```
{
  i = i+1;
  (*          i = i_ + 1 *)
  j = 3;
  (*          i = j_ + 1, j = 3 *)
  k = i + j;
  (*          i = j_ + 1, j = 3, k = i_ + 4 *)
}
```

Exécution symbolique, III

- Traitement des tests.
- Première solution : on reste dans le treillis plat. A la jonction, si une variable a la même valeur sur les deux branches, elle la conserve, et sinon elle passe à la valeur \top .
- Deuxième solution : on agrandit le treillis pour traiter les expressions conditionnelles à prédicat linéaire. C'est compliqué et d'une grande complexité, sans apporter des avantages significatifs.

Exécution symbolique, IV

- Traitement des boucles.
- Première solution : on ne fait rien. A la sortie d'une boucle, les variables sont indéfinies. Peut être suffisant.
- Sinon, le calcul du point fixe passe par la résolution d'une équation de récurrence.

Exemple

```
j = 3;  
for(i = 0; i < n; i++) {  
    j += 2;  
    ...  
    k = j - 1;  
    ...  
    j = k;  
}
```

$$\begin{aligned}j &= \bar{j} + 2 \\k &= \bar{j} + 1 \\j &= \bar{j} + 1 \\j_{i+1} &= j_i + 1 \\j_i &= i + 3\end{aligned}$$