Les tests logiciels : séance 2

M.Madani

ISIL - HEPL

2015-2016

Control flow testing

- En quelques mots
  - Technique en boîte blanche donc basée sur l'analyse du code.
  - Principalement utilisée pour les tests unitaires.
  - Applicable pour la plupart des programmes mais pour des portions relativement petites de code.
- Hypothèses
  - Une ligne de code non testée diminue la probabilité de trouver une faute.
  - Une instruction ou un prédicat erroné peut conduire à un changement dans le chemin d'exécution et le tester augmente la probabilité d'observer un dysfonctionnement.
- Limitations
  - Ne mets pas en évidence des exigences non formulées.
  - Se focalise sur un niveau à la fois(pas les niveaux plus bas).
  - Un programme avec des fautes peut donner au travers des tests des résultats corrects par coïncidence.

Les principales étapes

- Les entrées
  - Le programme.
  - Le critère d'adéquation pour T.
- Les étapes
  1. Choisir un critère pour l'adéquation de la suite de tests.
  2. Construire le graphe de contrôle du programme(coûteux).
  3. Calculer des chemins et des conditions additionnelles jusqu'à remplir le critère fixé.
  4. Calculer les données en entrées pour forcer les chemins(faisables) choisis et remplir les conditions.
- En pratique, on instrumentalise le code pour
  - vérifier que les chemins choisis correspondent aux chemins empruntés ;
  - atteindre de manière incrémentale le critère fixé(les chemins peuvent ne pas avoir été déterminé à l'avance).

Un bloc de base

Un bloc est la plus longue suite d'instructions consécutives dans un programme de telle sorte que le contrôle ne peut démarrer qu'à la première instruction et ne termine obligatoirement qu'à la dernière.

- Il y a un point d'entrée et un point de sortie.
- Si il n'y a qu'une seule expression, le point d'entrée et de sortie coïncide.
- Il ne peut y avoir de conditions dans un tel bloc, sauf pour la dernière instruction.

Un bloc est la plus longue suite d'instructions consécutives dans un programme de telle sorte que le contrôle ne peut démarrer qu'à la première instruction et ne termine obligatoirement qu'à la dernière.

- Il y a un point d'entrée et un point de sortie.
- Si il n'y a qu'une seule expression, le point d'entrée et de sortie coïncide.
- Il ne peut y avoir de conditions dans un tel bloc, sauf pour la dernière instruction.

Pour un appel de fonction, on peut décider de le mettre dans un bloc séparé ou non. On le met dans un bloc séparé si on veut considérer qu'il s'opère un transfert de contrôle vers la fonction. Une fonction peut aussi déroger à la règle selon laquelle le flux de contrôle ne peut s'arrêter avant le point de sortie : considérez par exemple le lancement d'une exception dans le corps de la fonction qui permet de se brancher bien plus loin dans le code... Enfin, une fonction contient probablement plusieurs chemins d'exécutions. Il peut être décidé dans un premier temps de ne pas en tenir compte et de considérer la fonction comme une instruction simple.

La recherche dichotomique

```
 1 int
 2 rechdicho (int t[], int length, int e)
 3 {
 4   int n = length - 1;
 5   int bas = 0, haut = n, milieu;
 6   int i = -1;
 7   do
 8     {
 9       milieu = (bas + haut) / 2;
10       if (e == t[milieu])
11         i = milieu;
12       else if (t[milieu] < e)
13         bas = milieu + 1;
14       else
15         haut = milieu - 1;
16     }
17   while ((e != t[milieu]) && (bas <= haut));
18   return i;
19 }
```

Inventaire des blocs

Bloc	Ligne(s)	Ligne d'entrée	Ligne de Sortie
1	4,5,6	4	6
2	9,10	9	10
3	11	11	11
4	12	12	12
5	13	13	13
6	15	15	15
7	17	17	17
8	18	18	18

Bloc	Ligne(s)	Ligne d'entrée	Ligne de Sortie
1	4,5,6	4	6
2	9,10	9	10
3	11	11	11
4	12	12	12
5	13	13	13
6	15	15	15
7	17	17	17
8	18	18	18

Ici, un choix doit être opéré. Soit on considère l'alternative comme un tout soit on décide de la décomposer.

```
1  int a = 0, b = 1, c;
2  if (((c = getvalue(a)) !=
        b) && (b--))
3    //do ...
```

```
1  c = getvalue (a);
2    if (c != b)
3      {
4        if (b != 0)
5          {
6            b = b - 1;
7            //do ...
8          }
9        else
10         {
11           b = b - 1;
12         }
13     }
```

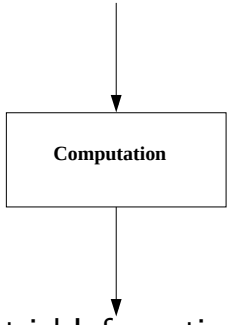
Graphe de contrôle

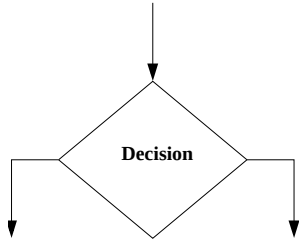
Un graphe de contrôle, noté $G(N, E)$ est défini par un ensemble fini de N noeuds et un ensemble fini E d'arêtes ou arcs dirigées.

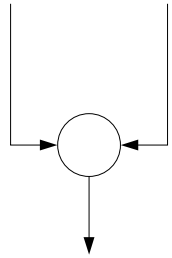
- Chaque noeud représente un bloc de base.
- Chaque arc (i,j) de E connecte les noeuds n_i et n_j de N et est représenté par une flêche.
- Il existe deux noeuds particuliers : le noeud de départ et le noeud de fin.
- Au départ du noeud de départ, il existe un chemin vers chaque noeud.
- A partir de chaque noeud, il existe un chemin vers le noeud de fin.
- Il n'y a aucun arc qui arrive au noeud de début et aucune qui repart du noeud final.

Construire un CFG

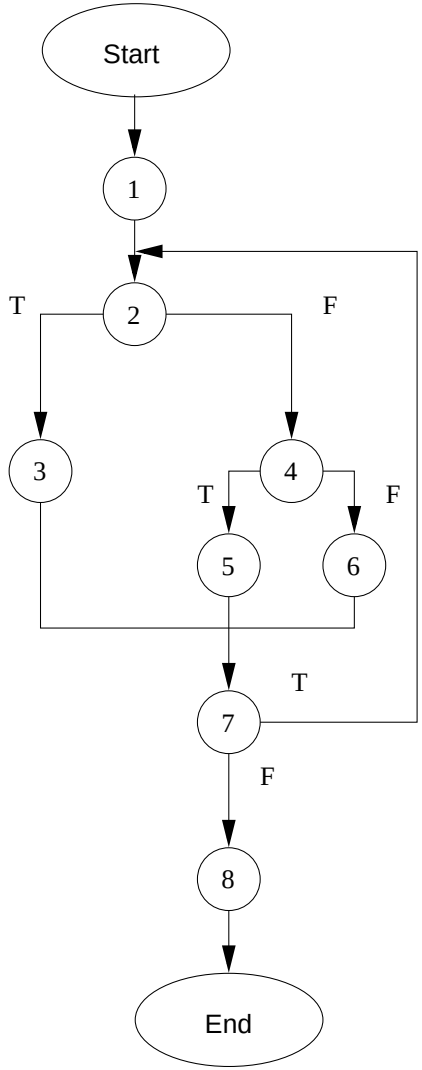
- Un graphe de contrôle(control flow graph) consiste à construire une représentation graphique d'un partie de code(une fonction par exemple).
  - Chaque bloc/instruction séquentielle est numérotée et représentée par un rectangle.
  - Une décision est représentée par un losange dont chaque branche de sortie est labelisée par T ou F.
  - La fusion de deux branches est représentée par un cercle sur lequelle deux branches arrivent et une en repart vers l'élément suivant.
- Alternative : le même symbole pour tout, pas de symbole pour fusion et n'est conservé que ce qui est nécessaire.

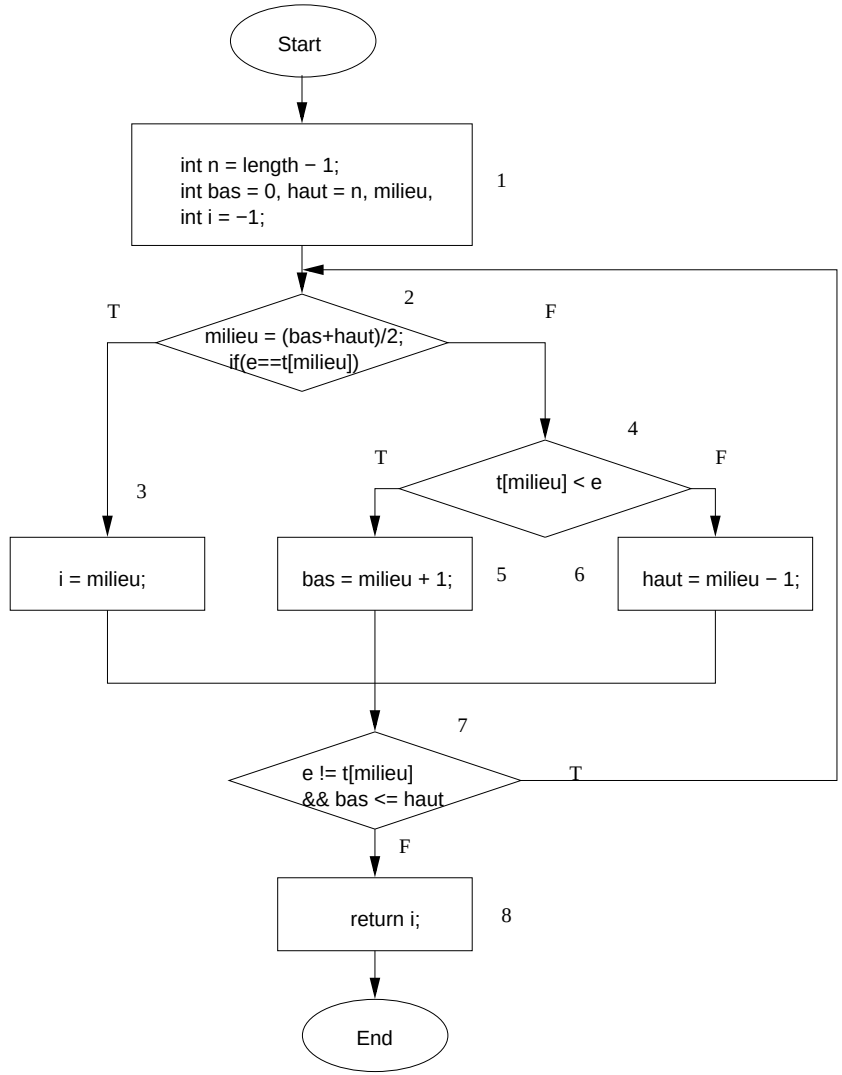






CFG pour la recherche dichotomique - slide 5





Chemins d'exécutions

- Un chemin est une séquence d'arcs du CFG $G = (N, E)$.
  - Un arc est une paire de noeuds $e_i = (n_p, n_q)$.
  - Les arcs sont numérotés : si n_p,n_q,n_r,n_s sont des noeuds et $k > 0$ arcs, $e_i = (n_p, n_q)$ et $e_{i+1} = (n_r, n_s)$ alors on a $n_q = n_r$.
- Il peut exister des chemins qui ne sont pas faisables$\Rightarrow$ pas possible de dériver des entrées pour ces chemins.
  - Le problème est indécidable.
- Exemples :
  - Chemin non valide : ((1,2),(3,7),(7,2)) .
  - Chemin valide :((Start,1),(1,2),(2,3),(3,7)).
- Alternative :représenter les chemins par les noeuds.
  - Chemin valide :(Start,1,2,3,7).

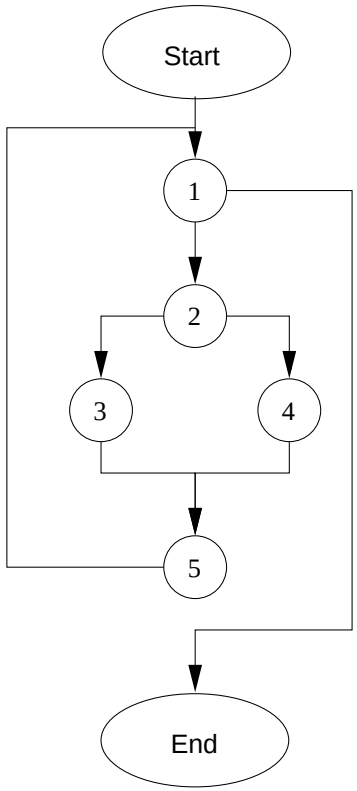
Vecteur chemin

- $G = (N, E)$
  - $N = \mathit{Start}, 1, 2, 3, 4, 5, 6, 7, 8, \mathit{End}$
  - $E = \{(\mathit{Start}, 1), (1, 2), (2, 3), (2, 4), (3, 7), (4, 5), (4, 6), (5, 7), (6, 7), (7, 2), (7, 8), (8, \mathit{End})\}$
- Chaque entrée du vecteur correspond à un arc et au nombre de fois qu'il est traversé ;
  - Longueur $l = |E|$;
  - 0 $\rightarrow$ l'arc n'appartient pas au chemin ;
  - 1 $\rightarrow$ l'arc n'appartient pas au chemin ;
- Exemples ($l = 12$)
  - Chemin complet (Start,1,2,3,7,8,End) $\rightarrow$ $V_{p1} = [111010000011]$.
  - Chemin complet pas faisable (Start,1,2,3,7,2,3,7,8,End) $\rightarrow$ $V_{p2} = [112020000111]$.

Chemins indépendants

- Soit P et G son graphe de contrôle. Il est toujours possible de calculer un ensemble de chemin(s) complet(s) et linéairement indépendant(s) de G.
  - Le nombre de chemins maximum indépendants $V(G) =$ complexité cyclomatique de G.
  - L'ensemble des combinaisons linéaires de ces $V(G)$chemins couvre l'ensemble des chemins de G.
- Rappel indépendance linéaire :
  $\sum_{i=1}^{n} \lambda_i x_i = 0 \Rightarrow \forall i \in \{1, ..., n\} \lambda_i = 0$
- $V_p = a_1 V_1 + a_2 V_2 + a_3 V_3 + ... + a_n V_n$ avec a_j $1 \leq j \leq n$ sont des entiers.
  - p est un chemin (par exemple pas de base) ;
  - S l'ensemble des chemins($p \in S$) ;
  - $V_1, V_2, V_3, ..., V_n$ une base de G ;

Chemins indépendants d'un G



- Chemins indépendants
  - $p_1 = (Start, 1, 2, 3, 5, 1, End)$.
  - $p_2 = (Start, 1, 2, 4, 5, 1, End)$.
  - $p_3 = (Start, 1, End)$.
- $p_4 = (Start, 1, 2, 3, 5, 1, 2, 4, 5, 1, End) = p_1 + p_2 - p_3$.

Calcul d'une base de G

1. Prendre un chemin complet p_{ref} quelconque de G. Ce chemin ne passe dans les boucles qu'une fois au plus.
2. Jusqu'à avoir $V(G)$ chemins
   - 2.1 Calculer nouveau chemin p_i en changeant la sortie d'une alternative dans p_{ref}
   - 2.2 Si p_i n'est pas encore chemin dans les chemins de base, alors l'ajouter.

Complexité cyclomatique [McCabe 1976]

- La complexité cyclomatique V(G) d'un graphe de contrôle

$$V(G) = E - N + 2$$

.

  - E est le nombre d'arcs du graphe.
  - N est le nombre de noeuds.

- $V(G) =$ nombre de chemins linérairement indépendants.
- $V(G)$ est souvent utilisé comme une mesure de la complexité, bien que la caractérisation précise et directe de celle-ci soit toujours l'objet de recherches.
- $V(G) = 10$ est une valeur seuil généralement acceptée qui indique si on la dépasse qu'il faut repenser le module, la fonction, ...
- $V(G) = D + 1$ pour D décisions binaires.

- En théorie des graphes, $v(G) = e - n + p$ est la complexité cyclomatique d'un graphe fortement connexe et est égale au nombre maximum de cycles linéairement indépendants.
- Un graphe de contrôle est fortement connexe si on ajoute un arc virtuel entre le noeud de sortie et le noeud d'entrée.

Complexité cyclomatique [McCabe 1976]

- La complexité cyclomatique V(G) d'un graphe de contrôle
$$V(G) = E - N + 2$$
.
  - E est le nombre d'arcs du graphe.
  - N est le nombre de noeuds.
- $V(G) =$ nombre de chemins linérairement indépendants.
- $V(G)$ est souvent utilisé comme une mesure de la complexité, bien que la caractérisation précise et directe de celle-ci soit toujours l'objet de recherches.
- $V(G) = 10$ est une valeur seuil généralement acceptée qui indique si on la dépasse qu'il faut repenser le module, la fonction, ...
- $V(G) = D + 1$ pour D décisions binaires.

Statement and block Coverage Criterion

- L'adéquation d'un T est mesurée sur un domaine de couverture S_e qui est l'ensemble des instructions du programme.

$$0 \leq \frac{|S_c|}{|S_e|-|S_i|} \leq 1$$

.

  - S_c les instructions couvertes par T.
  - S_e les instructions du programme.
  - S_i les instructions du programme que l'on ne sait pas atteindre.

- Alternative : le critère l'adéquation est sur un domaine de couverture relatif au blocs.

Statement Coverage

1. $T = \left\{ \begin{array}{l} t_1 =< t = [1,2,3,4], \mathit{length} = 4, e = 2 > \\ t_2 =< t = [1,2,3,4], \mathit{length} = 4, e = 1 > \end{array} \right\}$
   - La ligne 13 du listing slide 5 n'est jamais exécutée.
   - $SC = 10/11$ et $BC = 7/8 \Rightarrow$ plus de tests sont requis.
2. $T_2 = \left\{ \begin{array}{l} t_1 =< t = [1,2,3,4], \mathit{length} = 4, e = 2 > \\ t_2 =< t = [1,2,3,4], \mathit{length} = 4, e = 1 > \\ t_3 =< t = [1,2,3,4], \mathit{length} = 4, e = 3 > \end{array} \right\}$

Decision Coverage Criterion

- Les éléments usuels des langages pour les décisions sont `if`,`switch`,`while`, ...
- On cherche à savoir si une décision est correctement formulée et située au bon endroit.
- Une décision est établie sur base d'une ou plusieurs conditions.
- Exemples
  - `if ((a && b)||(a && c))...`
  - `x = p || q;`
- Une décision est dite couverte si chaque sortie possible a été prise concernant cette décision
- $0 \leq \frac{|D_c|}{|D_e|-|D_i|} \leq 1$. Ce critère inclut le précédent.

La recherche dichotomique

```
 1 int
 2 rechdicho (int t[], int length, int e)
 3 {
 4   int n = length - 1;
 5   int bas = 0, haut = n, milieu;
 6   int i = -1;
 7   do
 8     {
 9       milieu = (bas + haut) / 2;
10       if (e == t[milieu])
11             i = milieu;
12       else if (t[milieu] < e)
13             bas = milieu + 1;
14             haut = milieu - 1;
15     }
16   while ((e != t[milieu]) && (bas <= haut));
17   return i;
18 }
```

Decision Coverage

1. $T = \left\{ \begin{array}{c} < t_1 =< t = [1,2,3,4], \mathit{length} = 4, e = 2 > \\ < t_2 =< t = [1,2,3,4], \mathit{length} = 4, e = 10 > \end{array} \right\}$
   - Si pas de clause `else` dans le listing slide 5 $\Rightarrow$ T donne une couverture de blocs à 100% mais pas en termes de branches(2/3 décisions couvertes).

2. $T_2 = \left\{ \begin{array}{c} < t_1 =< t = [1,2,3,4], \mathit{length} = 4, e = 2 > \\ < t_2 =< t = [1,2,3,4], \mathit{length} = 4, e = 10 > \\ < t_3 =< t = [1..12], \mathit{length} = 12, e = 5 > \end{array} \right\}$
   - L'exécution sur t_3 donne le résultat inattendu de -1.
   - Combiens de décisions sont couvertes par t_3 ?
   - Quelle proportion de décisions sont couvertes par les tests ? par t_3 uniquement ?

Après correction, si on prend T_{def}, suffisant ?

3. $T_{def} = \left\{ < t_3 =< t = [1..12], \mathit{length} = 12, e = 5 > \right\}$

Condition Coverage

- La couverture des décisions n'implique pas automatiquement la couverture des conditions.
- Exemple
  - $T = \left\{ \begin{array}{r} < t_1 =< t = [1,2,3,4,5,6], \mathit{length} = 6, e = 5 > \\ < t_2 =< t = [1,2,3,4,5,6,7], \mathit{length} = 7, e = 2 > \end{array} \right\}$
  - Pour t_1 et t_2 on a

e !=t[milieu]	bas<haut	D
true	true	true
false	true	false

  -

	if	elseif	while
t1	ok	¬ ok	ok
t2	ok	¬ ok	ok
T	ok	ok	ok

- $< t_3 =< t = [1,2,3,4,5,6], \mathit{length} = 6, e = 2 >$ évalue à `false bas<haut` et met en évidence la faute(`bas<=haut`).

Condition Coverage II

- La couverture des conditions n'implique pas la converture des décisions.
- Exemple
  - $T = \left\{ \begin{array}{r} < t_1 =< t = [1,2], \mathit{length} = 2, e = -2 > \\ < t_2 =< t = [1,2], \mathit{length} = 2, e = 11 > \\ < t_3 =< t = [1,2,3,4,5,6,7], \mathit{length} = 7, e = 4 > \end{array} \right\}$
  -

	e !=t[milieu]	bas<haut	D
t1	true	false	false
t2	true	false	false
t3	false	true	false

- $< t_3 =< t = [1,2,3,4,5,6], \mathit{length} = 6, e = 2 >$ évalue D à *true* puis révèle la faute(`bas<=haut`). Dans l'exemple :faute révélée aussi avec $< t_3 =< t = [1,2], \mathit{length} = 2, e = 2 >$.

Condition/Decision Coverage Criterion

- Si la condition est simple alors équivalent à la couverture de la décision.
- $0 \leq \frac{|C_c|}{|C_e|-|C_i|} \leq 1.$
- Prise en compte des décisions et des conditions :$0 \leq \frac{|C_c|+|D_c|}{((|C_e|-|C_i|)+(|D_e|-|D_i|))} \leq 1.$

Multiple condition Coverage

- La couverture des décisions et des conditions ne signifie pas que toutes les combinaisons des conditions simples dans une condition composée ont été couvertes.
- Exemple : $D = (a < b)||(a > c)$
  - Si on dispose de tests correspondant aux lignes 1 et 4, des entrées pour les autres lignes pourraient révéler de nouvelles fautes.

	a<b	a>c	D
1	true	true	true
2	true	false	true
3	false	true	true
4	false	false	false

- Nbr de combinaisons de conditions dans un programme = $\sum_{i=1}^{n} 2^{k_i}$.

Modified condition/decision Coverage

- Chaque condition simple dans une une condition composée doit montrer qu'elle a un effet indépendant des autres conditions sur la sortie.

$c = c_1$ && c_2	c_1	c_2	c	commentaire
t_1	true	true	true	t_1 et t_2 couvre c_2
t_2	true	false	false	
t_3	false	true	false	t_1 et t_3 couvre c_1

- Requiert moins de tests que la couverture de toutes les combinaisons dans les conditions(croissance linéaire en fct de n =nbr de conditions.
- mais critère plus faible !

MC/DC Coverage II

- Construire un T pour $c = ((c1 \texttt{ and } c2) \texttt{ and } c3)$.

1.

Test	c_1	c_2	c_3	c
t_1				
t_2				
t_3				
t_4				

2.

Test	c_1	c_2	c_3	c
t_1		true	true	true
t_2		true	false	false
t_3		false	true	false
t_4				

MC/DC Coverage II

- Construire un T pour $c = ((c1 \text{ and } c2) \text{ and } c3)$.

3.

Test	c_1	c_2	c_3	c
t_1	true	true	true	true
t_2	true	true	false	false
t_3	true	false	true	false
t_4	false			

4.

Test	c_1	c_2	c_3	c
t_1	true	true	true	true
t_2	true	true	false	false
t_3	true	false	true	false
t_4	false	true	true	false

MC/DC Coverage III

- MC/DC requiert que
  1. chaque bloc est couvert ;
  2. chaque condition simple a pris les valeurs `true` et `false` ;
  3. chaque décision a pris toutes les sorties possibles ;
  4. chaque condition simple dans une condition composée c a montré qu'elle affectait de manière indépendante le résultat de c. Les tests sont choisis en conséquence.
- Remarques
  - Compromis à trouver entre jeu de tests minimal(temps d'exécution $<$?) et des tests simples dont le but est clairement identifié.
  - On a supposé pas de court circuits dans les conditions.

Prédicats d'un chemin

- A un chemin est associé un ensemble de prédicats(path predicate).
- Exemple(CFG slide 9)

Chemin	(Start,1,2,4,5,7,2,3,7,8,End)
Prédicats	1. $e == t[milieu] \rightarrow \texttt{false}$ 2. $t[milieu] < e \rightarrow \texttt{true}$ 3. $e! = t[milieu]$ && $bas <= haut \rightarrow \texttt{true}$ 4. $e == t[milieu] \rightarrow \texttt{true}$ 5. $e! = t[milieu]$ && $bas <= haut \rightarrow \texttt{false}$

- Les prédicats sont composés de
  - variables locales ;
  - des variables du vecteur d'entrée ;
  - de constantes.

Interprétation du prédicat

- L'interprétation du prédicat de chemin consiste à substituer symboliquement les opérations le long du chemin de telle sorte à n'exprimer les prédicats seulement en termes du vecteur d'entrée et de constantes.
- Exemple(pour le chemin slide 29)
  - $e \neq t[\frac{0+length-1}{2}]$
  - $(e \neq t[\frac{length-1}{2}])$ and $(\frac{length-1}{2} + 1 \leq length - 1)$
  - $e = t[\frac{(\frac{length-1}{2}+1)+(length-1)}{2}]$
- Le domaine du chemin sont les entrées qui forcent ce chemin.
  - Si les contraintes ne peuvent être toutes résolues, le chemin n'est pas faisable.
  - Le prédicat de chemin exprimé en termes du vecteur d'entrée partitionne le domaine d'entrée.

Substitution symbolique

```
1  int
2  fct (int x, int y)
3  {
4    int a;
5    a = y + 100;
6    if (x + a >= 0)
7      {
8        y = x + (y + a);
9        if (y > 0)
10         return 0;
11       else
12         return 1;
13     }
14   else
15     return -1;
16 }
```

- Le prédicat $x + a >= 0$ peut être réécrit $x + y + 100 >= 0$
- $y > 0$ est réécrit en $x + 2y + 100 > 0$